

Report Documentation Page				Form Approved OMB No. 0704-0188	
Public reporting burden for the 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 the 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. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.					
1. REPORT DATE 1996		2. REPORT TYPE		3. DATES COVERED 00-00-1996 to 00-00-1996	
4. TITLE AND SUBTITLE On the Systematic Design of Web Languages				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S)				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School ,Center for Information Systems Security Studies and Research (NPS CISR),Department of Computer Science,Monterey,CA,93943				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited					
13. SUPPLEMENTARY NOTES ACM Computing Surveys, Vol. 28, No. 2, pp. 315-317, June 1996					
14. ABSTRACT					
15. SUBJECT TERMS					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT Same as Report (SAR)	18. NUMBER OF PAGES 3	19a. NAME OF RESPONSIBLE PERSON
a. REPORT unclassified	b. ABSTRACT unclassified	c. THIS PAGE unclassified			

On the Systematic Design of Web Languages

Dennis Volpano

Geoffrey Smith*

“Semanticists should be the obstetricians, not the coroners, of programming languages.” -John Reynolds

Recently there has been phenomenal growth in the use of the Internet through the World Wide Web. Especially interesting is the emergence of so-called “web languages”, such as *Java*¹ [4, 5], which support the development of programs that can be downloaded to a client’s machine for execution by a web browser. This brings power and flexibility to web applications, but it also brings well-known end-point security problems, such as the threat of integrity and unauthorized disclosure attacks [1]. Yet, although security has been an issue in web language design, there appears to be no formal characterization of the kinds of security properties that one can expect of programs written in these languages.

A web language should, we believe, be designed so that all programs are guaranteed to satisfy explicitly-stated security properties. To this end, we advocate a more systematic approach to the design of secure web languages. One begins with a core language for which some set of desired security properties can be shown to hold with respect to a formal semantics. The language is then incrementally extended to include new features, as necessary. At each step, the designer has an obligation to establish that the security properties have been preserved.

For example, one kind of attack a web language should guard against is a *disclosure attack* where a malicious program downloaded by a client attempts to make the contents of certain private files public. In response, one might use discretionary access control mechanisms to simply forbid access to sensitive information, or to limit access to output devices. But such restrictions would likely render many applications useless.

For example, a downloaded mail tool would need access to a client’s sensitive mailfolder and to a nonsensitive, publicly-writable directory like `/tmp` for message composition and replies. However, granting such access rights requires care, since a mail tool might now be able to copy a client’s entire mailfolder to `/tmp`.

* Authors’ addresses: D. Volpano : Computer Science Department, Naval Postgraduate School, Monterey, CA 93943, USA; email: volpano@cs.nps.navy.mil; G. Smith: School of Computer Science, Florida International University, Miami, FL 33199, USA; email: smithg@fiu.edu.

¹Java is a registered trademark of Sun Microsystems Inc.

There is a form of static analysis, however, that can be applied to programs to protect against disclosure attacks while allowing utilities like mail tools to access files and directories at different sensitivity levels. It is called *secure information flow* analysis and it was pioneered by Dorothy Denning [2]. We briefly describe how a type system can be imposed on a simple imperative language to guarantee secure information flow. In part, we illustrate how well type theory can be applied to address a security problem, but our main point is that the security theorem stated below actually *guides* the language designer in evaluating potential language extensions.

To begin with, we suppose that we have a partial order of security classes τ . For simplicity, we may consider just two classes, L (low) and H (high), where L represents nonsensitive information and H represents sensitive information. There is a natural subtyping among these classes, based on the observation that L information can safely be considered to be H information, but not vice versa. Thus, $L \leq H$.

To type the phrases of the language we use three forms of types: τ (for expressions), $\tau \text{ var}$ (for variables), and $\tau \text{ cmd}$ (for commands). For example, we might have $x : L \text{ var}$ to indicate that x is a variable that holds values of type L . The rule for typing an assignment $x := e$ requires that $x : \tau \text{ var}$ and $e : \tau$. So if $y : H \text{ var}$, the assignment $y := x$ is legal, since the value of x can be coerced to H , but the assignment $x := y$ is illegal. Also, the assignment $x := 17$ is legal, since literals are taken to have type L .

More subtly, we also want the conditional

if *even*(y) **then** $x := 0$ **else** $x := 1$

to be illegal, because this results in an *implicit* flow of information from y to x . To achieve this, we give commands types of the form $\tau \text{ cmd}$. A command c has type $\tau \text{ cmd}$ only if every variable assigned to in c has type at least τ . Note that cmd is an antimonotonic type constructor: $\tau \text{ cmd} \leq \tau' \text{ cmd}$ iff $\tau' \leq \tau$. The rule for typing assignments says that if $x : \tau \text{ var}$ and $e : \tau$, then $x := e : \tau \text{ cmd}$. The rule for typing conditionals says that if $e : \tau$, $c : \tau \text{ cmd}$, and $c' : \tau \text{ cmd}$, then **if** e **then** c **else** $c' : \tau \text{ cmd}$. The conditional **if** *even*(y) **then** $x := 0$ **else** $x := 1$ is illegal, then, because *even*(y) only has type H , and $x := 0$ does not have type $H \text{ cmd}$.

The value of this type system is given by a security theorem, which is a property known as *noninterference* [3]. It asserts, intuitively, that no information can flow from H variables to L variables [6]:

Theorem. Suppose that program c is well typed, and μ and ν are two memories that agree on all L variables. If c terminates successfully when run on memories μ and ν , yielding memories μ' and ν' , respectively, then μ' and ν' agree on all L variables.

Many challenges remain before a truly secure web programming environment can be developed. However, we believe that theorems of this kind are necessary in any web language that hopes to provide secure web computing. By taking security into account at the outset of language design, one can

get a handle on what kind of language is possible if a particular security property is expected to hold. For instance, notice that the security theorem above requires c to terminate successfully when run on memories μ and ν . This means that if it should terminate abnormally on μ or ν then the potential for interference exists in any language with the ability to detect such status. Unfortunately this capability can be found in languages that return the exit status of a process or generate and handle exceptions. So extending the simple language with such features can be dangerous. The point here is that the systematic approach leads us to the discovery.

References

- [1] Bank, J., Java Security, available via <http://www-swiss.ai.mit.edu/~jbank/javapaper/javapaper.html>, December 1995.
- [2] Denning, D. and Denning, P., Certification of Programs for Secure Information Flow, *Communications of the ACM*, Vol. 20, No. 7, pp. 504–513, July 1977.
- [3] Goguen, J. and Meseguer, J., Security Policies and Security Models, *Proc. 1982 IEEE Symposium on Research in Security and Privacy*, pp. 11–20, April 1982.
- [4] Gosling, J. and McGilton, H., The Java Language Environment: A White Paper, Sun Microsystems Inc., available via <http://java.sun.com/docs/JavaBook.ps.tar.Z>, May 1995.
- [5] van Hoff, A., Shaio, S., and Starbuck, O., *Hooked on Java*, Addison-Wesley, 1995.
- [6] Volpano, D., Smith, G. and Irvine, C., A Sound Type System for Secure Flow Analysis, submitted to *J. of Computer Security*, February 1996.