AFRL-IF-RS-TR-2003-287
Final Technical Report
December 2003

SYSTEMS ASSURANCE

Syracuse University

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2003-287 has been reviewed and is approved for publication

APPROVED: /s/
W. JOHN MAXEY
Project Engineer

FOR THE DIRECTOR: /s/
WARREN H. DEBANY, Jr., Technical Advisor
Information Grid Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	Dec 03	Final Jul 00 – Sep 01

4. TITLE AND SUBTITLE

SYSTEMS ASSURANCE

5. FUNDING NUMBERS

C - F30602-01-1-0508
PE - 62301E
PR - OIPG
TA - 32
WU - P5

6. AUTHOR(S)

Steve J. Chapin

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Systems Assurance Institute
Syracuse University
Syracuse NY 13219

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/IFGB
525 Brooks Rd
Rome NY 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

AFRL-IF-RS-TR-2003-287

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: W. John Maxey, IFGB, 315-330-3617, maxeyw@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 Words)*

The Systems Assurance contract supported fundamental security and assurance research at the Center for Systems Assurance at Syracuse University. This research included intrusion detection mechanisms, software resilience, and steganography. The intrusion detection focused on packet filtering and intrusion sensing using programmable (smart) network interfaces, and buffer overflow detection and prevention. The Computational Resiliency project focused on novel applications of replication, migration, agreement protocols, and group communication to increase the assurance of scientific applications. The Protocol Steganography project defined a new model of information hiding in network flows. The project resulted in four software prototypes and a total of five refereed publications.

14. SUBJECT TERMS

information warfare, quality of service, anomaly detection

15. NUMBER OF PAGES

33

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Contents

1 Introduction

The Systems Assurance contract supported research into three major areas at the Center for Systems Assurance at Syracuse University: Protocol Steganography, Intrusion Detection and Prevention, and Computational Resiliency. Protocol Steganography is described in section 2. The intrusion detection and prevention sub-project had two portions: smart network interfaces and buffer overfbw prevention. The smart NIC project produced a software prototype implemented as a kernel module under Linux, and is described in section 3; the buffer overfbw portion is described in the attached papers. The Computational Resiliency project is described in 4 and in the attached papers.

2 Semantics-Preserving Application-Layer Protocol Steganography

Protocol steganography allows users who wish to communicate secretly to embed messages within other messages. These secret messages can be used for anonymous communication for purposes ranging from entertainment to protected business communication or national defense.

In this section, we describe our approach to application-layer protocol steganography, and describe how we can embed messages into commonly used TCP/IP protocols such as SSH and HTTP. We also introduce the notion of semantics preservation, which ensures that messages still conform to the host protocol, even after embedding. Strong semantics preservation ensures that the meaning of the message is unchanged, while weak semantics preservation only guarantees the less stringent condition that the message be semantically valid.

To demonstrate the effi cacy of our approach, we have implemented protocol steganography within the Secure Shell (SSH) protocol.

Steganography, from the Greek "covered writing", refers to the practice of hiding information within other information [13]. Historically, notions of classical steganography can be found even centuries before Christ. In recent years, steganography has become digital: the favorite media for information hiding are images, music scores, formatted and written text, digital sounds, and videos. This evolution of steganographic techniques has received particular attention, as have the security and robustness of such methods [1, 3, 16, 18, 19]. Traditionally, most steganographic systems relied on the secrecy of the encoding system [21]. At present, the security of a stegosystem depends on how well it conceals the existence of a hidden message and in the secrecy of a key, if used, for embedding the message. Protocol steganography is the art of embedding information within messages and network control protocols used by common applications [6].

An important consideration in the embedding process is whether it is semantics-preserving, i.e., whether the resulting message still conforms to the protocol specifi cation. That property guarantees that if the message is interpreted at any point during its transmission, it will produce meaningul results. In addition to that, semantic preservation in modifi ed messages helps to make them indistinguishable from unmodifi ed cover messages. Using protocol steganography, we can embed information in overt channels,

in contrast to the use of covert channels, which allow signaling mechanisms to occur where no explicit communication path exists. Advantages of protocol steganography include achieving greater bandwidth in hidden communication as well as taking advantage of the most widely-used network protocols.

We define two levels of semantics preservation, both of which imply that the stego-message is a correct message within the protocol. *Weak semantics preservation* means that the stego-message, while legal, has a different meaning than the original cover message. *Strong semantics preservation* means that the stego-message has the same meaning as the original cover.

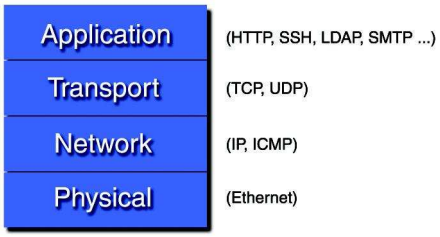

Figure 1: TCP/IP Protocol Suite Layers.

Networking protocols are divided into multiple layers, as shown in Figure 1. The physical layer is responsible for communicating with the actual network hardware (e.g., the Ethernet card), dealing with the format of the bits on the wire. Therefore, it is tied to the local network technology, such as Fast Ethernet or 802.11b wireless. The network layer handles routing, and it is the IP layer of the TCP/IP protocol suite. The network layer is invisible to user programs. The transport layer handles the quality-control issues of reliability, flow control, and error correction. The TCP/IP protocol suite defines two widely-used transport protocols: UDP and TCP[1].

There are several application protocols in the TCP/IP suite, including SMTP (for email service), FTP (for file transfer), SHH (for secure login), LDAP (for distributed directory services), and HTTP (for web browsing, which alone accounts for approximately 70% of all Internet traffic).

The most obvious way of hiding information within messages is to place data in unused or reserved fields of protocol headers or trailers. However, that method of steganography is easy to detect using simple intrusion detection systems, or is susceptible to traffic analysis, which makes it insecure and not robust. Even if analyzing the content of the hidden information becomes impossible, perhaps due to encryption, this approach is weak. Our techniques for protocol steganography aim to achieve strong steganography, wherein the system is both secure and robust.

A *secure* stego system can withstand an opponent that understands the system (or even has grounds for suspicion), meaning that the opponent cannot determine with a high degree of certainty the existence of the communication. A *robust* system can withstand an active attack, where the adversary makes legal (strong semantics-preserving)

[1]Other transport protocols, such as the Reliable Datagram Protocol (RDP), are defined but not widely used.

changes to the message. Given those goals and the intention to provide means of private communication, our approach to protocol steganography focuses mainly on transport layer protocols and application layer protocols, although other protocols at different layers of the TCP/IP protocol suite could also be considered. In particular, this paper describes how protocol steganography is feasible using the SSH protocol as proof-of-concept.

There are many potential applications for protocol steganography, mostly when information hiding is used in order to achieve private communication [27] and, in some cases, anonymity and plausible deniability, such as environments where censorship polices restrict web access [9]. More specifically, protocol steganography seems to be appropriate for environments where unobtrusive communications are required [13]. For example, in the military and intelligence agencies, even if the content of the communication is encrypted, the detection of extra communication with a battlefield could represent a sign of attack. Hiding information inside regular Internet traffic, such as browsing results, will avoid the need for extra communication, thereby giving no indication to one's adversaries that something is about to happen. On the other hand, considering a framework where the agents that wish to communicate secretly are not necessarily the initiators of the communication, the ability to embed messages in a variety of TCP/IP protocols gives us a much higher likelihood of being able to transmit the secret message within a reasonable time bound.

2.1 Framework for Secret Communication

Our model for protocol steganography involves two agents that wish to communicate secretly through channels of Internet traffic in a hostile environment (see Figure 2). The agents *A* and *B*, named for Simmons's [24] famous prisoners Alice and Bob, take advantage of a communication path already in place between themselves or two arbitrary communicating processes, the *sender* and *receiver*. We assume that Alice wishes to pass a message to Bob, and may in fact be operating in an environment over which their adversary has administrative control (such would be the case if Alice were an undercover investigator or intelligence operative).

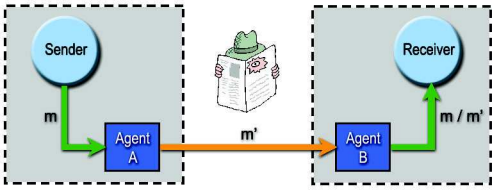

Figure 2: Framework for Secret Communication.

Consequently, two scenarios are possible depending on whether or not agent A and B are the same as the sender and the receiver, respectively. In the first case, agent A and agent B are trying to hide secret information in some of their own harmless messages, as in traditional steganography models. They both run a modified version of the communicating software that allows them to convey the secret message. In the

second case, agent A and agent B are placed somewhere along an arbitrary communication path, modifying the message in transit to hide meaningful information. In short, both the internal agent and the external confederate might be either end points of the communication or middlemen, acting to embed and extract the hidden message as the data passes them in the communication stream. In fact, the receiving middleman has the option of removing the hidden message, thus restoring and forwarding the original message. The midpoints where agents A and B can alter the message might be within the protocol stack of the sending and receiving machines (which is still distinct from the sending process), or at routers along the communication path. These arbitrary boundaries are indicated by the dashed boxes in Figure 2.

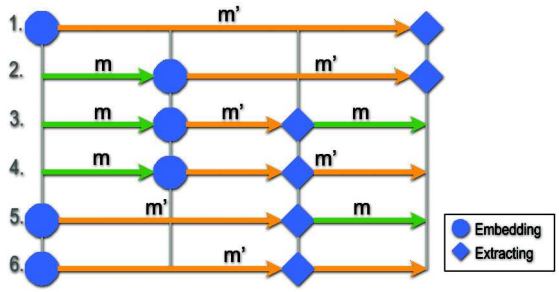

Figure 3: Message Paths.

Considering all combinations of internal agents and external confederates and all different points where the message can be altered yields six different roles for the agents, as shown in Figure 3. In this discussion, following the established information hiding terminology [20], agent A executes the *embedding* process and agent B the *extraction* process, represented in the picture as a circle and a diamond, respectively. As pointed out by Pfi tzmann [20], the embedding and extracting processes required the use of a *stego-key*, not shown in the picture. The cover (i.e. the original harmless message) is m, and the *stego-message* (i.e. the message with steganographic content) is m'.

1. *Agent A* acts as sender and *agent B* as receiver—the message along the entire path is m'.

2. *Agent A* is a middleman, embedding information to the message on its way, and *agent B* acts as *receiver*—the message from the *sender* to *agent A*'s location is m, while from there to the endpoint is m'.

3. Both agents are middlemen, and *B* restores the message to its original form—the message from the *sender*'s point to where *agent A*'s location is m, from *A*'s to *B*'s is m', and from there to the endpoint is m again, since extraction of the hidden content occurred at *B*'s location.

4. Both agents are middlemen, but *agent B* does not restore the message—the message from the *sender*'s point to the *agent A*'s location is m, and from *A*'s to the *receiver*'s point is m', with the hidden information extracted at *B*'s location.

5. *Agent A* is acting as *sender*, with *B* as a middleman extracting the embedded information and restoring the original message—the message from the initial point to *agent B*'s location is m', and from *B*'s location to the *receiver*'s point is m.

6. *Agent A* is acting as sender and *agent B* is a middleman extracting the hidden information without restoring the message as it travels to the *receiver*—the message from the end to end is m', but *B* gets the hidden content somewhere before the message reaches its destination.

Not every one of these scenarios might be realistic, but cases 1 and 3 certainly are. Therefore, they have been the focus of this study. All the options where the hidden content is extracted but the message is not restored seem very risky; in particular, case 4 where the message seen by the receiver is clearly different from that seen by the sender, neither of whom are the agents communicating secretly.

Having the agents acting as middlemen in the communication stream provides several advantages, because any packet that will flow past the locations where agents A and B are can be modified (as long as a semantics-preserving embedding function is available for the transport or application protocol in that packet). The idea is to lower our susceptibility to traffic analysis, as there is no longer a single source/sink for the stego-messages, and there is no specific protocol used. This also allows us to achieve a higher bit rate as well as privacy, anonymity, and plausible deniability, in some cases. An ideal situation would be that agent A is located on the last router inside the sender's domain (the egress router for that domain), and agent B is located on the first router outside the domain (the ingress router). This will have m' "on the wire" for the minimum possible time, also lowering the probability of detection.

2.1.1 Adversary Models

Depending on the goals of steganalysis, adversaries can be *active* or *passive* [20]. Passive adversaries observe the communication in order to detect stego-messages, find out the embedded information, if possible, and prove to third parties, when the case requires it, the existence of the hidden message. Active adversaries attempt to remove the embedded message without changing the stego-message significantly, i.e., they attempt to provide strong semantic preservation. In some cases, active adversaries do not need to verify the existence of the message before they attempt to block any secret communication, thus playing appropriately with the bits of the messages that pass through them is enough (e.g., zeroing unused header fields).

In steganography systems, adversaries can be passive or active [2], while in watermarking and fingerprinting systems, generally, only active adversaries raise concern. However, most of the literature in stegosystems deals with passive adversaries. For the purposes of this study, both passive and active adversaries are taken into account, because of hostility of the Internet environment, the constant improvement of routers and firewalls, and the goal of developing not only secure, but also robust, steganography techniques.

2.2 Related Work

Handel and Sandford [11] reported the existence of covert channels within network communication protocols. They described different methods of creating and exploiting hidden channels in the OSI network model (see Figure 4, based on the characteristics of each layer. In particular, regarding to the application layer, they suggested covert messaging systems through features of the applications running in the layer, such as programming macros in a word processor. In contrast, the protocol steganography approach studies hiding information within the information within messages and network control protocols used by the applications, not inside images transmitted as attachments by an email application, for example. They also considered techniques of embedding information that require substituting existing modules of the source code that implements a particular layer, and some that not. In a similar order of ideas, when agent A and agent B act as sender and receiver, respectively, some application modules will be replace for embedding and extracting secret messages.

Figure 4: The OSI Idealized Network Model Layers.

Examples of implementation of covert channels in the TCP/IP protocol suite (see Figure 1) are presented by Rowland [23], Project Loki [22], Ka0ticSH [12], and more deeply and extensively by Dunigan [7]. These researchers focused their attention in the network and transport layers of the OSI network model (shown in Figure 4). In spite of that, Dunigan [7] did point out in his discussion of network steganography that application-layer protocols, such as telnet, ftp, mail, and http, could possibly carry hidden information in their own set of headers and control information. However, he did not develop any technique targeting these protocols. More in detail, Rowland [23] implemented three methods of encoding information in the TCP/IP header: manipulating the IP identification field, with the initial sequence number field, and with the TCP acknowledge sequence number field "bounce." Dunigan [7] analyzed the embedding of information, not only in those fields, but in some other fields of both the IP and the UPD headers as well as in the ICMP protocol header. He based his analysis, mainly, in the statistical distribution of the fields and the behavior of the protocol itself. The Project Loki [12, 22] explored the concept of ICMP tunneling, exploiting covert channels inside of ICMP_ECHO traffic. All these approaches, without minimizing their importance, suffer from two problems: low bandwidth and simplicity of detection or defeat with straightforward mechanisms.

One such mechanism is described in Fisk et al. [10]. Their work defines two classes of information in network protocols: *structured* and *unstructured* carriers. Structured

carriers present well-defined, objective semantics, and can be checked for fidelity en route (e.g., TCP packets can be checked to ensure they are semantically correct according to the protocol). On the contrary, unstructured carriers, such as images, audio, or natural language, lack objectively defined semantics and are mostly interpreted by humans rather than computers. The defensive mechanism they developed aims to achieve security without spending time looking for hidden messages: using active wardens they defeat steganography by making strong semantic-preserving alterations to packet headers (e.g. zeroing the padding bits in a TCP packet). The most important considerations to their work related to protocol steganography are the identification of the cover-messages in used as structured carries, and the feasibility of similar methods of steganalysis that target application-layer protocols.

2.3 A Case Study: SSH

The SSH protocol is defined by the Internet drafts [28, 29, 30, 31] of the Internet Engineering Task Force (IETF). It is a "protocol for secure login and other secure network services over an insecure network" [30]. The main goal of the protocol is to provide server authentication, confidentiality, and integrity with perfect forward secrecy. There are several, both commercial and open-source, implementations of SSH. The latest version of the protocol is SSH2 and, being version most widely and currently used, it is the one object of this study.

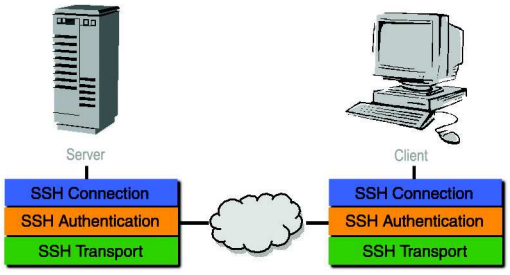

Figure 5: SSH2 Protocol Architecture.

The SSH2 protocol consists of three major components as illustrated in Figure 5:

- **Transport Layer Protocol**

  It provides server cryptographic authentication, confidentiality through strong encryption, and integrity plus, optionally, compression. Typically, it runs over a TCP/IP connection listening for connections on port 22.

- **User Authentication Protocol**

  It authenticates the client-side user to the server. It runs over the transport layer protocol.

- **Connection Protocol**

It multiplexes the encrypted tunnel into several logical channels. It runs over the user authentication protocol. It provides interactive login sessions, remote execution of commands, forwarded TCP/IP connections, and forwarded X11 connections.

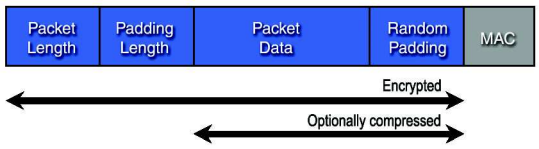

Figure 6: SSH2 Binary Packet Protocol.

In particular, the Transport Layer protocol defines the *Binary Packet Protocol*, which establishes the format SSH packets follow (see Figure 6). According to the specification [31], each packet is composed of five fields:

Packet Length

Number of octets representing the length of the packet data, not including the MAC or the packet length itself.

Padding Length

Number of octets representing the length of the padding.

Packet Data

The payload, the actual content of the message. If compression has been negotiated, this field is compressed.

Random Padding

An arbitrary-length padding, such as the total length of `packet length + padding length + packet data + padding` is a multiple of the cipher block size or 8, whichever is larger.

MAC (message authentication code)

When message authentication is negotiated, it contains the MAC octets. Only initially, the value of the MAC algorithm is none (before authentication).

An SSH client and server start the communication negotiating an encrypted session, followed by client password authentication. Establishing the encrypted session includes exchanging keys and negotiating algorithms (key exchange algorithms, server host key algorithms, encryption algorithms, MAC algorithms and compression algorithms) as well as determining a preferred language. The password authentication process is similar to the one in any remote login application, with the advantage of being more secure due to encryption. The password is prone only to key logging.

The main reason for selecting the SSH protocol as a case of study is the randomness of the content of its packets, which is an excellent factor when trying to blend hidden content in what is considered a "normal" traffic. In addition to that, it is widely used but encrypted, fact that by itself can keep adversaries away from trying to analyze its content, and, as with many other protocols, and pointed out by Barrett and Silverman [4] its design does not attempt to eliminate covert channels.

2.3.1 SSH Potential for Information Hiding

There are several potential places where information can be hidden without breaking the SSH protocol. Four of those ways of steganography are described below.

- **Generating a MAC-like Message**

  As shown in Figure 6, the SSH2 specification defines the fields:

uint32	packet_length
octet	padding_length
octet[n1]	payload; n1 = packet_length - padding_length - 1
octet[n2]	random padding; n2 = padding_length
octet[m]	mac (message authentication code); m = mac_length

  where `octet[m]` contains the computed MAC. The MAC is normally computed with the previously negotiated MAC algorithm using the key, the sequence number of the packet, and the unencrypted (but compressed, if compression is required) packet data. The MAC algorithms defined by the protocol are hmac-sha1, hmac-sha1-96, hmac-md5, and hmac-md5-96 whose digest lengths vary from 12 to 20 octets. Therefore, generating a MAC-like message will open the possibility to transmit from 12 to 20 octets of information.

- **Generating Random Padding-like Message**

  Basically, this idea is similar to the previous one, but stores the message in the random padding field.

- **Hiding information in as part of the Authentication Mechanism**

  The following is the defined format for the authentication request established by the SSH authentication protocol:

octet	SSH_MSG_USERAUTH_REQUEST
string	user name (in ISO-10646 UTF8 encoding)
string	service name (US-ASCII)
string	method name (US-ASCII)
...	method-specific data

  The first four fields cannot be modified if we are to conform to the protocol, but there is the possibility of embedding some information in the `method-specific data` field and still retaining the required semantics.

  The format of the response to the authentication request looks like this:

octet	SSH_MSG_USERAUTH_FAILURE
string	authentications that can continue
boolean	partial success

where `authentications that can continue` is a comma-separated list of authentication method names.

When the server accepts authentication, the response is:

octet	SSH_MSG_USERAUTH_SUCCESS

but only when the authentication is complete.

We defined a handshake between client and server about what method/type of steganography is going to be used in the MAC-like message generation or random padding-like message generation. The idea is to take advantage of the parameter exchange done by the regular authentication mechanism. The two agents, A and B, just need to agree on a covert meaning for the method-specific data sent as an option. Moreover, the protocol recommends the inclusion in the list of `authentications that can continue` only those methods that are actually useful; it also says that even if there is no point in clients sending requests for services not provided by the server, sending such a request is not an error, and the server should simply reject it. Thus, sending a bogus list of authentications that can continue is not an error.

Another advantage of using the authentication mechanism for hiding data is the fact that the plain text would be encrypted, so no matter what is sent in the string fields, it will not be subject to traffic analysis.

- **Adding additional encrypted content to the packet**

  The previous two approaches are only effective when the agents are the same as the sender and receiver (see Figure 2). But the following idea explores having, agent A and agent B, located somewhere along the line of communication of two arbitrary entities that produce SSH traffic.

  Intercepting the traffic and inserting an encrypted-like portion between the encrypted part of the packet and the MAC is an option, as detailed in Figure 7. The inserted portion consists of two parts: the hidden message itself and a "magic" number that tells agent B there is a hidden message in that SSH packet.

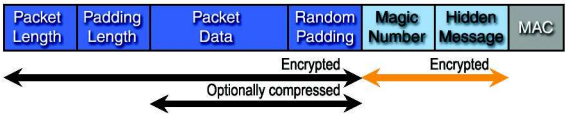

Figure 7: Adding an encrypted portion with a hidden message to a regular SSH packet.

This option offers the advantage of having agents communicating secretly anywhere and using any SSH traffic, but it requires careful study of its susceptibility to traffic analysis. Traffic analysis might indicate that those modified SSH

packets are longer than normal, which will indicate suspicion of being a stego-message and, ultimately, compromise the security of the method. The SSH protocol standard states that any implementation must be able to handle packets with uncompressed payload length of 32,768 octets or less, being the maximum total packet size 35,000 (including length, padding length, packet data, padding, and MAC). Therefore, the length can vary widely. How much variance there actually is in SSH packet length in typical traffic is an open question. Another question that needs to be answered is where along the communication stream the agents can be placed so an adversary analyzing the traffic cannot perceive the length difference (i.e., the adversary is not able to get both the original packet and the packet containing the stego-message). Another issue with this approach is that the "magic" number needs to be of certain minimum length in order to minimize the probability of having the magic number appear naturally in the data stream. We have chosen a two octet magic number for our initial implementation, but this introduces a one in 64k chance that we will incorrectly interpret a cover-message as a stego-message.

2.3.2 Prototype Implementation

The prototype of secret communication was implemented in C, modifying the version 3.4. of OpenSSH (http://www.openssh.org), a popular open-source SSH product. The approach for information hiding selected was generation of message that looks like a MAC, which is the first one of the potential cases described in the previous section. In consequence, this implementation assumes that the agents secretly communicating, *agent A* and *agent B*, act as sender and receiver, respectively. That corresponds to case 1, described in Section 2.1 and illustrated in Figure 3.

In order to simulate the randomness of the MAC, the embedded messages are previously compressed and then encrypted. The modified version of the SSH client reads the content to be embedded from a file compressed with GZip (http://www.gzip.org/) and encrypted with the GNU Privacy Guard software (http://www.gnupg.org/), using the Blowfish algorithm. It embeds exactly the same amount of octets reserved for computing the MAC according to the previously negotiated algorithm during the client-server handshaking. The technique used in the stegosystem is a cover generation method, which involves the generation of digital objects with the purpose of being cover for a secret communication [13]. Basically, when substituting the original MAC with the stego-message, a cover is generated, a MAC-like hidden message. At the receiving end, the modified version of the SSH server ignores recomputing the MAC and comparing it with the one gotten from the client, since the server is acting as agent B, and gets the MAC-like message and saves it into a file.

During an SSH session, once encryption has been negotiated and authentication performed, SSH transmits a packet for every keystroke. That means for every key typed a packet in the binary format is sent, and that implies a MAC is computed for every keystroke. Generating a MAC-like message for every keystroke opens a great opportunity for secret communication through an overt channel, since as much hidden data as the MAC length can be transmitted with every keystroke. More in detail, OpenSSH uses the following C structure to store the MAC:

```
struct Mac {
  char *name;
  int enabled;
  const EVP_MD *md;
  int mac_len
  u_char *key;
  int key_len;
};
```

where `mac_len` represents the length of the MAC as specified in the standard. Depending on the MAC algorithm negotiated, this value is between 12 and 20 octets. The same `mac_len` was used to generate the stego-messages. Therefore, at least 12 octets of information can be sent with every keystroke, once a MAC-like message is built.

The implementation was a proof-of-concept that illustrated what could be done in, for example, a scenario where a military base regularly connects with computers from other government agencies using means of secure login and encryption. In that sense, even if the communication is subject to traffic analysis, a traffic increase in critical situations will not be observable because the agents can camouflage special commands within the regular communication traffic. The adversary would not be able to guess they are running their own version of OpenSSH.

2.3.3 Discussion

The implemented approach, although simple, represents a proof-of-concept of the idea of application-layer protocol steganography. A stego-message is embedded into a packet without altering the semantics established by the protocol standard. Moreover, it looks "normal" to simple traffic monitoring. However, several issues need to be discussed and some other requires further exploration.

The first issue of concern is the impossibility of verifying the actual payload of the message was correctly transmitted, as a consequence of replacement of the MAC. Information about the error rates in transmission of SSH packets will be useful for better understanding the validity of this approach. However, augmenting a short MAC could be another way of the same idea of using the MAC to embed secret information. Since the SSH specification indicates that the length of the MAC can be between 12 and 20 octets, depending on the algorithm, it would be possible to select an algorithm with a short MAC and pad the stego-message to it. For example, if the `hmac-md5-96` algorithm [31], which computes a MAC of 12-octet length, is used, we can add 8 octets of secret information to each packet, bringing the pseudo-MAC up to the 20-octet limit. Of course, for this approach to work, the agents A and B must agree in advance on what algorithm to use, but that is very simple to achieve through the authentication mechanism. Moreover, when they are not planning to communicate secretly, agent A and agent B can choose to use the `hmac-sha1` algorithm, which computes a MAC of length 20, so the total length of their average SSH packet does not raise suspicion.

If robustness is defined as the impossibility of removing the stego-message without destroying the cover message [13], the embedding of a MAC-like message is robust.

An active adversary cannot recompute the MAC without knowledge of the encrypted payload of the packet, the keys, and the algorithms used. Therefore, any change on the MAC will be taken at the receiving end as a signal of existence of a middleman in the communication stream. SSH will issue a warning and the session will be interrupted. In a similar order of ideas, if the attacker modifies one of the MAC-like stego-messages, it will be easy to detect because of the encryption and compression. If the hidden message is not meaningful to agent B, a warning and action similar to the case of a corrupted MAC can be taken. Therefore, due to the behavior of the protocol, an active adversary cannot attack the stegosystem without being noticed and also disrupting legitimate SSH traffic. In this particular case, the minimal requisite fidelity pointed out by Fisk et al. [10] (degree of signal fidelity that is both acceptable to end users and destructive to cover communications) does not apply since the MAC cannot be corrupted to be acceptable.

It seems to be some controversy in the field about what is the better way of defining a perfectly secure steganography system, as reported by Moskowitsz et. al [17] and Katzenbeisser and Petitcolas [14]. However, information theory and the ideas of security taken from cryptography are today considered as the "right" approach to secure steganography. Most of the information-theoretical definitions [5, 2, 15, 32] and some game-theoretical definitions [8] of secure stegosystems assume prior knowledge of the distributions of the covers in order to quantify the information a passive adversary can gather from observing the communication channel. Our assumption made in the implementation, regarding the uniformity of the distribution of both cover and stego-messages, requires more detailed study. That study would involve estimation of the probabilistic model of the cover as well as the stego, and performing statistical tests to prove the randomness of the hidden message. It is not enough to say that the approach is secure based semantic preservation; information-theoretical analysis must be done.

2.4 Future Work

To this point, very little work has been done in exploring the distribution model of the covers and the stego-messages. Gathering enough data to estimate those models, in order to verify that a passive adversary with the knowledge of the distributions and the power to compare them cannot still determine the existence of a hidden message, is the next step.

The presented implementation of SSH protocol steganography involves only the first case of those described in Figure 3; we are currently developing cases where the agents act as middlemen. For that, we are implementing a Packet Transmogrifier[2] (PT): software that embeds a message into an arbitrary stream of packets, and later extracts that hidden message. The PT will either run within an OS kernel or on a router along the communication path between the two agents—agent A will use the PT to transform *m* into *m'*, and agent B can use the PT to reverse the transformation.

In principle, the idea of building the packet transmogrifier is conceived as a combination of several individual packet transformers (each of which could be used by an

[2]With appropriate apologies and thanks to Bill Watterson, creator of "Calvin and Hobbes" [25].

individual application to embed a message in a data stream). This will give us the flexibility of embedding hidden messages in packets of multiple types corresponding to different protocols, and with a variety of sources and destinations.

The embedding functions that the PT will carry are of a great deal of interest. We are investigating several approaches, depending on the protocol. For example, it seems plausible to use mimic functions [26, 27] to tailor the distributions of text content, resulting from browsing queries, in regular HTTP traffic; we can also embed data in HTTP cookies, DNS traffic, MIME data, etc.

Furthermore, we are looking into ways of maximizing the bandwidth of the secret communication. Towards that issue, we are searching for algorithms of the form $m = f(p)$, where p is the packet given as input to the transmogrifier, and m is the hidden message. In other words, rather than embedding a secret message, we search for an extraction algorithm that would produce m if given p, and embed a representation of that algorithm within the packet. While this is an impractically hard problem if we are forced to find an embedding (really, the corresponding extraction) for a complete, arbitrary message into an arbitrary packet in the general case, we can reduce the complexity of the problem in three ways:

1. having a small family of extraction functions from which we choose, and only embed enough information to distinguish which member of the family should be used for that packet,

2. not requiring that we extract data from every packet, and

3. not requiring that we extract the full message from a single packet.

By relaxing the third condition sufficiently, we can all but guarantee that we can extract at least one octet from almost any packet. Because a prime consideration in the effectiveness of the transmogrifier is its per-packet latency, we are first considering simple extractions. When we have obtained performance measurements, we will devise more complex and diverse algorithms and analyze the overall effectiveness of the approach. There is a natural tension between the achievable bandwidth and the ease of finding an embedding (richer embeddings, yielding higher bandwidth, will be harder to find, and therefore increase the latency of the packets, which adversely affects network performance).

The most direct approach to embed the choice of algorithm is to assign each algorithm from the family an index code and to embed that code. Given the small number of algorithms we envision, this is a trivial amount of information (which could be carried in the MAC of an SSH packet, or embedded in a cookie within an HTTP request).

2.5 Conclusion

In this paper, we have described semantics-preserving application-layer protocol steganography, and have presented methods for embedding secret messages in application-layer protocols. We have developed the notions of strong and weak semantics preservation. Our approach has several advantages over prior work:

- Because of its applicability to a wide range of protocols, we can embed messages in the vast majority of network traffic on the Internet.

- The use of non-source stego (en route embeddings and extractions) increases the available bandwidth and complicates traffic analysis because of the ability to choose traffic from a variety of senders and receivers.

- Semantics preservation dramatically increases the security of our steganography.

As a proof-of-concept, we implemented end-to-end protocol steganography in the SSH2 protocol. The software may be obtained from the authors. In the near future, we will expand our family of embedders/extractors to include HTTP, and will complete implementation of the Packet Transmogrifier. This will allow us to perform on-the-fly message embedding and extraction while a packet is en route. We will also perform further analysis on the distributions of our covers and stego-messages.

3 Interface-Based Intrusion Detection

The aim of the Interface-Based Intrusion Detection (IBID) project is to develop purely-local detection mechanism against network intrusion. Our current approach is to build a rule-based system suitable for embedding in a secure network interface card. Our first implementation is a Virtual Smart NIC for the Linux operating system, where we have interposed a control layer between the interface card and the kernel. Our layer replaces the normal Linux interrupt handler, and diverts all network traffic (both incoming and outgoing) to our filtering module.

3.1 Design Goals

IBID System is different from other intrusion detection systems or packet filters in the following aspects:

Locality Designed to be directly embedded into a network card, once IBID System detects an intrusion, its countermove decision is immediately made and executed without any delay. The system is not only cable of detecting intrusions, but also effectively coping with them.

Flexibility Although the IBID System examines a network packet at a time, it differs from packet filters in the flexibility of rules user can define, and expandability that may surpass current Network Intrusion Detection Systems (NIDS). IBID provides stateful filtering.

Performance and Stability The core of IBID (Virtual Smart Network Interface Card) is designed as a Linux Loadable Kernel Module (LKM), for performance optimality (avoiding interruption during the packet filtering and minimizing memory transfer between kernel space and user space).

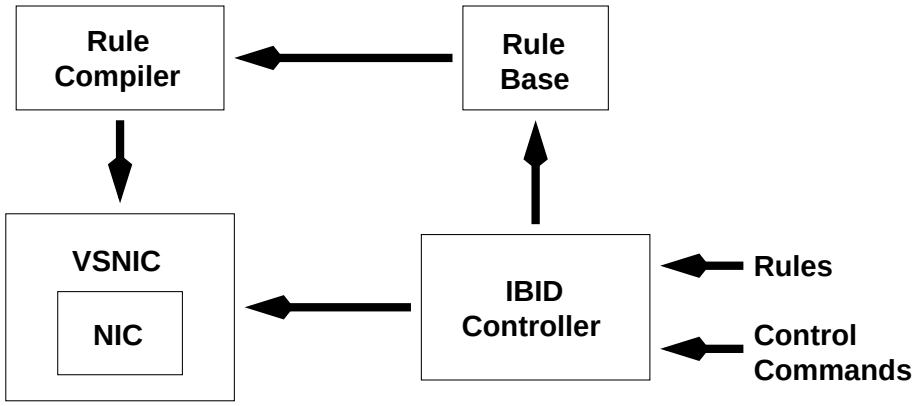

Figure 8: Architecture Overview

Traditional network IDS do not prevent attacks, and firewalls cannot do complex filtering because they are bottlenecks on overall network throughput. Providing local filtering allows each node to protect itself, and also distributes the cost of providing stateful, complex filtering.

3.2 Software Architecture Overview

The IBID system consists of four components: the (Virtual) Smart Network Interface Card (VSNIC), the Rule Compiler, the Rule Base, and the IBID Controller (Figure 8).

The SNIC works as a core component of the IBID System. In the current virtual implementation, it consists of an network interface card device and a loadable kernel module (LKM) comprising a network driver and a virtual machine which actually performs the packet filtering. The IBID Controller is a user interface module. It is executed as a user process, so that a user can define/delete/reset the rules, and start/stop/monitor the system. The Rule Base module receives a set of rules from a user through the IBID Controller, modifies its rule base, and deduces a conclusion from the rules it contains. The Rule Compiler receives a message of the conclusion from the Rule Base, compiles them into a lower-level representation (in the current implementation, BPF instruction codes) and sends the code to the SNIC.

3.3 Software Components

The system has been developed under the Linux operating system with a Pentium CPU and a 3c905B 100 BaseTX network card. We used kernel release 2.4.9, and the 3c59x network driver for the base of the VSNIC.

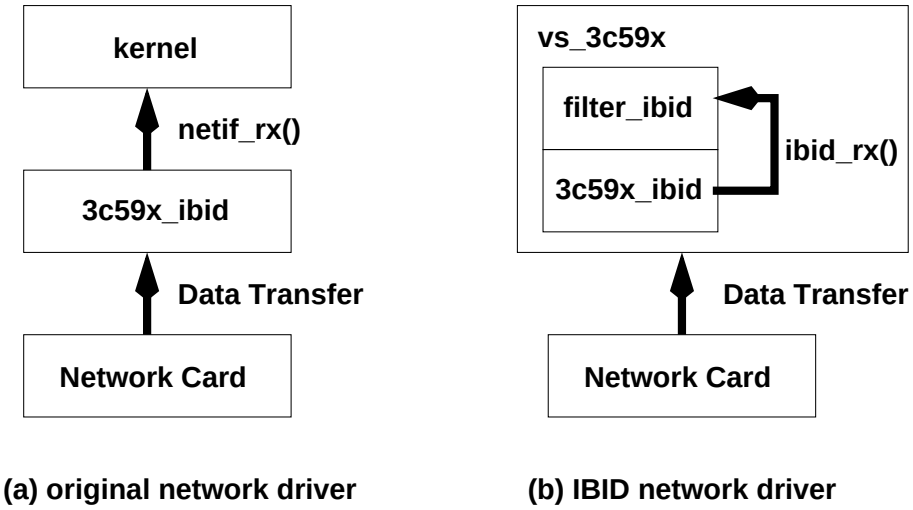

Figure 9: Data paths through the kernel

3.3.1 VSNIC (Virtual Smart Network Interface Card)

Network Interface Module

Our loadable kernel module that implemented the VSNIC is called vs_3c59x. Its primary purpose is to serve as an interface between the network device and the kernel, just like the original device driver. One difference is, however, while 3c59x calls the netif_rx() function in the kernel when it completes the necessary data transfer (Figure 9, part (a)), vs_3c59x first calls ibid_rx() function in itself, and calls netif_rx() after the completion of that filter function (Figure 9, part (b)).

This procedure assures that only acceptable packets are sent to the packet handlers of the kernel. In other words, the kernel won't even know that the packet has reached the network card if it was rejected by the VSNIC. This approach obviously places additional overhead on packet handling, but our focus in the software prototype is proof-of-concept rather than raw performance. Performance can be improved by either exploiting a multiprocessor architecture and giving vs_3c59x priority on the second processor, or building a hardware implementation of the SNIC (this is part of our planned future work, as described in the Conclusions). The filter codes are optimized as well, but this will be discussed in later sections.

The secondary task of vs_3c59x is to serve as an interface between the IBID Controller and the filter functionality. The IBID Controller requests the addition, deletion, reseting, starting, or stopping of filters by throwing an interrupt via the ioctl() system call. The interrupt handler in vs_3c59x catches the interrupt and processes the request according to the parameters given to ioctl().

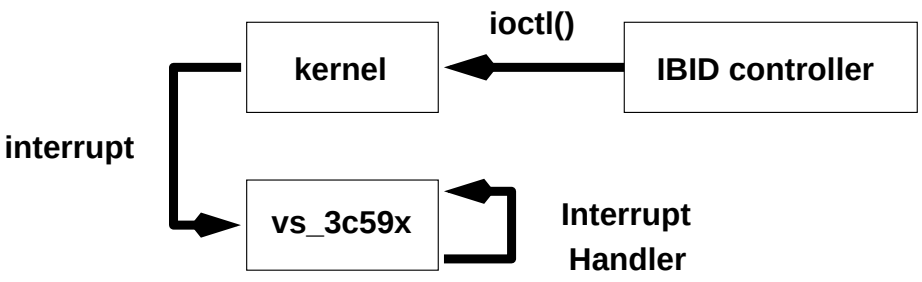

Figure 10: Operating VSNIC via ioctl()

Filter Function

The filter function used in the IBID System is borrowed from Linux Socket Filter (LSF), which is originally borrowed from the Berkeley Packet Filter (BPF). Thus, the design concept of the filter function is just the same as that of BPF:

1. Protocol independency
2. General instruction set
3. Minimized data reference
4. A single C switch statement represents decoding function
5. Using physical registers

A detailed description on BPF can be found in Maccane and Jacobson [SMVJ1992], so here we see briefly how it works.

Virtual Machine Architecture

As Maccane and Jacobson mentioned, BPF adopted a virtual machine architecture from the CMU/Stanford Packet Filter (CSPF) [SMVJ1992]. The advantage of this architecture is that it can reduce data references and comparisons, by explicitly skipping unnecessary evaluations of elements in a packet. For example, in order to verify the condition A AND (B OR C), the virtual machine does not necessarily verify all the predicates. Instead, it applies standard lazy evaluation with short-circuiting operators, as in the C programming language, and aborts evaluation of the logical predicate as soon as the final result is known.

For a bit more realistic example, instruction codes for a filter to accept all IP packets except those from 128.3.112.1 would be represented as follows:

```
ldh     [12]
jeq     #0x0800, Next, Reject
```

```
Next:
    ld     [26]
    jeq    #0x80037001, Reject, Accept
Accept:
    ret    1
Reject:
    ret    0
```

The first comparison checks for the IP protocol, and the second checks for the IP address 128.3.112.1. If the packet is not an IP packet, there is no point in looking for the IP address (in fact, doing so would be semantically nonsensical), so the packet is immediately rejected (by the branch to the `Reject` label). Only if the packet was an IP packet is the address then checked for a match.

The representation using predicate logic can be:

(Ether-Protocol(IP)) AND (NOT Ip-Source(0x80037001))

It should be noticed that the number of evaluations required to verify the representation is smaller in the former representation. While the latter always requires two evaluations, the former skips the second evaluation when Ether-Protocol(IP) is known to be false. The translation between above two representations is a task of the Rule Compiler and Rule Base system, which will be explained in the next section.

Rule Base System

The translation process is divided into two parts. The first half is performed by the Rule Base. The Rule Base translates a user-specified representation of a filter into a protocol-independent notation, based on the rules (or functions) the system already knows. For example, the Rule Base should know a rule such as "If a value of a half word at offset 12 is 0x0800 then Ether-Protocol(IP) is true," i.e., the packet is an IP packet. By applying unification and inference procedures (modus ponens, etc.) to those rules, the system can deduce a conclusion which is to be passed to the Rule Compiler. The Rule Compiler receives the conclusion and translates it into instruction codes for the virtual machine. The conclusion must be in a ground form, which is able to be directly translated into instruction codes. By a ground form, we mean that all the predicates in the form must be terms that the Rule Compiler can understand. Thus far, `(Offset x y z)` is the only term known to the Rule Compiler. By `(Offset x y z)`, it is meant that the value at the offset of `y` bytes, with the length of `x` bits is `z`. For example, a ground form deduced from the above predicate is:

```
(Offset 16 12 0x0800) AND (NOT (Offset 32 26 0x80037001)
```

Implementation

In our current implementation using LISP, rules are introduced to the system by a function called `tell`, and the deduction is returned by a function called `ask` (see Figure 11).

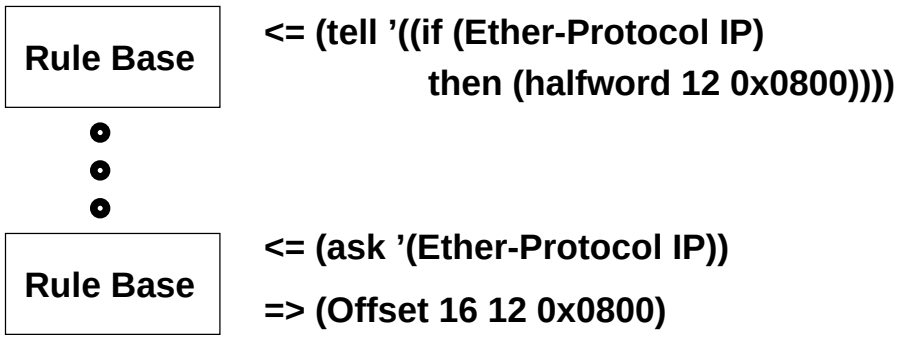

Figure 11: Tell and Ask

The function `ask` performs forward-chaining from the given condition, and deduces a ground form. An example of the performed deduction is:

```
(AND (Ether-Protocol IP)
     (AND (IP-Protocol TCP)
          (OR (IP-Source 0x80037001)
              (IP-Source 0x80037002))))
```

By modus ponens and IP Rules, this becomes:

```
=> (AND (Offset 16 12 0x0800)
        (AND (AND (Offset 16 12 0x0800)
                  (Offset 8 23 0x6))
             (OR (AND (Offset 16 12 0x0800)
                      (Offset 32 26 0x80037001))
                 (AND (Offset 16 12 0x0800)
                      (Offset 32 26 0x80037002)))))
```

then by distribution of AND:

```
=> (AND (Offset 16 12 0x0800)
        (AND (AND (Offset 16 12 0x0800)
                  (Offset 8 23 0x6))
             (AND (Offset 16 12 0x0800)
                  (OR (Offset 32 26 0x80037001)))
                      (Offset 32 26 0x80037002)))))
```

and by distribution of AND again:

```
=> (AND (Offset 16 12 0x0800)
        (AND (Offset 16 12 0x0800)
             (AND (Offset 8 23 0x6))
                  (OR (Offset 32 26 0x80037001)))
                     (Offset 32 26 0x80037002)))))
```

and finally, by elimination of AND:

```
=> (AND (Offset 16 12 0x0800)
        (AND (Offset 8 23 0x6))
             (OR (Offset 32 26 0x80037001)))
                 (Offset 32 26 0x80037002)))))
```

The equivalent representation in ordinal predicate logic would be:

```
  Ip AND Tcp AND (Ips-1 OR Ips-2)
=> Oip AND (Oip AND Otcp) AND ((Oip AND Oips-1) OR (Oip AND Oips-2))
=> Oip AND (Oip AND Otcp) AND (Oip AND (Oips-1 OR Oips-2))
=> Oip AND (Oip AND (Otcp AND (Oips-1 OR Oips-2))
=> Oip AND (Otcp AND (OIps-1 OR Oips-2))

  Ip := Ether protocol is IP
  Tcp := IP protocol is TCP
  Ips-1 := IP source is 128.3.112.1 (0x80037001)
  Ips-2 := IP source is 128.3.112.2 (0x80037002)

  Oip := 16 Bits at offset 12 is0x0800
  Otcp := 8 Bits at offset 23 is 0x6
  Oips-1 := 32 Bits at offset 26 is 0x80037001
  Oips-2 := 32 Bits at offset 26 is 0x80037002
```

With the rules predefined:

```
  (IF (Ether-Protocol IP) THEN (Offset 16 12 0x0800))
  (IF (IP-Protocol TCP) THEN (AND (Ether-Protocol IP)
                                  (Offset 8 23 0x6))
  (IF (IP-Source x) THEN (AND (Ether-Protocol IP)
                              (Offset 32 26 x)
```

Notice, once a predicate is declared in the left-hand side of a rule, it can be used in the right-hand side of other rules, just like a predicate with `Offset`. Variables can be used in rules. Any symbols starting with a small letter are considered as a variable, and will be unified with a filter statement.

The syntax for the Rule Base can be defined as below:

```
  <rule>      := (IF <predicate> THEN <predicate>)
```

```
<predicate> := (<predicate>)
             | (AND <predicate> <predicate>)
             | (OR <predicate> <predicate>)
             | (NOT <predicate>)
             | (symbol {<value>}*)
             | Offset <value> <value> <value>

<value>     := symbol
```

Rule Compiler

The Rule Compiler is implemented using libpcap. This library provides an interface for operating packet filters as well as compiling a given string into a filter. As seen above, the Rule Compiler in the IBID architecture uses only the compiling routines (gencode.c) of libpcap and lets the Rule Base do the remaining tasks. We are currently developing a Rule Compiler from scratch so that we can enrich the original BPF instruction set and perform further optimization.

3.4 Conclusion and Future Enhancements

Starting from the goal as mentioned above, we built the IBID System from modularized software components. In terms of enhancement of the functionality of the system, there are four areas we can examine:

1. Enrichment of the rules: As described in 2.2.1, this can be done via the IBID Control module. Users can define a new rule whenever they encounter a new signature, as well as a new protocol. This knowledge engineering part contributes to actual intrusion detections, and should be done on an ongoing basis, once the system development is complete.

2. Enhancement of the components: This includes improving an user interface of the IBID Control module, improving the Rule Base with additional inference rules or syntax, or optimization of the Rule Compiler. An enhancement of each individual component should be able to be done independently.

3. Hardware implementation of the Smart NIC. We are in the early stages of developing an FPGA-based hardware prototype of a smart NIC. We will first implement a Berkeley Packet Filter (BPF) virtual machine, and transport our rule-based filtering system from the Linux kernel to the SNIC. We are still considering whether to use a processor core (such as the PPC core in the Xilinx Virtex-II Pro) or to implement the BPF interpreter directly in the FPGA (this will enable us to use simpler, cheaper components).

4. Expansion of the underlying architecture of the system: Starting from an enhancement of BPF instruction set, this must involve all the modules: Rule Compiler, Rule Base, and IBID Controller. For example we may want to implement another register into the virtual machine (for example, for logging purpose) and specific operations on it.

The original BPF is stateless, and is an accumulator-based architecture. This keeps the code simple, but prevents us from performing certaintypes of filtering, such as that based on overlapping fragments. After we have the initial hardware implementation tested and debugged, we will devise extensions to the core BPF. We know that we will extend the BPF core to include state, and will examine common intrusion signatures to determine what additional features should be added.

We can enhance the Rule Base by adding a function evaluation feature as an example for enhancement of the components. If users can define functions to be used in rules, rules become more concise, flexible and intuitive. Using the example of 2.2.3, we can compare rules defined without a function, and rules defined with a function:

Ether-Protocol rules without a function:

```
(IF (Ether-Protocol IP) THEN (Offset 16 12 0x0800))
(IF (Ether-Protocol ARP) THEN (Offset 16 12 0x0805))
(IF (Ether-Protocol RARP) THEN (Offset 16 12 0x0806))
```

Ether-Protocol rules with a function:

```
(IF (Ether-Protocol x) THEN (Offset 16 12 (ether x)))
```

The Rule Base knows a rule as, "If Ether-Protocol is IP, then a value of a half word at offset 12 must be ether(IP)." And it knows a function `ether` such that `ether` returns the Ethernet protocol identifier for the given protocol. When applied to IP, `ether` returns 0x0800. The proposed syntax for this Rule Base is in Figure 12.

Notice every predicate now ultimately defines a comparison of values, and `Offset` becomes a two-argument function. The function definition and evaluation may be done by the implemention language environment (ex. LISP interpreter), or integrated into the Rule Base using the implemention language.

The current IBID system, with its defined rules, has not yet been integrated into any intrusion detection system. However, it has considerable promise for such application in today's network intrusion detection systems—benefits such as locality, flexibility, modularity, and domain-independence of the rule base system.

4 Computational Resiliency

The Computational Resiliency project developed mechanisms to allow applications to respond to attacks and faults, thereby restoring application readiness and ensuring continued operation. These mechanisms include support for replication, migration, camouflage, and mutation. The deliverables on this contract include software prototypes, formal models, and the information warfare-hardening of two applications of interest to DARPA/DoD.

```
<message>    := <rule-def> | <func-def>
<rule-def>   := (IF <predicate> THEN <predicate>))
<func-def>   := (symbol {(IF {<value>}* THEN <value>)}*)

<predicate>  := (<predicate>)
                | <predicate> AND <predicate>
                | <predicate> OR <predicate>
                | NOT <predicate>
                | (<value> = <value>)

<value>      := (<function> {<value>}*)
                | symbol

<function>   := Symbol

Built-in functions : offset, +, -, *, /
```

Figure 12: Syntax for Rule Base with functions

During this project, we developed a core library supporting Computational Resiliency. This library includes automated support for replication, migration, agreement, and functionality mutation in distributed applications scientific.

As originally specified, the goals and milestones of the project were as follows:

- Formal models
  - Core calculus
  - Resource policy mechanism
  - Equivalence notions
  - Protocol analysis
  - Extensions and analysis
- Software Development
  - Extend SCPlib Migration
  - Simple camouflage and decoys
  - Functionality mutation
  - Advanced camouflage
  - Policy frameworks
  - Computational Resiliency-aware schedulers
- Integration of two applications of interest to the DoD with the Computational Resiliency library

The first year of the project was spent solidifying the base library, adding thread replication, and extending migration. The actual achievements deviated somewhat from the initial goals, as it was necessary to ensure that the base library had a sufficiently reliable and secure base that the higher-level services would have a solid foundation.

The goals achieved for the library included:

- Passive replication with checkpointing for CRlib applications
- Active replication at a user-settable level
- Split/merge for replicated thread groups
- Message authentication using DSS
- Fuzzy agreement protocols
- Synchronous and asynchronous liveness checking

Lessons Learned

The primary lesson taken away is our dependence on the difficulty in increasing the resiliency of an application in isolation from the underlying system, and in the face of a determined, patient adversary. Our approach works well for attacks and failures that occur in the context of the application, but we were faced with a dilemma:

- assume that the base system provided stability and resistance to external attacks, and concentrate on the goals as put forth in the original proposal, thereby ignoring large classes of attacks, and leaving ourselves susceptible to them, or
- fortify the base system before proceeding with the development of the scheduling and camouflage systems.

In particular, we felt it necessary to deal with issues involving agreement, splitting and merging functionality of all threads in a group (necessary for future work in camouflage), and additional replication schemes (necessary to support scheduling alternatives in future work) before proceeding.

In its current state, our Computational Resiliency library will deal with classes of attacks and failures that:

- disable a subset of the threads in a group (either by killing processes or crashing the machines where they run), in a fail-stop sense;
- compromise a subset (up to 1/3 the total threads; 1/2 if the optional DSS authentication is used) leading to Byzantine failures;
- attempt to use man-in-the-middle techniques or spoofed messages to disrupt computation.

In either of the first two cases, the number of replicated threads will be restored to the prior level at the next liveness check. The use of signed messages for authentication thwarts the third class, and increases resiliency against Byzantine failures.

The CRlib will not help in the face of an adversary who is patient, stealthy, and puissant—such an adversary that might

1. break into multiple systems (potentially all of them where our jobs are running)

2. map the system and observe which processes are running CRlib jobs

3. coordinate an attack sufficient to compromise more than 1/2 to cause Byzantine failure, or to kill all threads in a group.

It is because of this type of attacker that we specified a "safe zone" in the original proposal. If we can rely on such a zone, then we can limit our exposure to these attacks by ensuring that less than 1/2 of our threads run outside the zone. It is unclear to us whether this is a reasonable stance to take.

The continued development of camouflage techniques may provide additional protection against these attacks. Even with excellent camouflage, there will be valid reasons to run "in the open" under a low threat condition. In that case, it is critical that intrusion detection systems recognize attacks and reconnaissance activity extremely early, so that camouflage can be enabled before the attackers have compromised many systems, giving them the ability to observe system transitions from uncamouflaged to camouflaged execution, thereby rendering the camouflage ineffective.

The following journal papers summarize the results of the project:

1. J. Lee, S. J. Chapin, and S. Taylor, "Computational Resiliency", to appear in a special issue of the journal Quality and Reliability Engineering International on Computer Network Security, 2003.

2. J. Lee, S. J. Chapin, and S. Taylor, "Reliable Heterogeneous Applications", to appear in IEEE Transactions on Reliability, special issue on Quality/Reliability Engineering of Information Systems, 2003.

3. Norka Lucena, Steve J. Chapin, and Joohan Lee. "Assuring Consistency and Improving Reliability in Group Communication Mechanisms in Computational Resiliency," IEEE Workshop on Information Assurance and Security, West Point, New York, 2003.

5 Conclusion

This report has summarized the results for three projects sponsored by the Systems Assurance contract. The prototype software has been delivered on CD-ROM.

References

[1] Ross Anderson, editor. *Information Hiding: Proceedings of the First International Workshop*, Cambridge, U.K., May 30-June 01, 1996. Springer.

[2] Ross J. Anderson and Fabien A.P. Petitcolas. On the limits of steganography. *IEEE Journal of Selected Areas in Communications*, 16(4):474–481, May 1998.

[3] David Aucsmit, editor. *Information Hiding: Proceedings of the Second International Workshop*, Portland, Oregon, U.S.A., April 14-17, 1998. Springer.

[4] Daniel J. Barrett and Richard Silverman. *SSH, The Secure Shell: The Definitive Guide*. O'Reilly, 2001.

[5] Christian Cachin. An information-theoretic model for steganography. In David Aucsmith, editor, *Information Hiding: Proceedings of the Second International Workshop*, pages 306–318, Portland, Oregon, U.S.A., April 14-17, 1998. Springer.

[6] Steve J. Chapin and Shawn Ostermann. Information hiding through semantics-preserving application-layer protocol steganography. Technical report, Center for Systems Assurance, Syracuse University, October 2002. [Internal Document].

[7] Tom Dunigan. Internet steganography. Technical report, Oak Ridge National Laboratory (Contract No. DE-AC05-96OR22464), Oak Ridge, Tennessee, October 1998. [ORNL/TM-limited distribution].

[8] J. Mark Ettinger. Steganalysis and game equilibria. In David Aucsmith, editor, *Information Hiding: Proceedings of the Second International Workshop*, pages 319–328, Portland, Oregon, U.S.A., April 14-17, 1998. Springer.

[9] Nick Feamster, Magdalena Balazinska, Greg Harfst, Hari Balakrishnan, and David Karger. Infranet: Circumventing web censorship and surveillance. In *Proceedings of the 11th USENIX Security Symposium*, pages 247–262, San Francisco, California, U.S.A., August 05-19, 2002. The USENIX Association.

[10] Gina Fisk, Mike Fisk, Christos Papadopoulos, and Joshua Neil. Eliminating steganography in Internet traffi c with active wardens. In Job Oostveen, editor, *Information Hiding: Preproceedings of the Fifth International Workshop*, pages 29–46, Noordwijkerhout, The Netherlands, October 7-9, 2002. Springer.

[11] Theodore Handel and Maxwell Sandford. Hiding data in the OSI network model. In Ross Anderson, editor, *Information Hiding: Proceedings of the First International Workshop*, pages 23–38, Cambridge, U.K., May 30-June 01, 1996. Springer.

[12] Ka0ticSH. Diggin em walls (part 3) - advanced/other techniques for bypassing fi rewalls. New Order, April 11, 2002. Retrieved on August 28, 2002 from the World Wide Web: http://neworder.box.sk/newsread.php?newsid=3957.

[13] Stefan Katzenbeisser and Fabien A.P. Petitcolas. *Information Hiding: Techniques for Steganography and Digital Watermarking*. Artech House, Norwood, MA, 2000.

[14] Stefan Katzenbeisser and Fabien A.P. Petitcolas. Defi ning security in steganographic systems. In *Electronic Imaging, Photonics West, SPIE*, volume 4675 of *Security and Watermarking of Multimedia Contents IV*, pages 50–56, 2002.

[15] Thomas Mittelholzer. An information-theoretic approach to steganography and watermarking. In Andreas Pfi tzmann, editor, *Information Hiding: Proceedings of the Third International Workshop*, pages 1–16, Dresden, Germany, September 29-October 01, 1999. Springer.

[16] Ira S. Moskowitz, editor. *Information Hiding: Proceedings of the Fourth International Workshop*, Pittsburg, PA, U.S.A., April 25-27, 2001. Springer.

[17] Ira S. Moskowitz, Garth E. Longdon, and LiWuChang. A new paradigm hidden in steganography. In *Proceedings of the New Security Paradigm Workshop 2000*, pages 41–50, Cork, Ireland, September 19-21, 2000. n.

[18] Job Oostveen, editor. *Information Hiding: Preproceedings of the Fifth International Workshop*, Noordwijkerhout, The Netherlands, October 7-9, 2002.

[19] Andreas Pfi tzmann, editor. *Information Hiding: Proceedings of the Third International Workshop*, Dresden, Germany, September 29-October 01, 1999. Springer.

[20] Brigit Pfi tzmann. Information hiding terminology. In Ross Anderson, editor, *Information Hiding: Proceedings of the First International Workshop*, pages 347–349, Cambridge, U.K., May 30-June 01, 1996. Springer.

[21] Niels Provos. Defending against statistical steganalysis. In *Proceedings of the 10th USENIX Security Symposium*, pages 323–335, Washington, DC, U.S.A., August 13-17, 2001. The USENIX Association.

[22] route@infonexus.com and alhambra@infornexus.com. Article 6. Phrack Magazine, 49, August 1996. Retrieved on August 27, 2002 from the World Wide Web: http://www.phrack.com/phrack/49/P49-06.

[23] Craig H. Rowland. Covert channels in the TCP/IP protocol suite. Psionics Technologies, November 14, 1996. Retrieved on August 23, 2002 from the World Wide Web: http://www.psionic.com/papers/whitep03.html.

[24] Gustavus J. Simmons. The prisoners' problem and the subliminal channel. In *Proceedings of CRYPTO '83*, pages 51–67. Plenum Press, 1984.

[25] Bill Watterson. *Something Under the Bed is Drooling*. Andrews and McMeel, pp. 101–104, Kansas City, MO, 1988.

[26] Peter Wayner. Mimic functions. *Cryptologia*, XVI(3):193–214, July 1992.

[27] Peter Wayner. *Disappearing Cryptography - Information Hiding: Steganography and Watermarking*. Morgan Kaufmann Publishers, San Francisco, CA, 2nd edition, 2002.

[28] T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen. SSH authentication protocol. Working Group Internet-Draft, September 20, 2002. Expires: March 21, 2002. Retrieved on October 26, 2002 from the World Wide Web: http://www.ietf.org/internet-drafts/draft-ietf-secsh-userauth-16.txt.

[29] T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen. SSH connection protocol. Working Group Internet-Draft, September 20, 2002. Expires: March 21, 2002. Retrieved on October 26, 2002 from the World Wide Web: http://www.ietf.org/internet-drafts/draft-ietf-secsh-connect-16.txt.

[30] T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen. SSH protocol architecture. Working Group Internet-Draft, September 20, 2002. Expires: March 21, 2002. Retrieved on October 26, 2002 from the World Wide Web: http://www.ietf.org/internet-drafts/draft-ietf-secsh-architecture-13.txt.

[31] T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen. SSH transport layer protocol. Working Group Internet-Draft, September 20, 2002. Expires: March 21, 2002. Retrieved on October 26, 2002 from the World Wide Web: http://www.ietf.org/internet-drafts/draft-ietf-secsh-transport-15.txt.

[32] J. Zöllner, H. Federrath, H. Klimant, A. Pfi tzmann, R. Piotraschke, A. Westfeld, G. Wicke, and G. Wolf. Modeling the security of steganographic systems. In David Aucsmith, editor, *Information Hiding: Proceedings of the Second International Workshop*, pages 344–354, Portland, Oregon, U.S.A., April 14-17, 1998. Springer.