Cloak and Dagger: Man-In-The-Middle and Other Insidious Attacks

Abstract

One of the most devastating forms of attack on a computer is when the victim doesn't even know an attack occurred. After some background material, various forms of man in the middle (MITM) attacks, including ARP spoofing, fake SSL certificates, and bypassing SSL are explored. Next, rootkits and botnets, two key pieces of crimeware, are introduced and analyzed. Finally, general strategies to protect against such attacks are suggested.

Keywords: Man-in-the-middle (MITM) Attack, ARP spoofing, rootkits, botnets, Mebroot, Torpig.

1 Introduction

Information has always been very valuable. Computers are entrusted to maintain and process massive amounts of information. This makes them valuable targets to attackers. One of the most devastating forms of attack is when an attacker gains access to the information without the victim even being aware of it.

This paper explores some of the means by which this surreptitious access to information can occur. Background material on basics of cryptography, the Diffie-Hellman key exchange, networking, Transport Layer Security and Secure Sockets Layer, and drive by downloads is provided in section 2. MITM attacks are defined in section 3. ARP spoofing, a form of a MITM attack, is explored in section 3.1. Futile defenses to MITM attacks are examined in section 3.2. A MITM attack on SSL using fake certificates is givenin section 3.3. Even more forms of MITM attacks are explored in section 3.4. Defenses are discussed in section 3.5. Finally, a new attack known as man in the browser is detailed in section 3.6.

MITM attacks are not the only stealthy means by which information security is breached. Rootkits and botnets, which are capable of doing much more harm, can reside on victim's computer while evading detection. Rootkits are defined in section 4. An

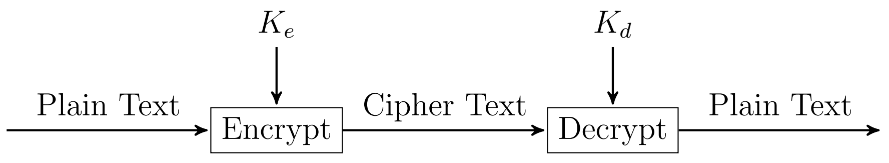


Figure 1: Process of encryption and decryption.

example rootkit, Mebroot, is analyzed in section 4.1. Defenses against rootkits are discussed in section 4.2. Botnets, which are often used in conjunction with rootkits, are defined in section 5. Attacker's motivation is examined in section 5.1. The Torpig botnet, and its recent takeover by security researchers, is investigated in 5.2.

We conclude with some general discussion on how to prevent these attacks in section 6.

2 Background

In this section, we begin with the basics of cryptography, pointing out the difference between symmetric and asymmetric encryption, followed by a description of the Diffie-Hellman key exchange protocol. Next, we present an abstract description of the man-in-the-middle attack. After that, we give some networking details that are necessary to understand a concrete man-in-the-middle attack on modern local-area networks. We first begin with a general discussion on cryptography.

2.1 Cryptography

When trying to communicate a message across an untrusted channel, cryptography is a natural solution. The original message, or *plain text*, is transformed into *cipher text* by encrypting the plain text with an encryption key, K_e. The cipher text will appear meaningless with no apparent relationship to the plain text. This allows the cipher text to be transferred across the untrusted channel with minimal risk of the plain text being intercepted. The cipher text can be transformed back into the plain text by decrypting it with the decryption key K_d. Collectively, the methods of encryption and decryption are known as a *cipher*. This process, illustrated in Figure 1, can be represented symbolically as

$$P = D(K_d, E(K_e, P)).$$

If the decryption key is the same as the encryption key, or efficiently derived from it, the cipher is known as a symmetric cipher; otherwise, it is an asymmetric cipher. Popular symmetric ciphers include the Advanced Encryption Standard (National

Institute of Standards and Technology, 2001) and the Triple Data Encryption Algorithm, commonly known as Triple DES (National Institute of Standards and Technology, 1999). Symmetric ciphers suffer from the key distribution problem–getting the communicating parties to agree upon a common key.

In public key cryptography, which use asymmetric ciphers, each communicating entity maintains one private key and one public key, K_{priv} and K_{pub} respectively. Extending the previous notation, public key cryptography is

$$P = D(K_{priv}, E(K_{pub}, P)).$$

As the names imply, the public key is made available freely to anyone who wishes to use it, but the private key is kept secret. If Alice wishes to communicate with Bob, she encrypts the message with Bob's public key, which is freely available, and sends the encrypted message to Bob. Anyone eavesdropping on this communication cannot decrypt the message unless they have Bob's private key. Anyone who wants to communicate with Bob can easily get access to his public key, so public key cryptography does not suffer from the key distribution problem. However, public key cryptography does have a different drawback. It is computationally expensive; usually this entails performing modular arithmetic over large integers that are few hundred digits long. In practice, a hybrid method is used: public key cryptography is used initially to exchange a random symmetric key, and this random key is used for the remainder of the session. Popular asymmetric ciphers include RSA (Rivest, Shamir, & Adleman, 1978) and El Gamal, 1985.

Cryptography does more than just keep messages confidential. It can be used to authenticate the sender of a message, and verify it has not been altered (National Institute of Standards and Technology, 2009). In particular, public key cryptography can be used for this task. The public and private keys can be applied in the reverse order as

$$P = D(K_{pub}, E(K_{priv}, P)).$$

If Bob sends Alice $E(K_{priv}, P)$, then Alice can be assured that the message P came from Bob as only Bob has access to K_{priv}. In this case, P is said to be *digitally signed* by Bob. See Stinson, 2005, Ferguson & Schneier, 2003, and Menezes, van Oorschot, & Vanstone, 1997 for more background on cryptography and its applications.

2.2 Diffie-Hellman Key Exchange Protocol

The Diffie-Hellman (DH) protocol allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure communication channel (Diffie & Hellman, 1976). Diffie-Hellman is also known as Diffie-Hellman-Merkle (Hellman, 2002). In short, DH is based on the fact that

$$(g^a)^b \equiv (g^b)^a \pmod{p},$$

where all computations are performed over a group of integers modulo p for some large prime p. This is true because multiplication in groups is associative. DH's cryptographic strength comes from the fact that it is easy to compute powers modulo a prime, but hard to reverse the process when large integers are involved. This intractable problem is known as the *discrete log problem*.

Alice and Bob can agree on a shared secret by performing the following steps (all arithmetic is modulo p):

1. Alice and Bob agree on a large prime p, and a generator g.
2. Alice picks a random number a, $0 < a < p$, and sends g^a to Bob. Alice keeps a private.
3. Bob picks a random number b, $0 < b < p$, and sends g^b to Alice. Bob keeps b private.
4. Alice computes $(g^b)^a$.
5. Bob computes $(g^a)^b$.

Both Alice and Bob are now in possession of g^{ab}, which is their shared secret key. Only a, b and $g^{ab} = g^{ba} \mod p$ are kept private. All other values–p, g, g^a, and g^b–are public.

2.3 Networking Basics

The Transmission Control Protocol (TCP) and the Internet Protocol (IP) together are at the heart of communication protocols used for the Internet. These protocols resulted from years of research funded by Defense Advanced Research Projects Agency (DARPA). The TCP/IP suite defines a set of rules that enable computers to communicate over a network. The rules specify data formatting, addressing, shipping, routing and delivery to the correct destination.

Computers that are in close proximity and connected into the same Local Area Network (LAN) communicate with each other using Ethernet. In this protocol, frames are sent to a destination Media Access Control (MAC) address, a 48-bit address that is unique to each Network Interface Card (NIC) on the network. The nodes on a LAN are connected using a *hub* or a *switch*. The only difference between them is that a hub is less intelligent and cheaper than a switch. It simply broadcasts every packet it receives to every computer on the LAN. For many years, hubs were very common and posed serious security problems for system administrators. Anyone on the LAN could put their NIC into "promiscuous" mode and eavesdrop on all data transferred on the LAN. Switches, on the other hand, send Ethernet frames only where they need to go instead of broadcasting to everyone. In addition to improved security, switches also increase the rate at which data can be transferred.

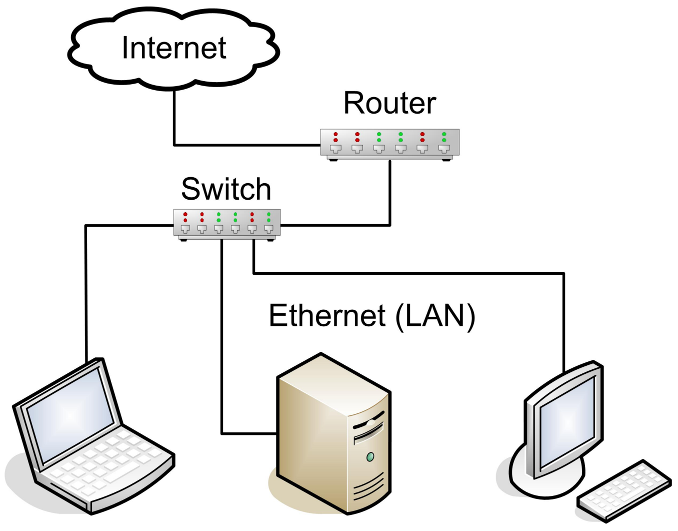


Figure 2: LAN connected to the Internet via a router. The router is seen by the switch as another host on the LAN.

To connect a LAN to the Internet, one needs a more intelligent device that can route packets to the Internet. This device is called a *router*. It is smarter than a switch, in the sense that it is programmable, and usually includes an interface by which it can be configured. Routers have the ability to communicate with other routers and determine the best way to send network traffic from one point to another on the Internet. For simplicity, let us assume that there is only one router on any given LAN. Then, since all traffic from the LAN must enter and exit through the router, it provides a useful choke point. The computers on the LAN can be protected from outside attackers by running a firewall along with an intrusion detection system at this choke point. This is illustrated in Figure 2.

A *default gateway* is the node on the LAN that is chosen by the switch when it encounters an IP address that does not belong to any node on the LAN. A router usually assumes the role of a default gateway. In home networks, the functionality of a switch, router, and wireless access point are often combined into one physical unit.

2.4 TLS/SSL

Transport Layer Security (TLS) is a security protocol from the Internet Engineering Task Force (IETF) that is based on the Secure Sockets Layer (SSL) 3.0 protocol developed by Netscape. TLS is the successor to SSL. Both protocols include cryptographic frameworks which are intended to provide secure communications on the Internet. SSL is not an industry standard as it was developed by Netscape. TLS is the widely recognized standard issued by the IETF for securing transmitted data. The current version of TLS is 1.2 and is described in RFC 5246 (Dierks & Rescorla, 2008). Version 1.1 of TLS is supported on most commercial browsers, web and email servers with support for version 1.2 forthcoming. By and large, TLS and SSL are interchangeable.

The TLS protocol runs above TCP/IP and below higher-level protocols such as HTTP or SMTP. It uses TCP/IP on behalf of the higher-level protocols, and facilitates the establishment of an encrypted connection between the client and server.

Both TLS and SSL follow a standard handshake procedure to establish communication. The handshake prior to an HTTPS session follows.

1. The client contacts a server that hosts a secured URL.
2. The server responds to the client's request and sends the server's digital certificate (X.509) to the browser.
3. The client now verifies that the certificate is valid and correct. Certificates are issued by well-known authorities (e.g. Thawte or Verisign).
4. The server could optionally choose to confirm a user's identity. Using the same techniques as those used for server authentication, TLS-enabled server software can check that the client's certificate is valid and has been issued by a certificate authority (CA) listed in the server's list of trusted CAs. This confirmation might be important if the server is a bank sending confidential financial information to a customer and wants to check the recipient's identity. (See the benefits of performing this optional step in Section 3.5.)
5. Once the certificate is validated, the client generates a random one-time session key, which will be used to encrypt all communication with the server.
6. The client now encrypts the session key with the server's public key, which was transmitted with the digital certificate. Encrypting using the server's public key ensures that others cannot eavesdrop on this sensitive exchange.

At this point, a secure session is established because the client and server both know the session key. Now, both parties can communicate via a secure channel. See Figure 3.

2.5 Drive-by downloads

A *drive-by download* is a catch-all term for software downloaded to your computer without your knowledge or intervention (Walker, 2005). Attackers often do this by exploiting the way Uniform Resource Locators (URLs) are handled by the browser. That is, a webpage link would be created that contains unusual or excessive set of characters. When a vulnerable browser attempts to parse this carefully crafted URL,

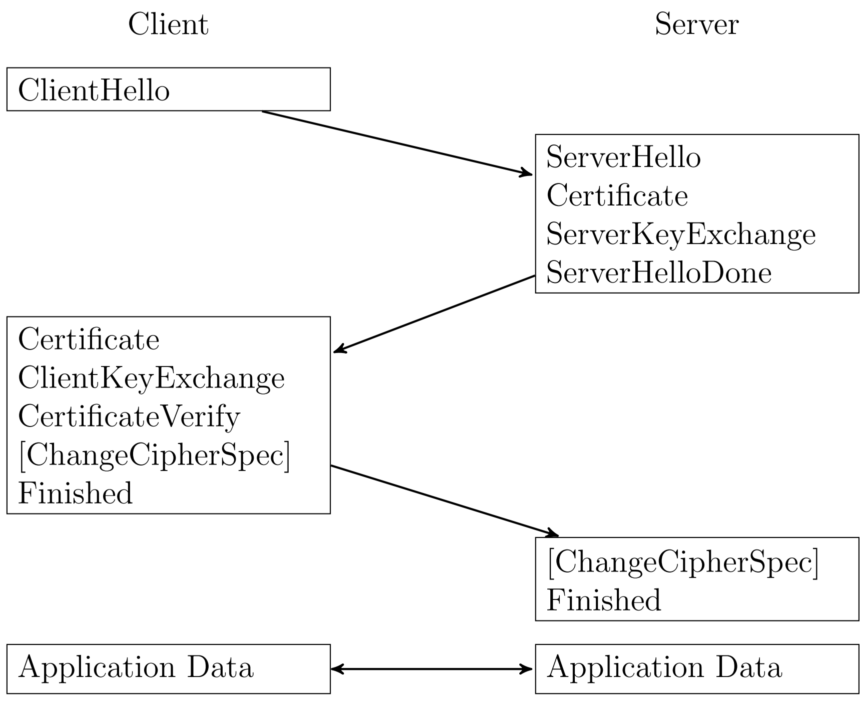


Figure 3: SSL/TLS protocol handshake and session key establishment. (Adapted from Dierks & Rescorla, 2008.)

the attacker can gain access to the victim's computer by running code outside the browser environment.

Once the attacker can execute code on the victim's computer, there is little restriction as to what that code can do. For example, a rootkit could be installed. The victim could be forced to become a botnet client. Advertising software could be installed that monitors the victim's browsing behavior and bombard them with pop-ups. Keystroke loggers could be installed which could lead to the capture of logins and passwords to various things such as bank accounts, email accounts, or credit card numbers.

The best defense against drive-by downloads is to keep the operating system and web browser up to date with the latest security patches. Since these mostly exploit the web browser, it is only a matter of time before the security community learns of the specific attack used and the appropriate software is fixed.

3 Trust, Certificates, and Man in the Middle (MITM) Attacks

Suppose that Alice wishes to communicate with Bob using public key cryptography. Mallory, the attacker, can participate actively or passively. In the latter role, she faithfully proxies the communication between Alice and Bob, while eavesdropping on their conversation–a breach of confidentiality. In the former role, Mallory can choose to edit, delete, or inject packets.

If Alice requests Bob's public key and Mallory is able to intercept it, then Mallory responds back to Alice with her public key, K_m. Alice is under the impression that she is talking to Bob and encrypts all her messages with K_m which Mallory can decrypt.

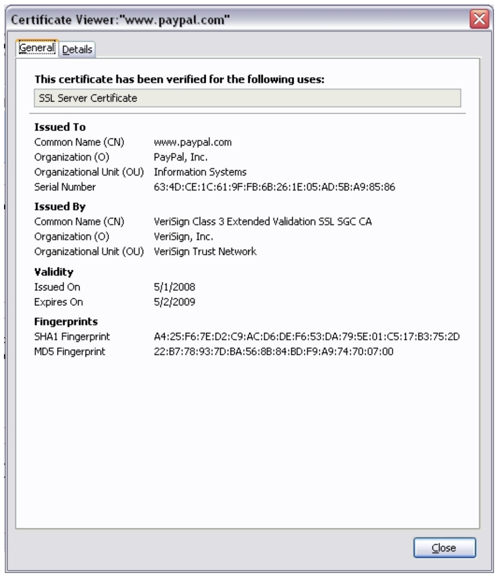
Meanwhile, Mallory, pretending to be Alice, sends K_m to Bob, telling him that it is Alice's public key and requests his public key. Bob, like Alice, encrypts all his messages with K_m which Mallory can decrypt.

Both Bob and Alice are under the impression that they are talking to each other, but all communication passes through Mallory and is completely controlled by Mallory. The attack mounted by Mallory is known as a *man in the middle attack.*

This problem arose because the public keys are sent directly by their owners. One solution is to exchange public keys through a trusted third party. This is accomplished by using *digital certificates* that contain the public key for an entity and an assurance from a trusted third party that the public key belongs to that entity. The trusted third party that issues digital certificates is called a *Certification Authority* (CA). As these certificates are digitally signed by CAs, the certificates provide protection against impersonation. Authenticity of certificates is easily verified since a CA's public key is "universally" available. For example they are embedded in browsers. When a certificate is for an individual entity (resp. CA), the certificate is a *personal (resp. root) certificate.*

Digital certificates contain at least the following information about the entity being certified:

- the public key of the certificate holder
- the common name of the certificate holder
- the common name of the CA that is issuing the certificate
- the date certificate was issued on
- the expiration date of the certificate
- the serial number of the certificate



This certificate has been verified for the following uses:

SSL Server Certificate

Issued To

Common Name (CN)	www.paypal.com
Organization (O)	PayPal, Inc.
Organizational Unit (OU)	Information Systems
Serial Number	63:4D:CE:1C:61:9F:FB:6B:26:1E:05:AD:5B:A9:85:86

Issued By

Common Name (CN)	VeriSign Class 3 Extended Validation SSL SGC CA
Organization (O)	VeriSign, Inc.
Organizational Unit (OU)	VeriSign Trust Network

Validity

Issued On	5/1/2008
Expires On	5/2/2009

Fingerprints

SHA1 Fingerprint	A4:25:F6:7E:D2:C9:AC:D6:DE:F6:53:DA:79:5E:01:C5:17:B3:75:2D
MD5 Fingerprint	22:B7:78:93:7D:BA:56:8B:84:BD:F9:A9:74:70:07:00

Figure 4: Digital certificate received from PayPal web server as viewed from a browser.

For obvious reasons, digital certificates do not contain the private key of the owner because it must be kept secret by the owner. See an example certificate in Figure 4.

A Public Key Infrastructure (PKI) is a system to facilitate the use of public key cryptography. Unfortunately there is some potential semantic confusion with the term (Anderson, 2008). It can mean either a key infrastructure that is public, or an infrastructure for public keys. Accordingly, there is no one standard that defines the component of a PKI. Typically, PKI refers to the former interpretations. In that case CAs and Registration Authorities (RA) normally provide the following:

- issue digital certificates
- validate digital certificates
- revoke digital certificates
- distribute public keys

The X.509 protocol suite is an International Telecommunication Union standard for a Public Key Infrastructure (Cooper et al., 2008). RAs verify the information provided at the time when digital certificates are requested. If the information is verified successfully by the RA, the CA can issue a digital certificate to the requester.

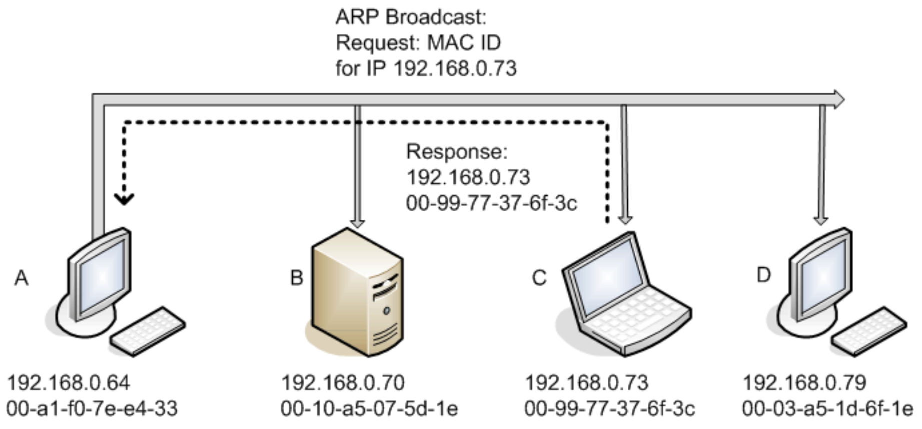


Figure 5: ARP request broadcast and response. Here host A is requesting a MAC (physical) address that corresponds to IP# 192.168.0.73 (host C).

3.1 MITM Attack on a Switched LAN using ARP Spoofing

How does Mallory intercept and relay communication between Alice and Bob on modern computer networks? Aren't they built on secure technology? The answer is no, unfortunately. The problem is that Ethernet, upon which virtually all modern LANs are based, was designed without any sort of authentication mechanism. An attack known as *ARP spoofing* takes advantage of this weakness and can intercept communications on a LAN running the Ethernet protocol (Wagner, 2001). This attack works against most networks that are in use at the time of this writing.

Recall our discussion from Section 2.3 on how two computers communicate on a LAN using Ethernet frames. The attack works as follows. To connect to a LAN, each host must be equipped with a Network Interface Card (NIC). Each NIC is assigned a unique Media Access Control (MAC) address by the manufacturer. Communication on Ethernet takes place by sending frames to destination MAC addresses. If a MAC address is unknown, the source node broadcasts an Address Resolution Protocol (ARP) request. This request specifies an IP address and asks the host with this IP address to reply back with its physical address. In other words, ARP finds a MAC address given an IP address.

Every node on the LAN receives every ARP request, but only the host with the matching IP address replies back with its physical address; the rest simply ignore it. The response is sent back using an ARP Reply that contains the requested IP number and the corresponding MAC address. When the source node receives this information, it stores it in a table of IP and MAC address pairs. This table is known as the *ARP cache* and the mappings are considered valid for a fixed amount time, after which they expire and are removed. Every node on the LAN maintains such a cache. Note that the source node enters the IP-MAC address pair contained in the ARP Reply into its cache without any validation or further checks. Put differently, there is total trust between the nodes on a LAN. To make the matters worse ARP is a *stateless protocol*, that is an ARP Reply is not matched to see if there are outstanding ARP Requests. Therefore, any malicious node can takeover a LAN and route all traffic through itself by simply manipulating cache entries at various hosts. The only requirement is that the malicious node is a host on that LAN. One easy way is to accomplish this is by connecting to an insecure wireless access point. Many corporations, hospitals, and retail outlets still use easily breakable WEP encryption (Tews, Weinmann, & Pyshkin, 2007). This weakness exists within the TCP/IP stack. Hence, it is a multi-platform vulnerability.

By injecting merely two ARP reply packets into a LAN, any malicious node M can control all traffic going back and forth between any two nodes on that LAN. For example, between an unsuspecting victim node A and the default gateway G. First, M sends G a spoofed ARP Reply $\langle \text{IP}_A, \text{MAC}_M \rangle$ claiming that it was assigned IP_A (which really belongs to A) but gives its own MAC address, MAC_M. The gateway would blindly replace its current correct entry with the spoofed one. At the same time M would send a similar spoofed ARP Reply, $\langle \text{IP}_G, \text{MAC}_M \rangle$, to A, replacing the correct ARP cache entry for the gateway computer at A with the spoofed one. From this point on, any traffic from A bound for the default gateway would instead go to the attacking computer M. Similarly, all traffic from G destined to A is routed instead to M. Neither A nor G would be aware of the intermediary that is relaying the traffic in the middle. See Figure 6.

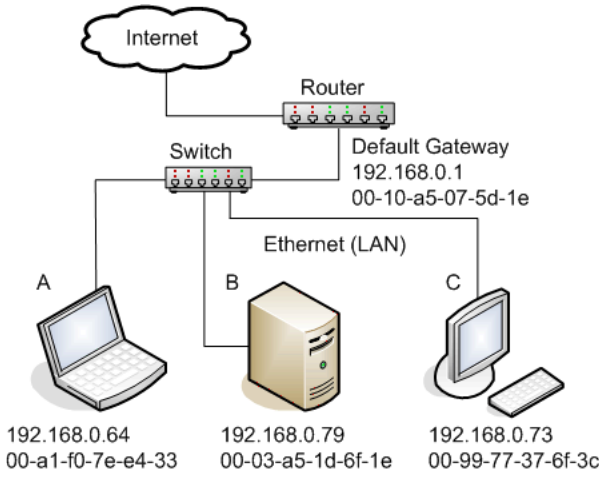
On a LAN with n nodes, that consists of $(n-2)$ nodes, 1 router, and 1 attacker, by inserting $2(n-2)$ spoofed ARP Replies, the attacker can take full control of the traffic destined to the Internet from that LAN. This process of inserting false entries into an ARP cache is also referred to *ARP poisoning*. It is worth noting that cache entries are purged after a timeout period. Therefore, to keep control of the network, the attacker must periodically poison each host for the duration of the hijacked session.

In addition to compromising the confidentiality and the integrity of the data as it passes through the local network, MITM attacks can also adversely affect availability. Either by simply slowing down, or completely dropping the network communication by associating a nonexistent MAC address to the IP address of the victim's default gateway.

3.2 Futile Defenses against MITM

It has become fashionable at many financial institutions in the United States to present the online user with a set of "secret" questions, in addition to their login credentials. After a successful login, the session might proceed along the following lines:

> To protect the security of your account, please answer the following questions:



After C inserts itself between B and the Default Gateway

At the default gateway

192.168.0.1	00-10-a5-07-5d-1e
192.168.0.64	00-a1-f0-7e-e4-33
192.168.0.73	00-99-77-37-6f-3c
192.168.0.79	*00-99-77-37-6f-3c*

ARP Cache (at every node)

192.168.0.1	00-10-a5-07-5d-1e
192.168.0.64	00-a1-f0-7e-e4-33
192.168.0.73	00-99-77-37-6f-3c
192.168.0.79	00-03-a5-1d-6f-1e

At B

192.168.0.1	*00-99-77-37-6f-3c*
192.168.0.64	00-a1-f0-7e-e4-33
192.168.0.73	00-99-77-37-6f-3c
192.168.0.79	00-03-a5-1d-6f-1e

Figure 6: ARP cache values before and after poisoning by node C to insert itself between B and the Default Gateway (Router). The second column shows after ARP poisoning. The two spoofed entries are emphasized.

Note: Your answers are NOT case sensitive.

What is the name of the school where you went to kindergarten?

Or questions such as

What is the last name of your favorite actor?

What is your favorite color?

Sometimes this “extra” security comes in the form of storing your favorite picture which is transmitted during the beginning of an encrypted session.

These additional measures are totally ineffective against MITM attacks. In the next section, an attack by Marlinspike is presented that defeats them.

3.3 MITM Attack on SSL Using Bogus Certificates

A *certificate chain* is a list of certificates used to authenticate an entity. Certificate chaining is a process by which *root* certificate authorities delegate the certificate issuing authority to intermediate CAs for efficiency and scalability reasons. This mechanism is part of the trusted computing paradigm. When certificate chains are involved in verification, to check authenticity of a certificate for an entity, the certificate chain is used to reach the *root* CA certificate. The root CA certificate is self-signed. However, the signatures of the intermediate CAs must be verified.

Chains can be longer than three. Most browsers verify certificate chains as follows:

1. Verify that the name on the certificate matches the name of the entity the client wishes to connect.
2. Check the certificate's expiration date.
3. Check the signature. If the signing certificate is in the list of root CAs in the client, stop, otherwise, move up the chain one link and repeat.

Assume an attacker is in possession of the domain `attacker.com` and a certificate is issued to it by CA$_2$. Consider the following certificate chain:

$$\text{Root CA} \rightarrow \text{CA}_1 \rightarrow \text{CA}_2 \rightarrow \texttt{attacker.com} \rightarrow \texttt{victim.com}$$

Anyone connecting to `victim.com`, first checks its name and expiration, and then verifies its signature by applying the public key of `attacker.com`. Assuming that this is successful, the process is repeated with `attacker.com`, CA$_2$, and CA$_1$, until Root CA is reached. In this example, all signatures and dates pass the validity test, and the Root

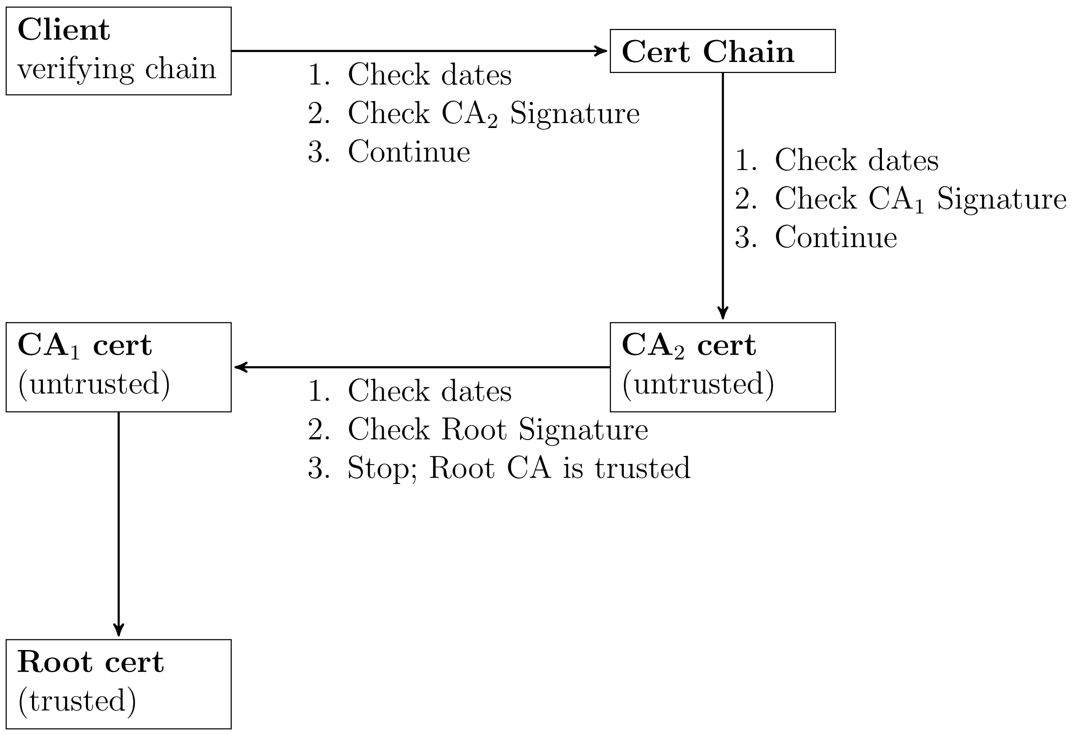


Figure 7: Certificate chain verification process by a client program. (Adapted from Figure 1 of IBM, 2009.)

CA would be reached successfully. Since the Root CA is always trusted, the whole chain is considered to be intact. Unfortunately there is a problem; `attacker.com` should not have the authority to issue certificates to other domains. This restriction is imposed in the Basic Constraints Extension of the X.509 specification (Cooper et al., 2008). It identifies whether the subject of the certificate is a CA and length of a certificate chain, including itself. The intent in the Standard is to prevent non-CAs from issuing certificates. For non-CAs, this field should be

```
CA:FALSE
```

indicating that the entity to which this certificate was issued is not a CA. Unfortunately many CAs did not explicitly set this field and most browsers simply ignored it. The implication of this careless practice is that any valid certificate could create a certificate for *any* other domain.

In 2002, Marlinspike released a software tool, *sslsniff*, that took advantage of this weakness. This tool has the capability to dynamically generate certificates for domains that are being accessed on the fly. The new certificate becomes part of a certificate chain that is signed by any certificate provided to sslsniff.

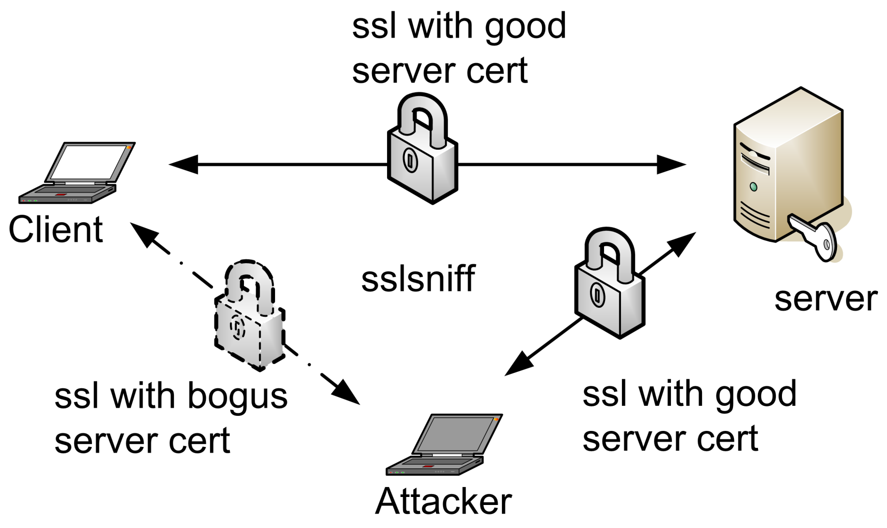

ssl with good server cert
Client
sslsniff
server
ssl with bogus server cert
ssl with good server cert
Attacker

Figure 8: MITM attack on secure web sessions using bogus certificates

Using sslsniff, one can perform MITM attack on an HTTPS session as follows. First, an HTTPS request from victimClient trying to connect to victimServer is intercepted using standard techniques such as ARP poisoning. The attacker then sends a bogus certificate in the name of victimServer. Unsuspecting, victimClient authenticates the certificate chain and sends a symmetric key, encrypted using the public key supplied by the attacker. The attacker decrypts the symmetric key, which is used as a session key. Simultaneously, the attacker opens an HTTPS session with victimServer and proxies the traffic between victimClient and victimServer, relaying the set "secret" questions and answers back and forth. All the data that is in transmitted between the client and the server is available to the attacker *in the clear* including sensitive information such as credit card numbers.

This weakness in the Basic Constraints field of X.509 has since been addressed by the CAs and the newer generation of popular browsers.

3.4 MITM Attack Using Other Means

Even though one may not be able to carry out MITM attacks using bogus certificates against newer web technology without raising too many red flags, there are a variety of other techniques that one can employ to launch an MITM attack and breach the confidentiality of secure web transactions. The techniques presented here are browser independent and are effective against web sites of some leading financial institutions.

Since it now appears as if HTTPS has been secured, what is the best way to hijack a web session? Marlinspike, 2009 provides an answer to this question by asking the following questions related to human-computer interaction (HCI):

1. How do people start an HTTPS session?
2. How are people assured that they are using a secured session?
3. How are people warned that there maybe a problem with the security of the session?

Most often, the answer to question 1 is either

1. User clicking on a button that posts to HTTPS, or
2. Through rerouting from the web server (HTTP response code 302). When the user types `victimServer.com`, the browser resolves it to `http://www.victimServer.com`. For example, the exchange might look like

   ```
   GET /index.html HTTP/1.1
   Host: www.victimServer.com
   ```

   When victimServer receives the above request, it reroutes the client as

   ```
   HTTP/1.1 302 Found
   Location: https://www.victimServer.com/index.html
   ```

That is, no one really types `https://` before starting an online transaction. In other words, access to HTTPS is via HTTP. The strategy of the attacker becomes, attack HTTP if HTTPS is secure.

Questions 2 and 3 can be best understood by studying how browsers have evolved over the years. Seven years ago, when sslsniff was released, excessive positive feedback was given by the browser that a user was using a secure connection. There were many lock icons, the address bar or uniform resource locator (URL) bar changed color, and a number of other indicators were deployed to give a "warm-and-fuzzy" feeling to the user that the page was secure. A *favicon*, short for favorites icon, is a 16x16 pixel square icon associated with a particular website that is displayed in the URL bar. A popular favicon in the older browsers during secure sessions was a small padlock. See Figure 9.

Another example of positive feedback is as follows. When a bogus certificate is detected by the browser, a dialog similar to the one shown in Figure 10 is presented to the user. Notice that by default, the certificate chain would be accepted for the session. According Marlinspike (2009), users typically click through these warning dialogs when they don't completely understand the meaning of the warning.

The trend in the newer browsers is to scale back the positive feedback while emphasizing the negative. For instance, instead of encouraging the user to simply click

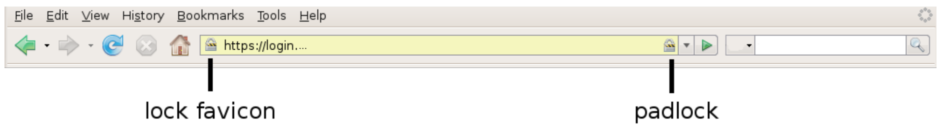
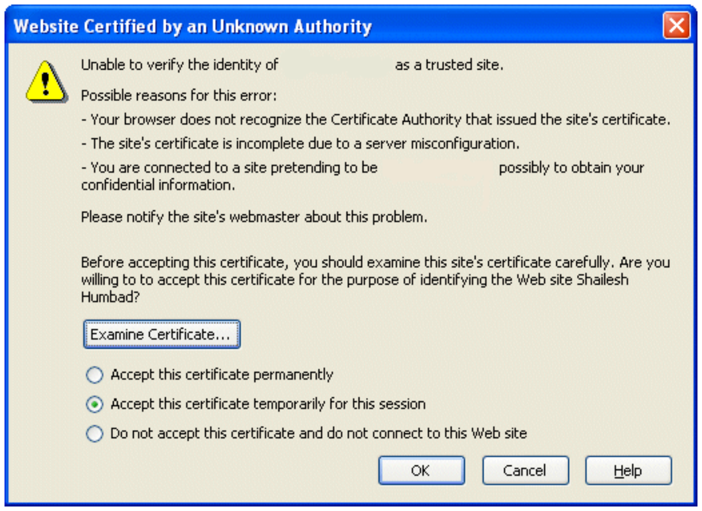


Figure 9: Positive feedback to the user by changing URL bar color and lock icons


Website Certified by an Unknown Authority

Unable to verify the identity of as a trusted site.

Possible reasons for this error:

- Your browser does not recognize the Certificate Authority that issued the site's certificate.
- The site's certificate is incomplete due to a server misconfiguration.
- You are connected to a site pretending to be possibly to obtain your confidential information.

Please notify the site's webmaster about this problem.

Before accepting this certificate, you should examine this site's certificate carefully. Are you willing to to accept this certificate for the purpose of identifying the Web site Shailesh Humbad?

Examine Certificate...
Accept this certificate permanently
Accept this certificate temporarily for this session
Do not accept this certificate and do not connect to this Web site
OK
Cancel
Help

Figure 10: Warning dialogs that are routinely ignored by most online users.

through the dialog as shown in Figure 10, more ominous looking dialogs like the ones shown in Figure 11 are generated when an invalid certificate is found in the certificate chain. In addition, newer browsers control the proliferation of lock icons, use plain colors for the URL bar, and employ normal favicons.

This shift in HCI with respect to online security has been referred by Marlinspike as going from giving the user *positive feedback* to *negative feedback*. His recent attack is based on the observation that any attack that triggers negative feedback is bound to fail, but the absence positive feedback during the attack is not so bad.

The attack proceeds as follows:

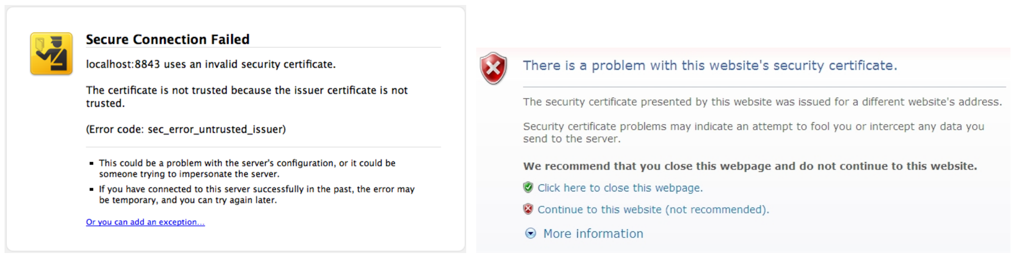
1. Intercept all web (HTTP) traffic and replace

   (a) <a href=`https://...`> by <a href=`http://...`>

   (b) Location: href=`https://...` by Location: href=`http://...`

   And keep a map of all replacements.

2. If there is an HTTP request from the client for a resource for which there was



Secure Connection Failed

localhost:8843 uses an invalid security certificate.

The certificate is not trusted because the issuer certificate is not trusted.

(Error code: sec_error_untrusted_issuer)

- This could be a problem with the server's configuration, or it could be someone trying to impersonate the server.
- If you have connected to this server successfully in the past, the error may be temporary, and you can try again later.

Or you can add an exception...

There is a problem with this website's security certificate.

The security certificate presented by this website was issued for a different website's address.

Security certificate problems may indicate an attempt to fool you or intercept any data you send to the server.

We recommend that you close this webpage and do not continue to this website.

Click here to close this webpage.

Continue to this website (not recommended).

More information

Figure 11: Negative Feedback

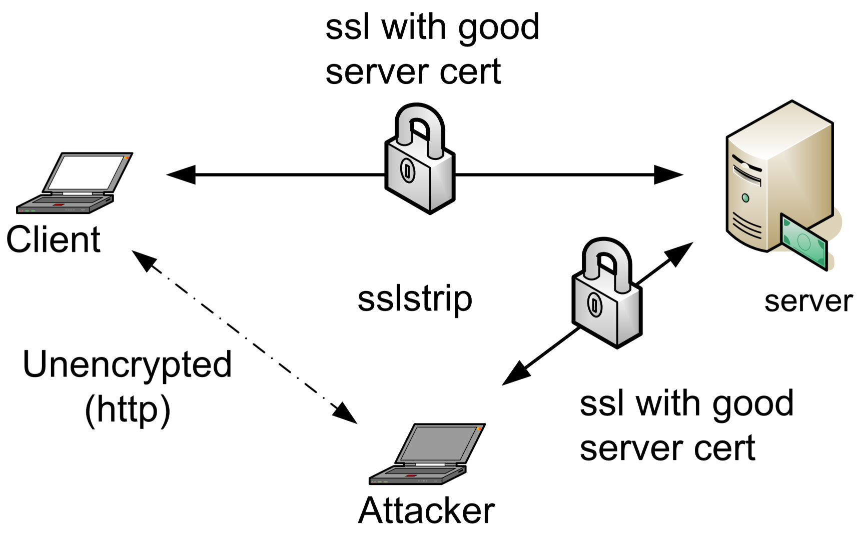


Figure 12: Hijacking secure online transactions

replacement in the previous step, issue an HTTPS connection to the server for the same resource, and

3. Relay response the server using HTTP to the client.

The key difference between this MITM attack and the attack described in Section 3.3 is that in the previous attack, the attacker uses HTTPS to connect to both the client and the server. By comparison, in this new MITM attack, the attacker only communicates with the server in encrypted mode. From the point of view of the server, this would appear like a normal secure online transaction. Compare Figures 8 and 12.

On the client side, there are no tell-tale signs of a breach since the attack suppresses nasty dialogs from popping up. This accomplishes the goal of not triggering any negative feedback. To complete the attack, Marlinspike adds some positive feedback. This is done by adding a lock favicon in the URL bar. That is, whenever a favicon request is noticed for a URL that is in the map, a lock favicon is returned. The only difference a security savvy user would notice is the absence of a lock icon in the status bar and `http` instead of `https` in the address bar.

Cached pages can pose a problem as they do not give the attacker a chance to replace https with http. The details on how to deal with this and other technical problems arising from sessions, cookies, and others can be found in Marlinspike, 2009.

The results from this experiment are remarkable. The security of over a hundred email accounts, a few credit card numbers, and a few hundred secure logins was breached in a matter of a single *24-hour* period. Another surprising aspect of this test was that *not a single user* attempting to initiate a secure transaction aborted it because the user became suspicious.

Marlinspike also showed how to extend the homograph attack (attack that attempts to deceive remote users about what server they are communicating with, by taking advantage of the fact that many different characters have nearly indistinguishable glyphs) to mount MITM against SSL. We omit the details of this attack and the technical problems posed by cached pages. The interested reader is referred to Marlinspike (2009).

3.5 Possible Defenses against MITM

We conclude this section by presenting some effective measures an online user can take to defend against MITM attacks. First and foremost is to educate oneself to look for signs of a breach. It is also important to understand the meaning of different warning dialogs presented by the browser.

If a web server offers its services only over HTTPS (on TCP port 443) and routinely redirects all HTTP (port 80) requests to the secure port 443, then sessions can still be hijacked. As long as HTTPS depends on HTTP, it is vulnerable because HTTP is not secure. Why not just turn off port 80? Unfortunately this would cause many `Server Not Found` errors for the users and it would not be good for business. One work around is to have the user type in `https://...` in the address bar. Alternately, the user could bookmark the secure site and issue an HTTPS request by selecting the bookmark. It is tempting to think that if browsers always try to connect over port 443 first, and only connect only to port 80 as a last resort, we can avoid the MITM attacks mentioned here. Unfortunately, the attacker can simply drop the requests to connect to port 443 and make the browsers think that the web server does not offer HTTPS. While this defense might not help in all cases, by including into browsers a select set of sites for which service over HTTPS is known to exist, one can reduce the risk of MITM attacks. The only long term solution is to secure everything, that is run only HTTPS.

Another measure that could improve security, that is not currently popular, is the verification of client certificates. By having servers verify the identity of the client, one can achieve better security. But, this requires significant changes to the existing PKI and is not immediately applicable.

3.6 Man in the Browser

Structurally similar to a MITM attack, a *man in the browser* attack is a form of a trojan that can do the following: (Philipp Gühring, 2006)

- Modify the appearance of a website before the user sees it.
- Modify the data entered by the user before it is sent to the server.
- Modify the responses of the server so that they are what the user expects.

All of the above can be done without the user having any visible effect for the user to detect.

These trojans are installed by some means, such as a drive-by download, or the user running untrustworthy programs. They then attack the browsers installed on the system by changing their configuration so the trojan is loaded when the browser starts. Browsers have this capability, known as Browser Helper Objects or Extensions depending on the browser, to allow third parties to extend and improve the browser. The trojan exploits this by creating an extension that watches the user's behavior. When a target site is visited, the trojan commences an attack. Anything entered by the user or sent by a server is vulnerable to capture. This could include login credentials, financial information, or even identity credentials sufficient for identify theft.

Philipp Gühring, 2006, provides several means to prevent this type of attack. Unfortunately all of the proposed solutions have serious drawbacks. The best option is to prevent the trojan installation in the first place instead of modifying the browser. This is further discussed in section 6.

4 Rootkits

A *rootkit* is a set of programs that allows a consistent, undetectable presence on a computer (Hoglund & Butler, 2006). They key part of the definition is "undetectable." Rootkits are used to hide specific files, folders, processes, and network connections from the end user. They are not inherently "bad." For example, corporations could use a rootkit to monitor and enforce computer use regulations, or law enforcement could use a rootkit to collect evidence. However, as with most tools, they can also be put to nefarious purposes. For instance, an attacker could use a rootkit to hide their intrusion into a computer. This makes it much more difficult to determine if a computer has been compromised.

It is also important to understand what rootkits are not. A rootkit is not an exploit. They are often used together, and occasionally a rootkit will use an exploit to operate, but they are not an essential piece. Rootkits are not viruses. Viruses self-propagate which makes it more likely to be discovered. However, viruses do often use techniques of rootkits to slow down their detection.

Modern operating systems are object oriented. To perform a task, typically functions from several objects are invoked, creating a long call chain. To maintain their stealth, rootkits exploit this behavior. They interject themselves into the chain and change the answers as they pass by. This principle of modification, known as *hooking*, is at the core of most rootkits.

Binary, a readily executable program, can come in several different formats such as the Common Object File Format(COFF), the Executable and Linking Format (ELF), a.out, and the Portable Executable (PE). The format used is generally dictated by the operating system and in the case of Microsoft Windows, PE and COFF formats are used (Microsoft Corporation, 2008). All of these formats are well documented. This allows an attacker to *patch* a program so that its does what the attacker wants. This is the most common form of modification that a rootkit uses.

Source code modification is another form of modification used by rootkits. An attacker can insert malicious code directly into a program that they author. This could take the form of an overt backdoor, or a subtle bug that the only attacker knows how to exploit.

Finally, a rootkit could use an entirely different means of modification. Operating systems are very complex leaving a multitude of places for rootkits to hide.

Despite the large varieties of potential techniques that a rootkit could use, they can still be categorized. One means of bifurcating these techniques is to classify them as either *kernel-mode* or *user-mode*. In some contexts, particularly for Microsoft Windows, these are also often known as *Ring 0* and *Ring 3*, respectively. Roughly speaking, kernel-mode code has free reign over a computer's memory and hardware, whereas user-mode code has restrictions placed upon it. It is more difficult to write kernel-mode code because the variances between computers are fully exposed. Modern rootkits will often be composed of pieces that run in kernel-mode and in user-mode.

4.1 Mebroot

During the boot process of a computer, the first physical sector of the hard drive, known as the Master Boot Record (MBR), is read and executed. It is responsible for starting the operating system. In the days when DOS was the prevalent operating system, MBR viruses were common (F-Secure Corporation, 2008).

At the Black Hat USA 2005 security conference, researchers Derek Soeder and Ryan

Permeh of eEye Digital Security presented a tool called BootRoot (eEye Digital Security, 2005). This tool is a proof-of-concept of the exploit they found in the Windows NT family of operating systems whereby a user-mode program could alter the MBR.

When attempting to detect a rootkit, one very helpful thing is the order of execution. If a detection program runs prior to the rootkit, it is much easier to detect the rootkit because the detection program can detect that something has changed. In the converse, any modifications that a rootkit would employ are already completed prior to the detection program starting. Since the MBR is the first thing executed when a computer boots, a rootkit installed into the MBR gives a clear advantage.

Mebroot is a rootkit that installs itself into the MBR. It has been extensively analyzed by GMER Team, 2008, and Kleissner, 2008. It infects a machine with the following operations:

- Write a new kernel-mode driver to the last sectors of the hard drive.
- Copy sector 0 (MBR) of the hard drive to sector 62, and then overwrite sector 0 with a new MBR loader.
- Write new kernel loader to sector 61 of the hard drive.

The system is then forced to restart, which loads the rootkit.

Much of the MBR loader code is copied from BootRoot (GMER Team, 2008). It loads a kernel driver which patches two functions of disk.sys. In particular, it prevents the kernel from overwriting the MBR, and when the MBR is read, the contents of sector 62, the original MBR, are returned. Finally the kernel-mode driver is loaded. This kernel-mode driver includes a networking layer and can bypass local software firewalls. It contacts a server to ensure that it is running the latest version of the rootkit.

What makes Mebroot particularly stealthy is how it is stored on the hard drive. Typically, a rootkit would patch file reading functions to disguise the existence of its files. This patching is potentially a means for the rootkit to be detected. Mebroot writes itself directly to the sectors of the hard drive, bypassing the normal filesystem of the hard drive. This makes the files invisible without having to patch anything.

4.2 Defenses against Rootkits

There are two main ways to defend against rootkits: prevention and detection. As facile as it seems, preventing a rootkit from installing itself in the first place is the best way to prevent rootkits from infecting a machine. Often rootkits are only a part of an attack on a computer. They often come along with botnet clients, and other malicious software. General prevention strategies are discussed further in section 6.

Rootkit detection is very similar to virus detection. There are two main approaches: look for known rootkits, or look for suspicious behavior.

To look for a known rootkit, first one needs to know what a specific rootkit "looks like." This is referred to as its *signature*. The signature could be detected in multiple ways. A file containing the whole rootkit, or a portion thereof, can be scanned before it is protected by the rootkit. The memory of the computer can be similarly searched for fragments of a rootkit. The downside to the approach of looking for known signatures is it requires the rootkit to be known.

Detecting suspicious behavior is more difficult to do. There is the potential for both false negatives and false positives. Many forms of binary patching can be detected by looking for code that is out of place. Detection software can also patch itself into the functions that a rootkit would use to hide itself. The downside to this approach is it requires the rootkit to use a known method of attack, and the detection software checks for all known methods of attack. If the detection software misses a single method, a rootkit could pass by unnoticed.

Recall that one of the behaviors that a rootkit exploits is the idiosyncratic methods by which programs work. Rootkits interject themselves into these methods to disguise their presence. This is also a weakness. Detection software can use two different approaches to answer the same question. If the answers differ, then there is likely a rootkit of some form involved. Rootkits defend against this by patching multiple functions that could be used to detect their presence. This creates a cat and mouse style struggle between rootkits and detection software.

These techniques and more are discussed further by Hoglund & Butler, 2006.

5 Botnets

A *botnet* is a group of individual computers, *bots*, under the control of a *bot herder*, also known as a *bot master*. As a practical matter, all of the bots, also known as *zombies*, are under complete control of the bot herder. A computer becomes a part of a botnet by installing a botnet client. They can be installed by techniques such as drive-by-downloads, a remote exploit on the user's system, or tricking the user into installing them.

After a botnet client is installed, the infected computer will contact a *command and control* (c&c) server. This c&c server is how the herder controls the bots. They can issue commands to tell the bots to do things such as:

- send spam
- be part of a denial of service attack

- scan a network for more computers to infect
- send all the keystrokes from the local computers including things like passwords, bank accounts, and credit card numbers
- install additional malware on the computer

A bot can even be set up to work as a c&c server for a portion of the botnet. This makes it even harder for the bot herder to be found.

According to research by Symantec's MessageLabs, 83.2% of all spam sent in June 2009 was directly sent by botnets. Some smaller botnets direct their bots to use webmail providers to send spam, which is not accounted for in the 83.2%, making it an underestimation (MessageLabs, 2009).

5.1 Attacker's Motivation

As with much computer crime, a primary motivator is profit. There are many ways a bot herder can directly monetize their botnet. A spammer could rent out the botnet. Online businesses can be extorted under threat of a denial of service attack. The captured keystrokes on the individual bots can contain things like credit card numbers, or bank account information which can be sold on the black market. Software can be installed on the bots netting the herder a commission. The possibilities are limited only by the imagination of the herder.

However, there are also non-profit related reasons why an attacker would want a botnet. Suppose that a herder wishes to break into a system. A botnet can help them in a couple ways. One obvious goal the herder will have is to minimize any evidence that ties them to the break in. If the attack is routed through the bots, the herder can force all forensic trails to end at the bots. Further, the attack can be spread across multiple bots which makes it harder for the victim to detect that they are under attack. It is a hard enough problem to detect the precursors of an attack from a single source since often the indicators individually are benign. When the precursors come from multiple sources, the problem is even harder.

Another non-profit motive is to defeat various forms of rate-limiting. Rate-limiting is used to slow down an attacker breaking into a system. For example, when logging onto a computer, the system may force a small delay between password entry attempts. There can also be a lock out whereby after several incorrect passwords, the system prevents access for some duration of time, or even permanently. With these artificial limits, the victim system can prevent the attacker from simply using a program to try many potential passwords at a very fast rate.

Often rate-limiting is implemented on a per computer basis. That is, each bot has their own separate allotted rate. Collectively, a botnet would have a very high rate

Data Type	Quantity
Mailbox account	54,090
Email	1,258,862
Form data	11,966,532
HTTP account	411,039
FTP account	12,307
POP account	415,206
SMTP account	100,472
Windows password	1,235,122

Figure 13: Data types captured from the Torpig botnet. (Adapted from Stone-Gross et al., 2009)

which allows the herder to perform the attacks the rate-limiting is designed to stop.

5.2 Torpig

The Torpig botnet, also known as Sinowal, was taken over for a period of ten days by researchers from the University of California, Santa Barbara (Stone-Gross et al., 2009). The botnet client operated by generating an expected location for the c&c server. The researchers exploited this by analyzing the algorithm and predicting where it will be next and placing their own c&c server there. Due to ethical considerations, the researchers made their c&c server totally passive. It never instructed the bots to do anything, and simply recorded all the data is was sent by the bots. Torpig utilizes the Mebroot rootkit to hide its existence on the infected computers.

Over the course of ten days, they recorded over 70GB of data. Their best estimate is that 180 thousand distinct infections contacted their c&c server. The botnet client was configured to send lots of data to the c&c, summarized in Figure 13

Torpig uses a man in the browser attack, described in section 3.6, on many financial institutions. It also scours the saved password cache of browsers, and records what passwords are entered by a victim. Derived from this captured data, the researchers found 8,310 accounts at 410 financial institutions. Using a heuristic validation, the researchers found 1,660 distinct credit cards. For 2008, Symantec estimated that credit cards can sell on the black market for $0.10 USD to $25 USD and bank accounts $10 USD to $1,000 USD (Symantec Corporation, 2009). Using these estimates, the researchers estimated that the Torpig controllers could have brought in between $83 thousand USD to $8.3 million USD from the captured financial data alone. The potential income for the Torpig controllers is much larger from the other means of monetization discussed in the previous section.

6 Best practices to secure information resources

Both physical security and computer security are fundamentally about the allocation of finite resources to maximize risk mitigation. Even if there were infinite resources, impenetrable security can not exist. The problem is that an attacker needs to find only one weakness to exploit. Preventing all attacks requires one to reinforce every potential weakness. To further complicate this, a reasonable maxim of security is that the more complicated the system, the harder it is to secure.

Since perfect security is untenable, it is best to focus on elements that can be controlled. User education and applying security patches are the best active tactics. User education is the first and best line of defense. Many of the attacks described in this paper rely upon the user overlooking small details. Understanding what a valid SSL certificate looks like, and checking validity can prevent many forms of MITM attacks. As was shown with man in the browser attacks and the potential of rootkits a vigilant user is not always enough. Keeping software up to date and security patches applied can help with other forms of attacks. Drive by downloads, man in the browser, and botnets are often installed by exploiting a bug in the victims computer. Security patches will lessen the number of known vulnerable exploits on a computer.

Additional items that can help include virus scanners, malware scanners, rootkit scanners, and firewalls. Each one of these items has its own limitations, such as only finding known malicious software or detecting known behavior. But each provide an additional layer of defense. The more layers, the harder it is for an attacker to succeed. This principle of adding many layers is known as *defense in depth*.

Suppose that an attacker does succeed in planting malicious software onto a computer. What can be done? In general, it takes an expert to reliably, fully remove malicious software. This can often require rebuilding the computer. At this point backups are invaluable. Frequently the hard drive cannot be trusted because some forms of malicious software infect every potential file on a hard drive. Backups also need to be tested. A non-working backup is as good as no backup at all.

If an attacker succeeds only in simple MITM attack, such as ARP spoofing or a fake SSL certificate, the only harm is the loss of information. This of course could be a very costly loss, but it does not necessitate the reduilding of a victimized computer.

References

Anderson, R. (2008). *Security Engineering* (Second ed.). Indianapolis, IN, USA: John Wiley & Sons Inc.

Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., & Polk, W. (2008, May). *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile* (No. 5280). RFC 5280 (Proposed Standard). IETF.

Dierks, T., & Rescorla, E. (2008, August). *The Transport Layer Security (TLS) Protocol Version 1.2* (No. 5246). RFC 5246 (Proposed Standard). IETF.

Diffie, W., & Hellman, M. E. (1976). New Directions in Cryptography. *IEEE Transactions on Information Theory*, *IT-22*(6), 644–654.

eEye Digital Security. (2005). *BootRoot.* Retrieved July 10, 2009 from `http://research.eeye.com/html/tools/RT20060801-7.html`.

El Gamal, T. (1985). A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of crypto 84 on advances in cryptology* (pp. 10–18). New York, New York, USA: Springer-Verlag New York, Inc.

F-Secure Corporation. (2008). *MBR Rootkit, A New Breed of Malware.* Retrieved July 10, 2009 from `http://www.f-secure.com/weblog/archives/00001393.html`.

Ferguson, N., & Schneier, B. (2003). *Practical cryptography.* New York, New York, USA: John Wiley & Sons Inc.

GMER Team. (2008). *Stealth MBR rootkit.* Retrieved July 10, 2009 from `http://www2.gmer.net/mbr/`.

Hellman, M. E. (2002). An overview of public key cryptography. *IEEE Communications Magazine*, *May 2002*(50th Anniversary Commemorative Issue), 42–49.

Hoglund, G., & Butler, J. (2006). *Rootkits: Subverting the Windows Kernel.* Addison-Wesley.

IBM. (2009). *Certificate chain verification.* Retrieved July 10, 2009, from `http://publib.boulder.ibm.com/infocenter/tpfhelp/current/index.jsp?topic=/com.ibm.ztpf-ztpfdf.doc_put.cur/gtps5/s5vctch.html`.

Kleissner, P. (2008). *Analysis of sinowal.* Retrieved July 10, 2009 from `http://web17.webbpro.de/index.php?page=analysis-of-sinowal`.

Marlinspike, M. (2009, February 16–19). *New Techniques for Defeating SSL/TLS.* Presented at Black Hat DC Briefings 2009.

Menezes, A., van Oorschot, P., & Vanstone, S. (1997). *Handbook of Applied Cryptography.* New York, New York, USA: CRC Press LLC.

MessageLabs. (2009). *MessageLabs Intelligence: Q2/June 2009.* Retrieved July 10, 2009 from `http://www.messagelabs.com/mlireport/MLIReport_2009.06_June_FINAL.pdf`.

Microsoft Corporation. (2008). *Microsoft Portable Executable and Common Object File Format Specification.* Retrieved July 10, 2009 from `http://www.microsoft.com/whdc/system/platform/firmware/PECOFF.mspx`.

National Institute of Standards and Technology. (1999, October 25). *FIPS PUB 46-3:*

Data Encryption Standard (DES). Retrieved July 10, 2009 from `http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf`.

National Institute of Standards and Technology. (2001, November 26). *FIPS PUB 197: Advanced Encryption Standard (AES).* Retrieved July 10, 2009 from `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.

National Institute of Standards and Technology. (2009, June). *FIPS PUB 186-3: Digital Signature Standard (DSS).* Retrieved July 10, 2009 from `http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf`.

Philipp Gühring. (2006). *Concepts against Man-in-the-Browser Attacks.* Retrieved July 10, 2009 from `http://www2.futureware.at/svn/sourcerer/CAcert/SecureClient.pdf`.

Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *21*(2), 120–126.

Stinson, D. (2005). *Cryptography: Theory and Practice* (Third ed.). New York, New York, USA: CRC Press LLC.

Stone-Gross, B., Cova, M., Cavallaro, L., Gilbert, B., Szydlowski, M., Kemmerer, R., et al. (2009). *Your botnet is my botnet: Analysis of a botnet takeover* (UCSB Technical Report). UCSB. Retrieved July 10, 2009 from `http://www.cs.ucsb.edu/%7Eseclab/projects/torpig/torpig.pdf`.

Symantec Corporation. (2009). *Symantec Global Internet Security Threat Report: Volume XIV Trends for 2008.*

Tews, E., Weinmann, R.-P., & Pyshkin, A. (2007, April). *Breaking 104 bit wep in less than 60 seconds.* Cryptology ePrint Archive, Report 2007/120.

Wagner, R. (2001). *Address resolution protocol spoofing and man-in-the-middle attacks.* Retrieved July 10, 2009 from `http://www.sans.org/rr/whitepapers/threats/474.php`. SANS Institute.

Walker, M. H. (2005). *Drive-by downloads: Stealthy downloads and Internet Explorer's new defense against them.* Retrived July 10, 2009 from `http://www.microsoft.com/windows/ie/community/columns/driveby.mspx`.