{norka,jmpease, pyadolla, chapin}@ecs.syr.edu

Syntax and Semantics-Preserving Application-Layer Protocol Steganography[*]

Norka B. Lucena, James Pease, Payman Yadollahpour, Steve J. Chapin

Systems Assurance Institute
Syracuse University
111 College Place 3-114, Syracuse, NY 13244

Abstract. Protocol steganography allows users who wish to communicate secretly to embed information within other messages and network control protocols used by common applications. This form of unobservable communication can be used as means to enhance privacy and anonymity as well as for many other purposes, ranging from entertainment to protected business communication or national defense. In this paper, we describe our approach to application-layer protocol steganography, describing how we can embed messages into a commonly used TCP/IP protocol. We also introduce the notions of syntax and semantics preservation, which ensure that messages after embedding still conform to the host protocol. Based on those concepts, we attempt to produce reasonably secure and robust stegosystems. To demonstrate the efficacy of our approach, we have implemented protocol steganography within the Secure Shell (SSH) protocol. Findings indicate that protocol steganographic system is reasonably secure if the statistical profile of the covermessages and the statistical profile of its traffic match their counterparts after embedding.

Keywords: steganography, application protocols, syntax, semantics, SSH

1 Introduction

Steganography, from the Greek "covered writing," refers to the practice of hiding information within other information [1]. Its purpose is to allow two parties to communicate in such a way that the presence of the message cannot be detected. While cryptography focuses on protecting the content of the message, steganography conceals the mere existence of the message. Classical steganography comprises a broad variety of methods and materials, ranging from tattooing messengers' heads to using invisible ink and microdots. Modern steganography involves digital media and techniques: images, formatted and written text, digital sounds, and video, as well as some others less orthodox such as storage devices and TCP/IP packets [2]. In recent years,the evolution of stegosystems has received particular attention, as have the security and robustness of their

[*] This work was supported in part by the State of New York, the CASE Center's SUPRIA program at Syracuse University, and the Air Force Research Laboratory (AFRL).

methods [3–7]. In this context, protocol steganography arises as a new means of hiding information in Internet messages to achieve secret communication.

Protocol steganography is the art of embedding information within messages and network control protocols used by common applications [8]. Protocol steganography takes advantage of existing application-layer network traffic to communicate privately, which could be a useful and important means of communication in many different areas. It can be effective in law enforcement for undercover investigations and espionage. For example, it could have been convenient for "Enron whistleblower" Sheron Watkins to have set up a private communication channel with the District Attorney's office that worked without having to deploy special anonymity frameworks, but utilizing the traffic generated by one of her regular web-browsing sessions. The business arena can also benefit from hiding the communication when doing important negotiations.

Early attempts at hiding information within network protocols were based on the discovery of covert channels—communication channels neither designed nor intended to transfer information at all [9]—in TCP/IP packets [10–13]). In contrast, our approach of protocol steganography specifically targets application-layer protocols such SMTP (for email service), FTP (for file transfer), SSH (for secure login), LDAP (for distributed directory services), and HTTP (for web browsing, which alone accounted for over 53% of all Internet traffic in 2002 [14]). We aim to hide information within the format and structure of the protocol, and not in the transmitted content, such as images, sounds, text, or video. Information hiding within these content types can be achieve using well-known steganographic techniques before the content is sent across the network.

The most relevant feature of a steganographic system is how secure it is. At the moment, there is controversy in the field regarding the definition of a perfectly secure system [15, 16]. The most cited approaches are based on information theory and the ideas of security taken from cryptography definitions [17–20]. There are other definitions such as the Ettinger's game-theoretical definition [21] and the complexity-theoretical definitions in [22, 23]. However, to the best of our knowledge, there is no record of any implemented stegosystem proven secure under those definitions. We recognize the enormous effort put behind producing an exact mathematical definition of security, but for this paper we limited our approach to produce an empirically and "reasonably secure" [24] stegosystem.

The remainder of this paper is organized as follows. Section 2 explains the concepts of security and robustness in terms of protocol steganography. Section 3 describes the model for secret communication considered in our approach and discusses its potential advantages. Section 4 presents a summary of the research to date and related work in relevant areas of steganography. Section 5 explores the concept of protocol steganography through the SSH protocol, describes a prototype implementation, and discusses consequences and important issues regarding security and robustness of the approach as well. Finally, Section 6 lists some conclusions and remarks of lessons learned.

2 Security and Robustness in Protocol Steganography

Steganographic systems are usually defined in terms of three elements: capacity, security, and robustness. *Capacity* is the amount of information that can be hidden in the cover. *Security* refers to the difficulty that a knowledgeable adversary (one who understands the stegosystem) has in obtaining evidence or even grounds for suspicion that a secret communication is taking place. *Robustness* is the amount of alteration a stegomessage can support without the hidden message being destroyed [1, 25]. For this study, we focus in examining both security and robustness of our steganographic methods against the threat of passive and active adversaries more than in increasing their capacity.

The protocol steganography model assumes prior knowledge of the distribution of the covers, standard practice when defining stegosystems. This allows to produce appropriate embedding and extraction methods which minimize or eliminate alterations in the statistical profile of the covermessages. Protocol steganography however needs to deal not only with the characteristics of the covermessage itself but also with the statistical profile of its traffic such as the distribution of the payload length. A *reasonably secure* protocol stegosystem is one in which the adversary cannot distinguish between a covermessage and a stegomessage by analyzing the meaning of the packet payload and the statistical properties of the protocol traffic. Stegomessages are *reasonably robust* if, after alterations from a malicious attacker, they are rendered inadequate regarding their protocol semantics. Stegomessages that are not semantically valid usually cause the interruption of the overt communication.

Seeking to produce both secure and robust stegosystems, we define two concepts for stegomessages: syntax preservation and semantics preservation. *Syntax preservation* guarantees that the stegomessage is well formed within the rules of the protocol; the actual meaning of the stegomessage may be different than the original cover. *Semantics preservation* means that, as observed at a point along the message's path through the network, the stegomessage has the same meaning as the original cover. Semantics preservation is stronger than, and implies, syntax preservation. Semantics preservation increases robustness—it reduces or eliminates the possibility for an active attacker to render the hidden message useless without causing substantial damage to the packet, thereby breaking the overt communication. The early work in covert channels was, in general, neither syntax nor semantics preserving, and depended on routers not performing tight checking against the protocol specification.

3 Framework for Secret Communication

Our model for protocol steganography involves two agents who wish to communicate secretly through arbitrary Internet traffic in a hostile environment (see Figure 1). *Alice* and *Bob* [26] are two agents who wish to communicate secretly. To achieve that, they use a communication path already in place between themselves or two arbitrary communicating processes, the *sender* and *receiver*. Adversaries located between Alice and Bob can be both active or passive. A passive adversary, *Eve*, observes the communication to discover stegomessages. Eve's eventual goal is to find the embedded information, and prove its existence to a third party, if necessary. An active adversary, *Mallory* [27] attempts to remove the embedded message during the communication process, while preserving the integrity of the cover.

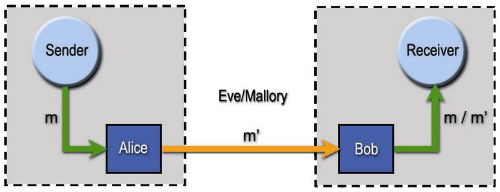


Fig. 1. Framework for Secret Communication.

Two scenarios are possible depending on whether or not Alice and Bob are the same as the sender and the receiver, respectively. In the first case, Alice and Bob 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, Alice and Bob are placed somewhere along an arbitrary communication path, modifying messages 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 covermessage. The midpoints where Alice and Bob 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 1.

Considering all combinations of internal agents and external confederates and all different points where the message can be altered yields six different combinations of roles for the agents, as shown in Figure 2. In this discussion, following the established information hiding terminology [28], Alice executes the *embedding* process and Bob the *extraction* process, represented in the picture as a circle and a diamond, respectively. As pointed out by Pfitzmann [28], the embedding and extracting processes may require the use of a *stegokey*, not shown in the picture. The *cover* (i.e. the original harmless message) is m, and the *stegomessage* (i.e. the message with steganographic content) is m'.

The six possible sets of agent roles are as follows:

1. *Alice* acts as sender and *Bob* as receiver—the message along the entire path is m'.

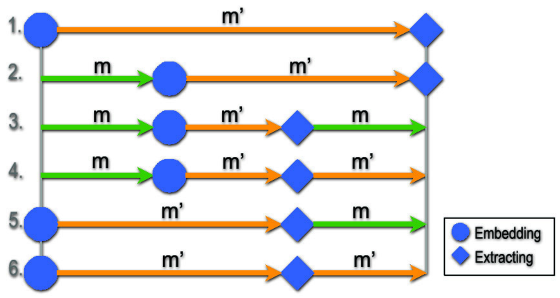


Fig. 2. Message Paths.

2. *Alice* is a middleman, embedding information to the message on its way, and *Bob* acts as *receiver*—the message from the *sender* to *Alice*'s location is m, while from there to the endpoint is m'.
3. Both agents are middlemen, and *Bob* restores the message to its original form—the message from the *sender*'s point to where *Alice*'s location is m, from *Alice*'s to *Bob*'s is m', and from there to the endpoint is m again, because extraction of the hidden content and restoration of the original cover message occurred at *Bob*'s location.
4. Both agents are middlemen, but *Bob* does not restore the message—the message from the *sender*'s point to the *Alice*'s location is m, and from *Alice*'s to the *receiver*'s point is m', with the hidden information extracted at *Bob*'s location while the message was in transit.
5. *Alice* is acting as *sender*, with *Bob* as a middleman extracting the embedded information and restoring the original message—the message from the initial point to *Bob*'s location is m', and from *Bob*'s location to the *receiver*'s point is m.
6. *Alice* is acting as sender and *Bob* is a middleman extracting the hidden information without restoring the message as it travels to the *receiver*—the message from end to end is m', but B gets the hidden content somewhere before the message reaches its destination.

Even though not every one of these scenarios might be realistic, cases 1 and 3 certainly are. Thus, they were the focus of this study. All the options where the hidden content is extracted but the message is not restored seem risky. In particular, case 4 wherein the message seen by the receiver is clearly different from that seen by the sender, neither of whom are the agents communicating secretly.

3.1 Issues with Middlemen

Having the agents acting as middlemen in the communication stream provides several advantages, because any packet that will flow past the locations where Alice and Bob are can be modified (as long as an embedding function that preserves both syntax and semantics is available for the transport or application protocol in that packet). That intermediate location lowers the susceptibility to traffic analysis, as there is no longer a single source/sink for the stegomessages, and there is no specific protocol used. It also allows us to achieve a higher bit rate as well as privacy, anonymity, and plausible deniability, in some cases. In the case of undercover operations, for example, an ideal situation would be that Alice is located on the last router inside the sender's domain (the egress router for that domain), and Bob is located on the first router outside the domain (the ingress router). In such scheme, m' will be "on the wire" for the minimum possible time, lowering the probability of detection.

Detection of packet modifications along the communication path might seem trivial for an observer monitoring the network. We argue that it is not. First of all, the modified packets at the embedding and the extraction points will be both syntax and semantics preserving, which evades routing and intrusion detection defense mechanisms. Secondly, individual packet comparison from both sides of an embedding/extraction point is resource intensive and not currently done by IDS systems to avoid the overhead incurred with large amounts of traffic. Lastly, routine network operations for most IPSs, for example, involve the collection of aggregate traffic statistics rather than individual packet analysis, because of the high volume [29].

IP fragmentation is another issue that can affect the reliability of the communication when the agents are middlemen. When the application-layer protocol uses TCP as transport protocol, we assume that the packets used as carrier are delivered reliably. If there still exist packet loses, they are treated as communication errors. Fragmentation rates in packets of TCP applications are minimal. In addition, most of them set the "don't fragment" bit on. In contrast, when the application-layer protocol uses UDP, additional mechanisms need to be implemented to guarantee that Bob actually receives the message sent by Alice.

4 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, 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 messages and network control protocols used by the applications, not inside images transmitted as attachments by an email application, for example.

Examples of implementation of covert channels in the TCP/IP protocol suite are presented by Rowland [13], Project Loki [12], Ka0ticSH [30], and more deeply and extensively by Dunigan [10]. These researchers focused their attention in the network and transport layers of the OSI network model. In spite of that, Dunigan [10] did point out in his discussion of network steganography that application-layer protocols, such as Telnet, FTP, SMTP, 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. Rowland [13] 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 [10] analyzed the embedding of information, not only in those fields, but in some other fields of both the IP and the UDP 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. Project Loki [12, 30] explored the concept of ICMP tunneling, exploiting covert channels inside of ICMP_ECHO traffic. All these approaches, without minimizing their importance, can be detected or defeated with the latest router and firewall technology.

One such mechanism is reported in Fisk et al. [31]. 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 covermessages in used as structured carrier, and the feasibility of similar methods of steganalysis that target application-layer protocols.

Recently, researches are focusing more of their attention in the use of covert channels using specifically the HTTP protocol. Bowyer [32] described a theoretical example without implementation, wherein a remote access Trojan horse communicates secretly with its control using an HTTP GET request. Although this approach takes advantage of the semantics of regular HTTP messages, as we intent to do, it is different from our approach because it can be blocked by restricting access to certain websites, or by scanning images for steganographic content. Bauer [33] proposed the use of cover channels in HTTP to enlarge anonymity sets and provide unobservability in mix networks. He shares our view of using traffic generated by other subjects to hide communication.

5 A Case Study: SSH

The SSH protocol provides secure remote login and other secure network services over an insecure network [34]. It does so through mechanisms that supply server authentication, confidentiality, and integrity with perfect forward secrecy. There are several implementations of SSH, both commercial and open-source. The latest and most widely used version of the protocol is SSH2.

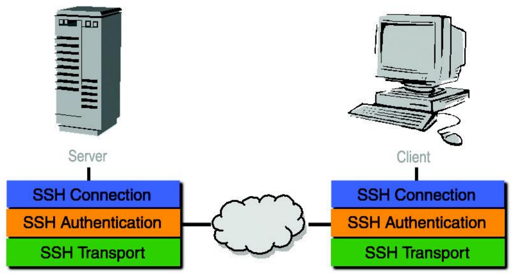


Fig. 3. SSH2 Protocol Architecture.

The SSH2 protocol consists of three major components shown in Figure 3:

- **Transport Layer Protocol**. 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**. Authenticates the client-side user to the server. It runs over the transport layer protocol.
- **Connection Protocol**. 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.

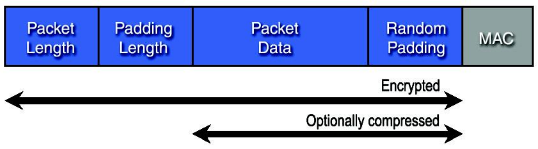


Fig. 4. SSH2 Binary Packet Protocol.

In particular, the Transport Layer protocol defines the *Binary Packet Protocol*, which establishes the format SSH packets follow (see Figure 4). It consists of five fields. *Packet length* is an unsigned 32-bit integer representing the length of the packet data in octets. *Padding length* is the number of octets representing the length of the padding. *Packet data* is the actual content of the message. *Random padding* is an arbitrary-length padding appended to the packet data, so the payload reaches the block cipher sizes specified by the protocol. *MAC* corresponds to the message authentication code, which is computed if previously negotiated. The packet length, padding length, packet data, and random padding fields are encrypted. The packet data and the random padding are compressed before encryption, if compression was specified during the connection setup.

SSH was selected as our first Protocol Steganography case of study for several reasons, with the randomness of the content of its packets being the most important. Encrypted traffic provides an appropriate cover for other messages with uniform distribution, e.g. additional encrypted data. We can blend hidden content securely within what is considered “normal” traffic, without altering the statistical properties of the payload. In addition to that, the fact that the SSH traffic is encrypted may deter adversaries from trying to analyze its content, as pointed out by Barrett and Silverman [35]. Lastly, SSH is widely used and use TCP as transport protocol, which guarantees delivery packets even when they are fragmented.

5.1 Prototype Implementation

We identified several potential possibilities of information hiding in the SSH protocol structure, but selected only two of them for implementation: generating a MAC-like message and adding additional encrypted content to the packet. Such methods of hiding information match, respectively, cases 1 and 3 of our framework of secret communication, described in Figure 1. Case 1 assumes that *Alice* is the sender and *Bob* is the receiver. In Case 3, both agents *Alice* and *Bob* are middlemen located along the communication path. Then, *Bob* needs to restore the stegomessage to the covermessage after extracting the hidden message embedded by *Alice*.

Both implementations were coded in C, tested under Red Hat 8.0, and each of them runs independently of the other. For implementing the first scenario of secret communication, generating a MAC-like message, we modified version 3.5 of Open SSH (http://www.openssh.org), a popular open-source SSH product. For the second scenario, adding encrypted content, we developed a kernel module to capture packets in transit and we tested the system using unmodified OpenSSH 3.5.

Generating a MAC-like Message. In this steganography scenario *Alice* (the sender) and *Bob* (the receiver) are running identically-modified software. At first sight, it might seem strange to pursue secret communication over an already encrypted channel. However, this example of protocol steganography is appropriate for environments where unobtrusive communications are required in the presence of traffic analysis, particularly the number and frequency of messages. In the military and intelligence agencies, even if the content of the communication is encrypted, a significant increase in communications between military units could signal an impending attack [1]. For example, *Alice* might be working at the Pentagon and *Bob* might be a high-level commander in the Middle East. To avoid eavesdropping by terrorists, they encrypt their messages using OpenSSH. It is not possible for the adversary to decipher the messages being sent, but the adversary can perform traffic analysis by studying the length and frequency of the messages exchanged. A sudden increase in traffic gives a clear indication that something “big” is going on.

As shown in Figure 4, the SSH2 specification defines a message authentication code field. The MAC is computed with a 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 allows us to transmit up to 20 additional octets of per packet.

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 and extracts exactly the same amount of octets reserved for computing the MAC in the selected algorithm. At the receiving end, the modified version of the SSH server ignores recomputing the MAC and comparing it with the one received from the client, because the server is action as *Bob*. Instead, *Bob* saves the MAC-like message into a file.

The drawback of this implementation is the impossibility of verifying whether the actual payload of the message was correctly transmitted or not, as a consequence of replacing 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 might be a way of getting around this issue. Because the different MAC algorithms offered by SSH produce MACs of different lengths, it would still be possible to select an algorithm with a short MAC and pad the stegomessage to it. For example, if the hmac-md5-96 algorithm, which computes a 12-octet MAC 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, *Alice* and *Bob* must agree in advance on what algorithm to use. That is trivial to set up through the SSH authentication mechanism. Moreover, when they are not planning to communicate secretly, *Alice* and *Bob* can choose to use the hmac-sha1 algorithm which computes a MAC of length 20, so the average total lengths of their SSH packets does not raise suspicion.

Because we are maintaining the randomness of the covermessage when creating a stegomessage as well as the distribution of the payload length, we consider this stegomethod to be reasonably secure. *Eve* cannot distinguish between two encrypted payloads (cover and stego) of the same size. Because of particular properties of the SSH protocol, embedding a MAC-like message is reasonably robust. SSH takes any change in the MAC at the receiving end as a signal of existence of an attacker somewhere in the middle of the communication stream. SSH issues a warning and the session will be interrupted (normal behavior of the protocol). *Mallory* cannot then recompute and substitute the MAC (besides that involves having knowledge of the encrypted packet payload, the keys, and the algorithms used). *Mallory* cannot make subtle changes to the packet either, such as switching some bits. Our implementation takes similar actions to the ones SSH takes when there when the hidden message is not meaningful to *Bob*.

Adding Additional Encrypted Content to the Packet. This prototype implementation works in the secret communication environments described in cases 2, 3, or 4. However, we will consider only case 3 in this discussion because it is the most challenging. Both *Alice* and *Bob* are middlemen located somewhere along the communication path. *Alice* intercepts a packet from the sender, embeds a portion of her secret message on it, and sends it on. *Bob* extracts the hidden content and restores the message as it originally was before it reaches its destination. *Alice* and *Bob* can be any two parties who wish to communicate secretly by taking advantage of available SSH traffic on the Internet.

This implementation intercepts the SSH traffic and inserts an additional encrypted message at the beginning of the already encrypted payload, as detailed in Figure 5. A 32-bit "magic" number marks the presence of a hidden message.

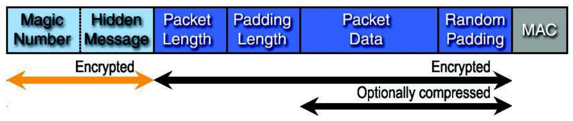


Fig. 5. Adding an encrypted portion with a hidden message to a regular SSH packet at the beginning of the encrypted payload.

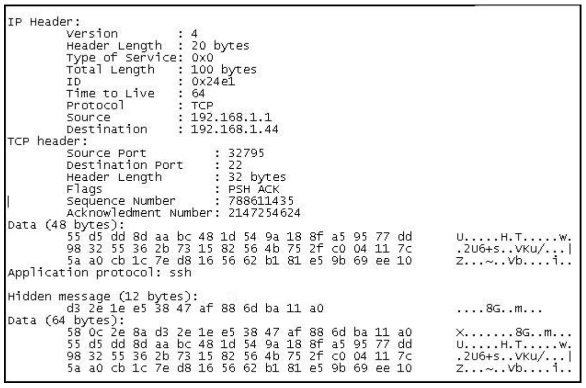
To be able to intercept SSH traffic, we implemented a *Packet Transmogrifier*[1] (PT), written in C for Linux 2.4 kernels. The PT is a piece of software that captures arbitrary packets in transit, embeds secret messages into a stream of outgoing packets, and correspondingly extracts the hidden message when used downstream. It was implemented as a kernel module for deployment in Linux-based routers. In principle, the PT uses a combination of several individual protocol-specific packet transformers as plug-in modules (each of which could be used by an individual application to embed a message in a data stream). This approach gives us the flexibility of embedding hidden messages in packets of multiples types corresponding to different protocols, and with a variety of sources and destinations. The current implementation of the PT provides a series of default protocol-dependent embedder and extractor functions that are called based on the options selected by the user and the payload type of a particular IP packet. The corresponding functions for handling SSH packets are called `sshEmbedder` and `sshExtractor`.

When establishing an SSH session, the client and the server negotiate the algorithms to be used in the session, the MAC algorithm among them. Next, they

[1] With appropriate apologies and thanks to Bill Watterson, creator of "Calvin and Hobbes" [36].

initiate the key exchange. The number of messages exchanged till this point by the client and server are unencrypted, therefore, `sshEmbedder` and `sshExtractor` are not interested in modifying such packets. The analyze their content and discharge them if they are any of the plain-text packets. Once the key exchange is done, both sides, client and server, turn on encryption, perform authentication, and the secure connection is establish. From that particular stage, `sshEmbedder` begins altering the SSH packets, embedding encrypted hidden messages. Conversely, `sshExtractor` attempts to extract a secret message from every encrypted packet and reformats the SSH packet to its original form. The functions `sshEmbedder` and `sshExtractor` are semantics preserving. The SSH traffic reminds encrypted after embedding or extraction, hence both the cover and the stegomessage have the same semantic meaning to a third party observer.

From monitoring SSH traffic, we learned that the most common packet sizes in Telnet-like SSH session are 48 and 80 bytes, with each comprising approximately 23% of the recorded data. For testing the functionality of this implementation, we elected to embed data in chunks of at least 12 octets with a 32-bit (4-octet) CRC to verify the integrity of the message when transmitted. That is, the total length of the SSH payload is incremented by at least 16 octets after the embedding. Hence, a portion of the SSH packets with payload length 48 bytes are converted into 64-octet packets. Similarly, a portion of the SSH packets with payload length between 49 and 64 are transformed into 80-octet packets. Figure 6 shows a sample output of the PT when embedding messages.



```
IP Header:
        Version        : 4
        Header Length  : 20 bytes
        Type of Service: 0x0
        Total Length   : 100 bytes
        ID             : 0x24e1
        Time to Live   : 64
        Protocol       : TCP
        Source         : 192.168.1.1
        Destination    : 192.168.1.44
TCP header:
        Source Port          : 32795
        Destination Port     : 22
        Header Length        : 32 bytes
        Flags                : PSH ACK
|       Sequence Number      : 788611435
        Acknowledment Number: 2147254624
Data (48 bytes):
        55 d5 dd 8d aa bc 48 1d 54 9a 18 8f a5 95 77 dd      U.....H.T.....w.
        98 32 55 36 2b 73 15 82 56 4b 75 2f c0 04 11 7c      .2U6+s..VKu/...|
        5a a0 cb 1c 7e d8 16 56 62 b1 81 e5 9b 69 ee 10      Z...~..Vb....i..
Application protocol: ssh

Hidden message (12 bytes):
        d3 2e 1e e5 38 47 af 88 6d ba 11 a0                  ....8G..m...
Data (64 bytes):
        58 0c 2e 8a d3 2e 1e e5 38 47 af 88 6d ba 11 a0      X.......8G..m...
        55 d5 dd 8d aa bc 48 1d 54 9a 18 8f a5 95 77 dd      U.....H.T.....w.
        98 32 55 36 2b 73 15 82 56 4b 75 2f c0 04 11 7c      .2U6+s..VKu/...|
        5a a0 cb 1c 7e d8 16 56 62 b1 81 e5 9b 69 ee 10      Z...~..Vb....i..
```

Fig. 6. Sample Output of the Packet Transmogrifier when Embedding Information in SSH Traffic where a 48-byte payload is enlarged to a 64-byte payload.

If *Eve* is able to observe both sides of the communication at the location where the PT is placed, it would be trivial to notice the difference in the payload size. The scenario is nevertheless implausible because of the high volume traffic on the Internet and the multitude of potential insertion points along the communication path, which make packet-by-packet comparison impractical.

Still, to avoid detection through automated tools when increasing the payload length of the packets, we need to simulate the packet length distribution of the SSH payload at any given time. We are currently adding capability to fit a given payload length distribution within a one-minute interval using the Chi-square test for goodness of fit. Therefore, we conclude that this stegomethod is not reasonably secure when *Eve* has knowledge of the covermessage payload length distribution.

6 Conclusions and Lessons Learned

In this paper, we have described semantics-preserving application-layer protocol steganography, and have presented methods for embedding secret messages in an application-layer protocol. We have developed the notions of syntax and semantics preservation in accordance to the goal of achieving a reasonably secure and robust stegosystem. We raised issues that evidence the need for definition of new theoretical paradigms of security. They must involve not only fitting the statistical profile of the cover itself but also fitting the statistical profile of how the transmission of the covers. Our approach has several advantages over prior work:

– Because of its applicability to a wide range of protocols, we can theoretically 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 practical ability of producing secure and robust stegomethods in network protocols.

As a proof-of-concept, we implemented an end-to-end protocol steganography approach in the SSH2 protocol as well as one with agents as middlemen. The SSH approach is not general, but represents a step toward finding generalized methods of embedding which is the ultimate goal of protocol steganography. The packet transmogrifier is a valuable contribution from the SSH implementations. It allows us to perform on-the-fly message embedding and extraction while a packet of arbitrary network traffic is en route. The software may be obtained from the authors upon request. In the near future, we will expand our family of embedder/extractor functions to include HTTP as well as other protocols.

References

1. Katzenbeisser, S., Petitcolas, F.A.: Information Hiding: Techniques for Steganography and Digital Watermarking. Artech House, Norwood, MA (2000)
2. Johnson, N.F., Jajodia, S.: Steganalysis: The investigation of hidden information. In: Proceedings of the IEEE Information Technology Conference, Syracuse, New York, USA (1998) 113–116

3. Anderson, R., ed.: Information Hiding: Proceedings of the First International Workshop. In Anderson, R., ed.: Lecture Notes in Computer Science 1174, Cambridge, U.K., Springer (1996)
4. Aucsmith, D., ed.: Information Hiding: Proceedings of the Second International Workshop. In Aucsmith, D., ed.: Lecture Notes in Computer Science 1525, Portland, Oregon, U.S.A., Springer (1998)
5. Moskowitz, I.S., ed.: Information Hiding: Proceedings of the Fourth International Workshop. In Moskowitz, I.S., ed.: Lecture Notes in Computer Science 2137, Pittsburg, PA, U.S.A., Springer (2001)
6. Oostveen, J., ed.: Information Hiding: Preproceedings of the Fifth International Workshop, Noordwijkerhout, The Netherlands (2002)
7. Pfitzmann, A., ed.: Information Hiding: Proceedings of the Third International Workshop. In Pfitzmann, A., ed.: Lecture Notes in Computer Science 1768, Dresden, Germany, Springer (1999)
8. Chapin, S.J., Ostermann, S.: Information hiding through semantics-preserving application-layer protocol steganography. Technical report, Center for Systems Assurance, Syracuse University (2002)
9. Kemmerer, R.: A practical approach to identify storage and timing channels: Twenty years later. In: Proceedings of the 18th Annual Computer Security Applications Conference (ACSAC 2002), San Diego, California (2002) 109–118
10. Dunigan, T.: Internet steganography. Technical report, Oak Ridge National Laboratory (Contract No. DE-AC05-96OR22464), Oak Ridge, Tennessee (1998) [ORNL/TM-limited distribution].
11. Handel, T., Sandford, M.: Hiding data in the OSI network model. In Anderson, R., ed.: Information Hiding: Proceedings of the First International Workshop, Cambridge, U.K., Springer (1996) 23–38
12. route@infonexus.com, alhambra@infornexus.com: Article 6. Phrack Magazine, 49 (1996) Retrieved on August 27, 2002 from the World Wide Web: http://www.phrack.com/phrack/49/P49-06.
13. Rowland, C.H.: Covert channels in the TCP/IP protocol suite. Psionics Technologies (1996) Retrieved on August 23, 2002 from the World Wide Web: http://www.psionic.com/papers/whitep03.html.
14. CAIDA.org: Characterization of internet traffic loads, segregated by application - OC48 analysis (2002) Retrieved on October 15, 2003 from the World Wide Web: http://www.caida.org/analysis/workload/byapplication/oc48/20020305/apps_perc_20020305/index.xml.
15. Katzenbeisser, S., Petitcolas, F.A.: Defining security in steganographic systems. In: Electronic Imaging, Photonics West, (SPIE). Volume 4675 of Security and Watermarking of Multimedia Contents IV. (2002) 50–56
16. Moskowitz, I.S., Longdon, G.E., LiWuChang: A new paradigm hidden in steganography. In: Proceedings of the New Security Paradigm Workshop 2000, Cork, Ireland (2000) 41–50
17. Cachin, C.: An information-theoreic model for steganography. Technical Report Report 2000/028 (2002) http://www.zurich.ibm.com/ cca/papers/stego.pdf.
18. Anderson, R.J., Petitcolas, F.A.: On the limits of steganography. IEEE Journal of Selected Areas in Communications **16** (1998) 474–481
19. Mittelholzer, T.: An information-theoretic approach to steganography and watermarking. In Pfitzmann, A., ed.: Information Hiding: Proceedings of the Third International Workshop. Volume 1768 of Lecture Notes in Computer Science., Dresden, Germany, Springer (1999) 1–16

20. Zöllner, J., Federrath, H., Klimant, H., Pfitzmann, A., Piotraschke, R., Westfeld, A., Wicke, G., Wolf, G.: Modeling the security of steganographic systems. In Aucsmith, D., ed.: Information Hiding: Proceedings of the Second International Workshop. Volume 1525 of Lecture Notes in Computer Science., Portland, Oregon, U.S.A., Springer (1998) 344–354
21. Ettinger, J.M.: Steganalysis and game equilibria. In Aucsmith, D., ed.: Information Hiding: Proceedings of the Second International Workshop. Volume 1525 of Lecture Notes in Computer Science., Portland, Oregon, U.S.A., Springer (1998) 319–328
22. Hopper, N., Langford, J., von Ahn, L.: Provably secure steganography. In Yung, M., ed.: Advances in Cyptology - CRYPTO 2002: Proceedings of the 22nd Annual International Cryptology Conference. Volume 2442 of Lecture Notes in Computer Science., Santa Barbara, California, U.S.A., Springer (2002) 77–92
23. Reyzin, L., Russell, S.: More efficient provably secure steganography. Cryptology ePrint Archive: Report 2003/093 (2003) http://eprint.iacr.org/2003/093/.
24. Fridrich, J., Goljan, M.: Practical steganalysis of digital images - state of the art. In: Proceedings of the SPIE Photonics West (Security and Watermarking of Multimedia Contents IV). Volume 4675., San Jose, California, USA (2002) 1–13
25. Provos, N., Honeyman, P.: Hide and seek: An introduction to steganography. IEEE Security & Privacy Magazine **1** (2003) 32–44
26. Simmons, G.J.: The prisoners' problem and the subliminal channel. In: Proceedings of CRYPTO '83, Plenum Press (1984) 51–67
27. Schneier, B.: Applied Cryptography. John Wiley & Sons, Inc (1996)
28. Pfitzmann, B.: Information hiding terminology. In Anderson, R., ed.: Information Hiding: Proceedings of the First International Workshop, Cambridge, U.K., Springer (1996) 347–349
29. Korn, F., Muthukrishnan, S., Zhu, Y.: Ipsofacto: A visual correlation tool for aggregate network traffic data. In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, ACM Press (2003) 677–677 Demonstration Session.
30. Ka0ticSH: Diggin em walls (part 3) - advanced/other techniques for bypassing firewalls. New Order (2002) Retrieved on August 28, 2002 from the World Wide Web: http://neworder.box.sk/newsread.php?newsid=3957.
31. Fisk, G., Fisk, M., Papadopoulos, C., Neil, J.: Eliminating steganography in Internet traffic with active wardens. In Oostveen, J., ed.: Information Hiding: Preproceedings of the Fifth International Workshop, Noordwijkerhout, The Netherlands, Springer (2002) 29–46
32. Bowyer, L.: Firewall bypass via protocol steganography. Network Penetration (2002) Retrieved on January 05, 2003 from the World Wide Web: http://www.networkpenetration.com/protocol_steg.html.
33. Bauer, M.: New covert channels in HTTP - adding unwitting web browsers to anonymity sets. In Samarati, P., Syverson, P., eds.: Proceedings of the 2003 ACM Workshop on Privacy in the Electronic Society, Washington, DC, USA, ACM Press (2003) 72–78 ISBN 1-58113-776-1.
34. Secure Shell Working Group, I.E.T.F.I.: The secure shell. Retrieved on October 26, 2003 from the World Wide Web: http://www.ietf.org/html.charters/secsh-charter.html (2003)
35. Barrett, D.J., Silverman, R.: SSH, The Secure Shell: The Definitive Guide. O'Reilly (2001)
36. Watterson, B.: Something Under the Bed is Drooling. Andrews and McMeel, pp. 101–104, Kansas City, MO (1988)