CONCERNING ENTERPRISE NETWORK VULNERABILITY TO HTTP TUNNELLING

C. Daicos and G.S. Knight
Dept of Electrical and Computer Engineering, Royal Military College of Canada

Abstract: It has been understood for some time that arbitrary data, including the communications associated with malicious backdoors and Trojan horses, can be tunnelled by subverting the HTTP protocol. Although there are a number of demonstration programs openly available, the risks associated with this vulnerability have not been characterised in the literature. This research investigates the nature of the vulnerability and the efficacy of contemporary network defence strategies such as firewall technology, intrusion detection systems, HTTP caching and proxying, and network address translation. All of these techniques are quite easily circumvented by HTTP tunnelling strategies. This vulnerability is serious for most enterprise environments today. The use of some Internet services is considered to be a requirement for business operations in many organisations. Even with very strict firewall rule sets and layered defence architectures, legitimate web traffic originating from within the protected network is often allowed. Web traffic also forms a large portion of the traffic crossing network boundaries, which makes the HTTP protocol an attractive target for subversion. This research explores techniques that may be used to hide malicious traffic in what seems to be legitimate HTTP traffic originating from within the protected network. The covert channel provides external control of a computer on the protected network from a machine anywhere on the Internet. The techniques explored by this project are used in parallel research projects to detect such malicious tunnel traffic and validate new intrusion detection technology.

1. INTRODUCTION

The purpose of this research is to investigate and characterise the security risk presented by HTTP tunnels in an enterprise network environment. HTTP tunnels are an important class of network vulnerability for which there has been little formal treatment in the literature. This is the case even though the basic mechanisms of HTTP tunnelling have been understood for some time. Open-source demonstration programs for the UNIX environment are available [HTTPT, RWWW] and in the Windows environment there are commercial ventures [HTC, TotalRC] that provide software and gateway services to individuals who wish to circumvent the firewall protecting their host network. It is common for users of peer-to-peer networking applications such as Morpheus and Gnutella to use HTTP tunnels to access the Internet after their network administrator has blocked access at the firewall to the normal ports these programs communicate on.

In many enterprise environments today access to the World Wide Web is considered to be a requirement for the network's support to enterprise operations. This is the case for many business organisations, and includes commercial, government and military networks. The messages sent to request a specific web page, and the web page returned, can contain covert information. This channel for covert information can be used to signal commands to a backdoor/Trojan horse program inside a protected network. This can give the outside attacker control of an internal enterprise computer. The covert channel can be used to modify, delete, or steal sensitive information from the internal enterprise computer. The well-known backdoor Back Orifice 2000 [BO2K] has a communication mode that uses an HTTP tunnel to penetrate the target network's firewall.

HTTP tunnelling is a current exploitable vulnerability of enterprise networks and precludes organisations from providing Internet access for many sensitive networks. The current intrusion detection sensor technology and firewalls provide little support for preventing or detecting such covert channels.

This paper describes an investigation of covert HTTP tunnels that included the development of a proof-of-concept application that was used to penetrate the security perimeter of several networks. The work also tries to anticipate the next generation of covert HTTP tunnels that will use obscuring techniques to hide the tunnel traffic more effectively in the background noise of the legitimate HTTP traffic [RFC2068] being carried on the network. HTTP tunnels are an important class of network vulnerability and it is expected that the risk associated with such tunnels will be a major driver of research in computer system configuration validation, and intrusion detection technology.

The techniques and proof-of-concept application resulting from this research are being used as test generators in parallel research projects to detect malicious tunnel traffic and validate new intrusion detection technology.

Section 2 of the paper provides some background on the simple use of HTTP tunnels to penetrate a network firewall. Section 3 demonstrates how a tunnel can penetrate a network with a more robust security architecture. Ways in which tunnels can be made more covert and harder to detect are presented in section 4. Section 5 provides a discussion about research strategies that might be useful in defeating covert HTTP tunnels, and concluding remarks are provided in the last section of the paper.

2. HTTP TUNNELING

2.1 Subverting the HTTP Protocol

Consider a location on the Internet described by the following URL:

```
http://anysite.com:80/
```

A client request method is a method issued to an HTTP server by an HTTP client that declares its intentions. The client methods include GET (retrieve a page), POST (client provides content to a server), PUT, CONNECT, etc. [Won00]. An example of a message that a web browser might send to the machine anysite.com when asked to retrieve a web page from this resource is:

```
GET / HTTP/1.1
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: zip
User-Agent: Mozilla/4.0
Host: anysite.com
Connection: Keep-Alive
```

This is the Request Header. The first line of this request "GET / HTTP/1.1" requests a document at / from the server. The "1.1" is the version of the HTTP protocol in use by the browser. The name of the document requested in a GET message (just a '/' in this example) is a data field that is totally controlled by the sender of the message. It can be almost anything.

Given a request like the one above, the web server looks for the resource (e.g. web page) associated with requested name and returns it to the sender of the message, preceding it with some header information in its response. The resource associated with the URL depends on how the server is implemented. It could be a static file or it could be dynamically generated. In this case, the server might return:

```
HTTP/1.1 200 OK
Date: Mon, 06 Nov 2001 20:21:35 GMT
Server: Apache/1.3.6 (Unix)
Last-Modified: Fri, 03 Nov 2001 12:00:09 GMT
Accept-Ranges: bytes
```

```
Content-length: 1000
Connection: Close
Content-Type: text/html

<html> …
```

The above text is called the response header and the part with the HTML code (two carriage returns below the header) is the body.

To create the covert HTTP tunnel, information can be encoded in the name strings of the client's GET messages and in the bodies of the responses from the server. The concept behind the mechanism described below is quite general; where specific detail is provided it is for the case of the application developed by the researchers.

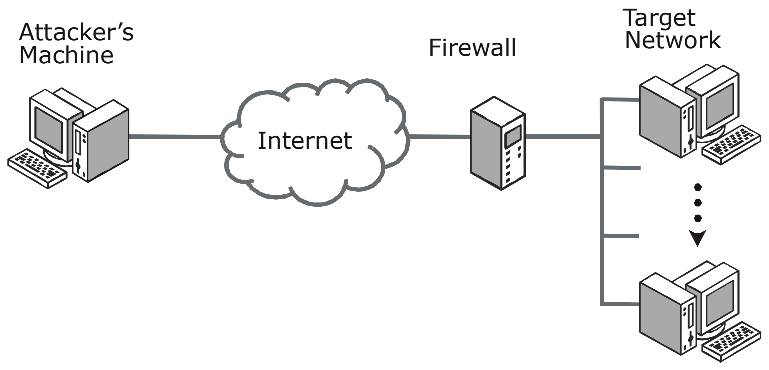

Figure 1 – A Simple Network Architecture

Specifically consider the case described by Figure 1, which can be used to describe an *HTTP reverse tunnel*. To illustrate the capabilities of the reverse tunnel assume that a machine on the target network is running a malicious program (the client) that will communicate with an attacker (the server) somewhere on the Internet, in violation of the security policy of the network. How this program first establishes itself on the machine is not within the scope of the discussion. Network computers can become compromised by viruses, removable media, during the movement of laptop computers between trusted and untrusted environments, or by legitimate users either wittingly or unwittingly installing unauthorised programs. Also assume that firewall on target network runs a highly restrictive rule set and that the only outbound communication allowed is HTTP GET requests. The only inbound communication allowed is the response messages from web servers.

The client code behaves like a very simple web browser. Once executed on the target machine, it calls "home" to the server program by issuing a simple GET request whose header might look like this:

```
GET /some_sub_dir/index.html HTTP/1.1
User-Agent: Mozilla/4.0
Host: 123.109.117.215
Connection: Keep-Alive
```

The IP specified in the "Host" field of the request header is the IP of the attacker's computer. The sub-directory specified as the requested resource in the GET field may not be a directory listing at all, but acts as a code-word to the server program on the outside that a covert client is calling home. The server program on the attacker's computer, which is listening on port 80 will receive the request from the target machine and return a confirmation embedded in the body of an innocuous-looking web page; just as if it was serving up a page about news, sports, or weather:

```
HTTP/1.1 200 OK
Date: Mon, 06 Nov 2001 20:21:35 GMT
Server: Apache/1.3.6 (Unix)
Last-Modified: Fri, 03 Nov 2001 12:00:09 GMT
Accept-Ranges: bytes
Content-length: 1000
Connection: Close
Content-Type: text/html

<html>
<body>
normal-looking HTML here ...
IDENTIFIER TAG =Confirmed
...
</body></html>
```

Once the target machine on the inside of the firewall has received the acknowledgement, it issues a new GET request every few seconds. The attacker can respond to any one of these requests with a web page containing an encoded command to be executed on the client machine. If no data is returned from the attacker's machine, the Trojan client simply times out and retries after some set period of time.

When it retrieves a page with a system command embedded in the body of the HTML (indexed by some string or keyword) like this:

```
<html>
<body>
normal-looking HTML here ...
SYS_COMMAND=dir c:\secret_files\*.doc
...
</body></html>
```

The target machine executes that command on the target system, parses the output from the standard output into a series of strings that are then embedded in a series of GET requests. These GET requests are received on the HTTP port of the attacking machine, and the strings from each request are reassembled to display the output from the results of the system command executed on the compromised machine.

2.2 Matching a Legitimate Protocol

The power and stealth of this exploit lies in the fact that the connection is *one way*. Unlike some existing exploits that establish a *two-way* connection on obscure or redirected ports, or attempt to use port 80 to connect directly to a telnet or ssh server, commands in this reverse tunnel scheme are only piped to the target machine *when it asks for them*.

The reverse tunnel described here is different from a popular tool known as "httptunnel" [HTTPT]. Httptunnel tunnels through a firewall from the inside out like a reverse tunnel, but it has no automatic client initiation. It requires modification to redirect data at the endpoints, and it uses client methods like POST and PUT to do file transfers, which are more easily recognisable in the packet stream.

With all signalling initiated by the client, and with all the details of the transfer buried or even encrypted inside the body of an innocent looking web page, the stream of data becomes hard to distinguish from the normal traffic produced by real web browsers on the network.

3. HTTP TUNNELING AND NETWORK DEFENCES

Now consider the more robust security architecture of the system illustrated in figure 2. The firewall and its rule set are similar to the last example. In this case however, the firewall is also performing network address translation (NAT). The entire target network is running in a private address space so that an individual machine address is not routable from a remote Internet machine. The only machine on the target network that is visible to the Internet is the firewall, and it does not accept any unsolicited inbound traffic (only responses to HTTP GET messages). The traffic flowing by the firewall is being monitored by an intrusion detection system (IDS). The IDS will alarm the network administrator if it recognises the signature of an intruder in the traffic in-bound to, or out-bound from, the network. A caching HTTP proxy server vets all HTTP traffic on this network. In this configuration no network web browser communicates directly with a web server on the Internet. Web browsers may only communicate with the proxy server. The proxy server opens a separate new connection with the outside Internet web server on behalf of the internal web browser and passes requested web pages back to the browser. The proxy matches the return traffic with the requests and ensures that only pages that were actually requested are passed back through to the client. The use of the HTTP proxy is password protected.

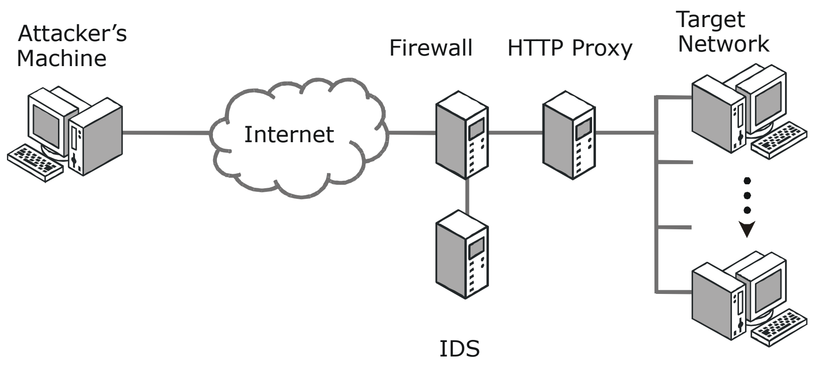

Figure 2 – A More Robust Network Security Architecture

The network in this example might be typical of many contemporary enterprise network environments. However, the security features of this network do not substantially change the operation and effectiveness of the HTTP reverse tunnel. Recall, the tunnel information is carried in the name of the resource requested and in the content of the web page returned. These fields are not changed in transit through the security perimeter. In the proof-of-concept tunnel application some additional features were employed to provide a password when required and additional header lines were required to route the request through the HTTP proxy. These modifications were minor and predictable. The information required (e.g. proxy's address, password) is available on the machine that has been compromised by the tunnel client software. It is also possible that the tunnel client software can subvert the legitimate web browser on the target machine. This would make it very difficult to differentiate legitimate user initiated web traffic and traffic associated with the covert tunnel.

Although several well-known HTTP tunnelling programs can be detected by a signature-based IDS, this is not the case in general. The IDS usually detects a keyword string that is unique to the control of communication by the tunnel's information coding scheme. This kind of signature-based detection is not useful in the general case were the attacker is using a unique application, or is obscuring or encrypting the communication. The IDS/firewall can also detect/block traffic to well-known HTTP tunnel gateways. But, an attacker is not obliged to operate from a well-known HTTP tunnel gateway. The attacking machine could be at any address on the Internet.

The proof-of-concept tunnel application was successfully trialed against several test networks and live enterprise networks that included the security features described above. The application was designed to be deployed on Windows 2000 machines. The tunnel provides a command shell and basic ftp-like services. The client is written in C++ and the server is a GUI-based application written in Visual Basic.

4. OBSCURING HTTP TUNNELS

The majority of traffic crossing the firewall in many large enterprise networks is HTTP traffic. Sometimes web traffic is the only traffic of any appreciable volume to cross the network perimeter. This makes HTTP an attractive target for abuse. The HTTP tunnel traffic is hidden in the large volume of background noise provided by the legitimate HTTP traffic being carried on the network. The authors anticipate a new generation of covert HTTP tunnels that will use obscuring techniques to hide the tunnel traffic more effectively.

It is expected that creating and detecting tunnels will be an evolution of measure and counter-measure. The tunnel builder will strive to provide tunnel communications that are better and better matches to legitimate HTTP traffic (measure). The intrusion detection community will identify means to separate normal HTTP traffic from anomalous traffic that may indicate malicious use of the protocol (counter-measure).

The following techniques are measures that can be used to obscure a covert tunnel.

4.1 Non-Friendly Client Access

The attacker would not consider it desirable for any client to be able to pull down the synthetic web pages from the tunnel's server program. Only actual covert tunnel clients should be able to request the "special" web pages that constitute the tunnel. In the event that a system administrator would like to check the content of the web pages that a suspicious computer is requesting, she should not be able to observe any questionable HTML. Cutting off access completely to unfriendly clients is suspicious, so clients that cannot be identified are served a page of daily updated sports scores or something similar. Serving "404 page not found" errors is also a possibility.

4.2 System Header and Operating Camouflage

As stated earlier, the header for a GET request from any given web browser will look something like this:

```
GET / HTTP/1.1
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: zip
User-Agent: Mozilla/4.0
Host: anysite.com
Connection: Keep-Alive
```

If configured properly, an HTTP proxy or firewall can scan these headers and reject them based on a ruleset that disallows unregistered or misconfigured browsers to access the internet. To circumvent this, the tunnel client can scan the infected system for recognisable browsing processes, and compile a header to exactly match the User-Agent, Accept-Language, and most other fields.

4.3 Concurrent Activity

By setting the exploit to run only when another web-client process is active would avert suspicion when analyzing HTTP traffic during off-hours.

4.4 Asynchronous Communication

The response time of early remote-shell prototypes communicating through reverse-HTTP tunnels was hampered by the inability to transmit data the moment it was available. Because communication is always client-initiated, the server side of the covert channel is forced to wait for a subsequent GET from the client before it can send information. This model generates many unnecessary connections, and requires that response time be a function of polling frequency. Asynchronous communication without polling is possible. For example, consider the behaviour of web browsers when attempting to load a large page on a slow connection. When a large number is specified in the `Content-length` header of the page served, the web browser is obligated to wait expectantly for the remainder of the data. Major browsers have a timeout of between two and four minutes. If in that time the server has more data to send, it simply sends it through the already open connection. Conversely, if during this period the user wishes to terminate the connection and reload the page, the browser tears down the connection, establishes a new one, and issues a new GET.

We can mimic the above behaviour to make data transfer on a reverse tunnel asynchronous. This allows for real time interaction with a remote shell. By specifying random large numbers in the `Content-length` header of a response to a GET, a connection can be left open for a two-minute window. If in that time there is another command to execute, it is sent immediately through the open connection. Upon command execution, the client will reset the existing connection, establish a new one, and issue a new GET with the results. In this way, asynchronous communication is established without violating the behaviour expected of a web browser.

4.5 Reverse Tunnel Traffic Encoding

GET strings are becoming longer and longer as companies continually add more functionality to their websites with Common Gateway Interface (CGI) programs, and proprietary html-like formats such as .ASP, .CFM, and others. It is not uncommon to see very long GET strings.

A reverse tunnel can exploit this fully by parsing the information being sent back to the attacker computer into strings formatted with the same conventions observable in the Internet applications. For example, a request string when logging on to a hotmail server might be:

```
http://lc2.law13.hotmail.passport.com/cgi_bin/loginerr?
curmbox=F000000001&a=20d9351fd99bfbdd66c9715defff8069&e
rror=4&sec=no&reauth=&id=2&fs=1&cb=_lang%253dEN&ct=1008
617408&_lang=EN&domain=hotmail%2ecom&utf8=0
```

A request string encoded by a tunnel might be:

```
http://119.124.155.107/p99377/rnad.srf?lc=4105&Pd=6528&
ru=htQr_7982626_1265_shx%3fmsu%3d_673kkd/clo2230005=%3f
```

```
C%nQs%3d1%26msnruend%3d1&tw=1000000&kv=2&cbid=6528&ts=_
5&da=passport.c?%%om&r=20.0248.1&t=pf=9bf5474785dbfb620
f5925b718cbc965
```

It is fair to say that the requests look similar. The first one is from a legitimate Hotmail session, whereas the second contains the first 200 characters of a file named “Draft Earnings Forecast Q3 2002” and is destined for an attacker’s machine. An observer would likely notice nothing special about this traffic, and current firewalls cannot detect the difference. They would allow it to pass unhindered.

The Reverse Tunnel’s server responses of course, are easier to hide. Since the server can embed its response in a pile of HTML, it is nearly impossible to trace.

Keyword searches or body scans on traffic can be eluded by basic encoding and steganographic techniques used to hide the data in transit. For example, GET requests might be base64 encoded, and limited to 512 byte lengths, while server responses are base64 encoded and embedded in HTML, or encoded in some standard image format.

5. DEFEATING HTTP TUNNELS

As has been demonstrated by the research, HTTP tunnelling is a current exploitable vulnerability of enterprise networks. To manage the risks associated with such tunnels, one must either detect the traffic or lock down the internal network such that the tunnel client cannot establish itself. Both alternatives provide open research opportunities.

The current generation of security perimeter defences and IDSs offer little protection from a tunnel once it is established. We believe that HTTP tunnel traffic encoding will make it difficult for signature-based IDSs to mitigate the threat. Anomaly-based intrusion detection technologies may be better suited to this problem. If the anomaly-based malicious use sensor can characterise the legitimate web traffic on the enterprise network, then traffic that differs from this norm can be flagged as malicious and subject to special analysis. This is the motivation for development of the proof-of-concept tunnel application described in this paper. The application will serve as a test bed to generate test and validation traffic to be used in research into new anomaly-based detection technology.

An alternative to monitoring and detecting established tunnels is to control the configuration of the enterprise network tightly enough to ensure that covert code cannot be run on the internal systems. Again, the current generation of operating systems seems to have difficulty offering adequate protection. Integrity checking applications and system configuration validation techniques can be used to detect and prevent the compromise of network computing platforms.

6. CONCLUSION

The Reverse Tunnel is a serious threat to computer security, with obvious applications in computer espionage. The magnitude of this threat is almost wholly derived from the fact that it targets a well-known protocol on a well-known port that is easy to exploit, expensive to fix, and far too useful to terminate. This makes for an exploit whose variations will likely continue to be a threat for a long time.

The HTTP protocol and the network components that implement and secure it have no real way of differentiating between legitimate and offending traffic. Contemporary network defence strategies such as firewall technology, intrusion detection systems, HTTP caching and proxying, and network address translation are not effective in detecting and preventing covert HTTP tunnels. It is expected that creating and detecting tunnels will be an evolution of measure and counter-measure. To manage the risks associated with such tunnels, one must either detect the traffic or lock down the internal network such that the internal tunnel client cannot become established.

REFERENCES

[BO2K] Sourceforge, Back Orifice 2000, http://bo2k.sourceforge.net/

[HTC] HTTP-Tunnel Corporation, http://www.http-tunnel.com

[HTTPT] Httptunnel, http://www.nocrew.org/software/httptunnel.html

[RFC2068] Hypertext Transfer Protocol -- HTTP/1.1, http://www.ietf.org/rfc/rfc2068.txt

[RWWW] Reverse WWW Tunnel Backdoor (rwwwshell), http://www.securiteam.com/tools/5WP08206KU.html

[Smi00] Smith, J.C., Covert Shells, http://rr.sans.org/covertchannels/covert_shells.php, 2000.

[TotalRC] TotalRC, http://www.totalrc.net

[Won00] Wong, Clinton. *HTTP Pocket Reference*. O'Reilly & Associates, Sebastopol, Californian, 2000.