Semantics-Preserving Application-Layer Protocol Steganography [*]

Norka B. Lucena, Douglas F. Calvert, James Pease, Steve J. Chapin
Center for Systems Assurance
Syracuse University
111 College Place 3-114, Syracuse, NY 13244
{norka,jmpease,chapin}@ecs.syr.edu, dfcalver@maxwell.syr.edu

Abstract

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 paper, 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 efficacy of our approach, we have implemented protocol steganography within the Secure Shell (SSH) protocol.

Keywords: steganography, application protocols, semantics

1 Introduction

Steganography, from the Greek "covered writing," refers to the practice of hiding information within other information [14]. 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, 17, 19, 20]. Traditionally, most steganographic systems relied on the secrecy of the encoding system [22]. 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 [7].

An important consideration in the embedding process is whether it is semantics-preserving, i.e., whether the resulting message still conforms to the protocol specification. That property guarantees that if the message is interpreted at any point during its transmission, it will produce meaningful results. In addition to that, semantic preservation in modified messages helps to make them indistinguishable from unmodified 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.

Networking protocols are divided into multiple layers, as shown in Figure 1. The physical layer is responsible for communicating with the actual network

[*]This work was supported in part by the State of New York, and the CASE Center's SUPRIA program at Syracuse University.

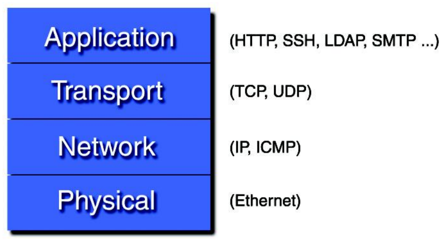


Figure 1: TCP/IP Protocol Suite Layers.

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), SSH (for secure login), LDAP (for distributed directory services), and HTTP (for web browsing, which alone accounts for approximately 70% of all Internet traffic).

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) changes to the message.

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.

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.

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

There are many potential applications for protocol steganography, considering when information hiding is used for both positive and negative means. When using information hiding for positive means, protocol steganography is appropriate to achieve private communication [29] and, in some cases, anonymity and plausible deniability, such as environments where censorship polices restrict web access [10]. More specifically, protocol steganography seems to be appropriate for environments where unobtrusive communications are required [14]. For example, 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. 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. When using information hiding with malignant purposes, the study of protocol steganography can help improving the design of network protocols and firewalls. Protocols can be harder to misuse. Firewalls can be harder to bypass.

The reminder of this paper is organized as follows. Section 2 describes the model for secret communication considered in protocol steganography and discusses its potential advantages. Section 3 presents a summary of the research to date and related work in relevant areas of steganography. Section 4 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. Section 5 analyzes future research opportunities in the area. Finally, Section 6 lists some conclusions.

2 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 [26] 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).

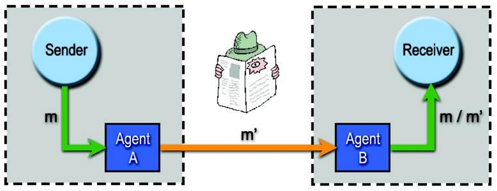


Figure 2: Framework for Secret Communication.

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.

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 [21], 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 Pfitzmann [21], the

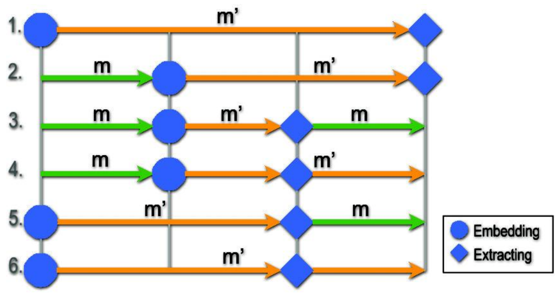


Figure 3: Message Paths.

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'.

The six possible sets of agent roles are as follows:

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 and restoration of the original cover message 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 while the message was in transit.

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 wherein 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 Adversary Models

Depending on the goals of steganalysis, adversaries can be *active* or *passive* [21]. 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 appropriately manipulating the bits of the messages that pass through them is enough (e.g., zeroing unused header fields).

Steganography systems consider both passive and active adversaries [2], while in watermarking and fingerprinting systems, generally, only active adversaries raise concern. However, most of the literature in stegosystems deals only 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.

3 Related Work

Handel and Sandford [12] 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 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 do 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.

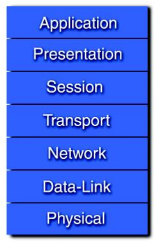


Figure 4: The OSI Idealized Network Model Layers.

Examples of implementation of covert channels in the TCP/IP protocol suite (see Figure 1) are presented by Rowland [24], Project Loki [23], Ka0ticSH [13], and more deeply and extensively by Dunigan [8]. 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 [8] 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 [24] 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 [8] 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 [13, 23] 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 reported in Fisk et al. [11]. 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.

Lastly, Bowyer [5] described a theoretical example without implementation, wherein a remote access Trojan horse communicates secretly with its control using an http GET request. To send data upstream to a faux webserver, a remote access Trojan horse could append data at the end of a GET request. Downstream communication is possible by sending back steganographic images, or embedding data within the HTML (e.g., in HTML tags). 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 has low bandwidth and can be blocked by restricting access to certain websites, or by scanning images for steganographic content.

4 A Case Study: SSH

The SSH protocol is defined by the Internet drafts [30, 31, 32, 33] of the Internet Engineering Task Force (IETF). It is a "protocol for secure login and other secure network services over an insecure network" [32]. 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.

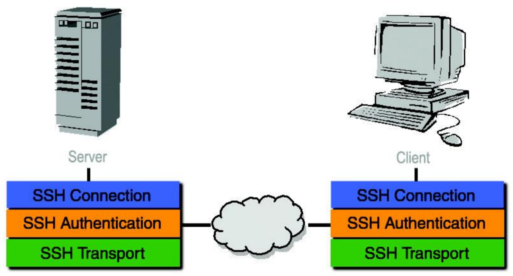


Figure 5: SSH2 Protocol Architecture.

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.

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 [33], each packet is composed of five fields:

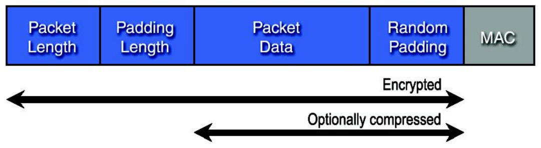


Figure 6: SSH2 Binary Packet Protocol.

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.

4.1 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 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 at the beginning of the encrypted part of the packet 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.

  This option offers the advantage of having

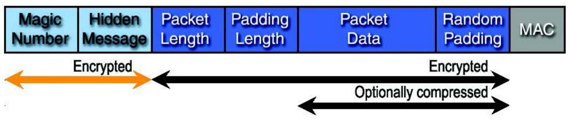


Figure 7: Adding an encrypted portion with a hidden message to a regular SSH packet.

  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 four octet magic number for our initial implementation, but this introduces a one in 4,096M chance that we will incorrectly interpret a cover-message as a stego-message.

4.2 Prototype Implementation

For implementing a prototype, two of the potential cases for information hiding discussed in the previous section were selected: the first one, generating a MAC-like message, and the last one, adding additional encrypted content to the packet. Both implementation were coded in C, tested under Red Hat 8.0, and each of them runs independently of the other.

4.2.1 Generating a MAC-like Message

This prototype of secret communication was implemented modifying version 3.5. of OpenSSH (http://www.openssh.org), a popular open-source SSH product. It 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 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 [14]. 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 is a proof-of-concept that illustrates 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.

4.2.2 Adding Additional Encrypted Content to the Packet

This prototype represents the case 3 of secret communication, described in Section 2 and shown in Figure 3. In this scenario, both agents, *A* and *B*, are middlemen located somewhere in the communication path. *Agent A* intercepts a message from the sender, embeds some secret message on it, and sends it back. *Agent B* extracts the hidden content and restores the message as it originally was before it reaches its destination. In order to be able to do that, we implemented a Packet Transmogrifier[2] (PT), inspired by the Calvin and Hobbes transmogrifier shown in Figure 8. The PT is a piece of software that embeds a message into an arbitrary stream of packets, and later extracts that hidden message.

In principle, the PT is conceived as a combination of several individual packet transformers (each of which could be used by an individual application to embed a message in a data stream). This 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 current implementation of the PT provides a series of default embedder and extractor functions, protocol dependent, that are called based on the options selected by the user and the application protocol of a particular IP packet.

The corresponding functions for handling SSH packets are called, `sshEmbedder` and `sshExtractor`, respectively. Both of them analyze SSH packets in a similar way, but execute opposite processes: embedding and extraction of the hidden content.

[2]With appropriate apologies and thanks to Bill Watterson, creator of "Calvin and Hobbes" [27].

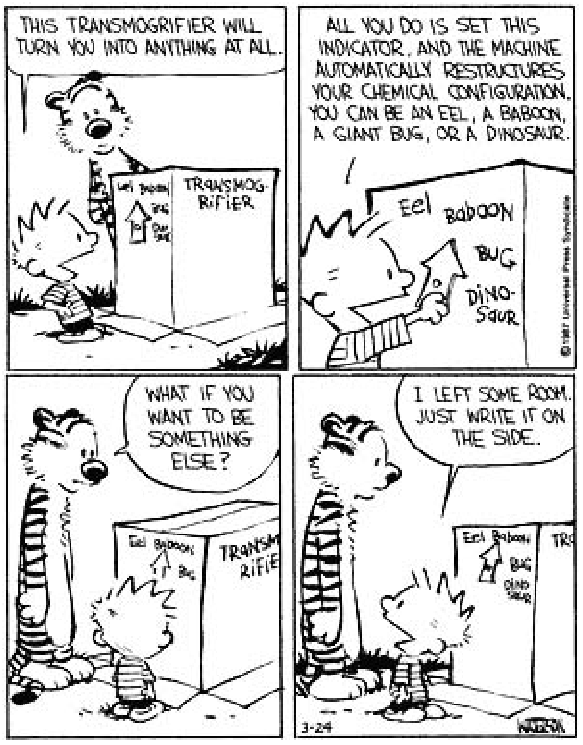

THIS TRANSMOGRIFIER WILL TURN YOU INTO ANYTHING AT ALL.
TRANSMOG-RIFIER
ALL YOU DO IS SET THIS INDICATOR, AND THE MACHINE AUTOMATICALLY RESTRUCTURES YOUR CHEMICAL CONFIGURATION. YOU CAN BE AN EEL, A BABOON, A GIANT BUG, OR A DINOSAUR.
Eel
BABOON
BUG
DINO-SAUR
WHAT IF YOU WANT TO BE SOMETHING ELSE?
I LEFT SOME ROOM. JUST WRITE IT ON THE SIDE.
3-24
WATTERSON

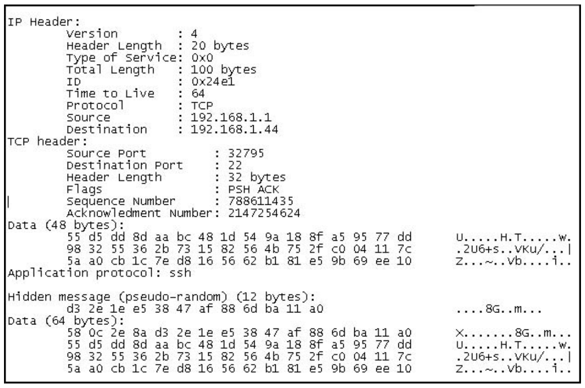
Figure 8: The Calvin and Hobbes Transmogrifier

When establishing an SSH session [4], initially the client contact the server and following both, client and server, disclose the SSH protocol version they support as well as the implementation version. Then they start communicating using the binary packet protocol mentioned in the SSH Transport Layer Protocol [33] and illustrated in Figure 6. The client and the server negotiate the algorithms to be used in the session, exchanging a list of supported algorithms and methods for key exchange, host key, encryption, compression, and message authentication code computation as well as a list of supported languages. Immediately after that, the key exchange process begins. The number of messages exchanged at this point between client and server depend on the chosen key exchange mechanism. All the packets sent until this point by client and server are unencrypted, therefore, `sshEmbedder` and `sshExtractor` are not interested in working with such packets. They analyze their content and discharge them if they correspond to any of those plaintext packets. Once the session key exchange is done, both sides, client and server, turn on encryption, perform authentication, and the secure connection is established. From that particular point, `sshEmbedder` begins altering the SSH packets, embedding content that looks encrypted. Such content, as mentioned in the previous section is composed of two parts: the hidden message itself and "magic" number that will allow `sshExtractor` to identify packets containing secret information. Conversely, `sshExtractor` checks for the existence of a "magic" number in every encrypted packet and if so, extracts it along with the hidden message and reformats the SSH packet to its original form.

We selected to have a hidden message of 12-byte length and a "magic" number of 4-byte length, which will add 16 bytes to the total SSH packet length. This amount of bytes seems small, considering that the maximum SSH packet size can be of 35,000 bytes, but it is large enough to allow us to have low probability of error when identifying a "magic" number, and high bandwidth considering that 12 bytes have been transmitted by every encrypted packet; that is, for every keystroke in a telnet-type session.

For testing purposes, the 12-byte hidden message is an arbitrary message generated pseudo-randomly; in actual use, this would be 12 octets of stego content. The "magic" number is calculated based on the message using a trapdoor one-way function (a special type of one-way function which uses a secret information *y* in *f(x)* [25]). Thus, computing the "magic" number involves performing a series of bitwise operations in the message using a secret key. Finally, both the "magic" number and the message are encrypted and added at the beginning of the SSH payload.

Figure 9 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 (pseudo-random) (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..
```

Figure 9: Sample Output of the Packet Transmogrifier when Embedding Information in SSH Traffic

Because the SSH payload is 16 bytes larger after embedding, the total length of the IP packet is also increased in the same amount of bytes. On the other hand, when extracting the IP total length is decreased.

The implementation of this case of in-transit protocol steganography can be used in many scenarios since there are no assumptions about the communicating parties. It allows us to take advantage of any arbitrary SSH traffic as long as the packet transmogrifier is placed somewhere along that communication path.

4.3 Discussion

The implemented approaches, although simple, represent 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, the modified packet looks "normal" to simple traffic monitoring, although depending on the point of observation a difference on packet length can be noticed for the second approach. In spite of that, several issues need to be discussed and some other requires further exploration.

When generating a MAC-like message, the first issue of concern is the impossibility of verifying that 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 [33], 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 [14], 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. Furthermore, 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. In the same order of ideas, adding an additional encrypted content to the packet is robust because recomputing the MAC involves the knowledge of the actual payload. In consequence, 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. [11] (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.

There is some controversy in the field about what is the better way of defining a perfectly secure steganography system, as reported by Moskowitsz et. al [18] and Katzenbeisser and Petitcolas [15]. 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 [6, 2, 16, 34] and some game-theoretical definitions [9] 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 on semantic preservation; information-theoretical analysis must be done.

5 Future Work

To this point, we are in the process of gathering data to estimate the distribution model of the covers and the stego-messages. The purpose of that is to determine whether or not a passive adversary with knowledge of the distributions and computational power to compare them is able to prove the existence of a hidden message.

In addition to that, we are incorporating the packet transmogrifier into a Linux-based router to demonstrate the efficacy of in-transit protocol steganography. One important issue related to that is the selection of a proper location to place the transmogrifier. A good location will minimize its detection as well as the detection of modified packets, such as in cases where the packet length has been changed.

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 [28, 29] 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).

6 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 an end-to-end protocol steganography approach in the SSH2 protocol as well as one with agents as middlemen. 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.

References

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

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

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

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

[5] L. Bowyer. 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.

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

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

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

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

[10] N. Feamster, M. Balazinska, G. Harfst, H. Balakrishnan, and D. 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.

[11] G. Fisk, M. Fisk, C. Papadopoulos, and J. Neil. Eliminating steganography in Internet traffic with active wardens. In J. Oostveen, editor, *Information Hiding: Preproceedings of the Fifth International Workshop*, pages 29–46, Noordwijkerhout, The Netherlands, October 7-9, 2002. Springer.

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

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

[14] S. Katzenbeisser and F. A. Petitcolas. *Information Hiding: Techniques for Steganography and Digital Watermarking*. Artech House, Norwood, MA, 2000.

[15] S. Katzenbeisser and F. A. Petitcolas. Defining security in steganographic systems. In *Electronic Imaging, Photonics West, SPIE*, volume 4675 of *Security and Watermarking of Multimedia Contents IV*, pages 50–56, 2002.

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

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

[18] I. S. Moskowitz, G. 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.

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

[20] A. Pfitzmann, editor. *Information Hiding: Proceedings of the Third International Workshop*, Dresden, Germany, September 29-October 01, 1999. Springer.

[21] B. Pfitzmann. Information hiding terminology. In R. Anderson, editor, *Information Hiding: Proceedings of the First International Workshop*, pages 347–349, Cambridge, U.K., May 30-June 01, 1996. Springer.

[22] N. 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.

[23] 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.

[24] C. 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.

[25] B. Schneider. *Applied Cryptography*. John Wiley & Sons, Inc, 1996.

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

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

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

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

[30] 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.

[31] 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.

[32] 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.

[33] 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.

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