Surf Jacking

"HTTPS will not save you"

By Sandro Gauci, **ENABLE**SECURITY

Date: 10 August 2008

Introduction

In this paper we will describe a security issue that affects major web sites and their customers. Attackers exploiting this vulnerability are able to hijack an HTTP session even when the victim and the attacker's connection is encrypted using SSL or TLS. We will first describe the components that make up this vulnerability for those who do not have in-depth knowledge of HTTP and HTTPS. If you're familiar with the HTTP protocol, feel free to skip to the section "Underlying protocols".

ENABLESECURITY is releasing a tool called surfjack.py which demonstrates the ideas described in this paper. In the section "Proof of Concept" we describe how the tool works and the various options that it supports.

Finally we will describe a simple solution that was successfully applied to two Banks found vulnerable. We shall also discuss why this solution might not be so straightforward to implement in large Single Sign-On services such as Google's network.

Why make use of encryption? 3
The devil is in the details 3
HTTP Cookies 3
HTTP Redirect 3
Underlying protocols 4
What is Surf Jacking? 5
Proof of Concept 5
Prior Art 7
Vulnerable Services 8
Suggestions 9
About EnableSecurity 10
Disclaimer 10
Redistribution 10

Why make use of encryption?

A large percentage of network traffic, even in this day and age, tends to be clear text. According to a one source[1], peer to peer tends to be the most traffic intensive, followed by HTTP. On the other hand, tunneled and encrypted protocols (such as HTTPS) are on the end of these statistics. Most of the network traffic is therefore not encrypted and the contents can be easily understood by anyone able to inspect network traffic between two points. However, when the traffic is likely to contain sensitive information such as credit card details or business plans, it is at least recommended, and many times required, that such traffic is encrypted.

When making use of encryption, many network protocols are tunneled over Transport Layer Security (TLS). Everything from receiving email through IMAP to voice over IP calls using RTP can be tunneled over TLS, to auto-magically turn an insecure protocol into one that supports a good layer of security.

TLS is able to provide network traffic with the following features of interest for the scope of this paper:

- Certificate based authentication which helps protect against man in the middle and replay attacks
- Encryption of traffic to help protect the contents of the traffic against anyone watching

These features is what makes HTTPS (which consists of HTTP over TLS) a reliable and safe mechanism for transport of sensitive information over untrusted networks. Examples of untrusted networks would be free wireless at the café or Internet access from the network point in your hotel room. The Internet itself is considered an untrusted network since there is no guarantee that an attacker is not sitting between client and server, passively monitoring network traffic or performing a man in the middle attack. In such cases, by making use of TLS, a client can ensure that the communication is encrypted and that the server is who it claims to be.

The devil is in the details

Many online services that distribute sensitive information make use of HTTPS. The basics of HTTP are quite straightforward and the reader is expected to be familiar with terms such as GET, POST and HTTP headers. However, the underlying technologies and details are what makes this protocol so flexible and therefore powerful. In this paper we shall be mostly talking about the Cookie support in HTTP and ways that seemingly legitimate servers can redirect HTTP traffic.

HTTP Cookies

The Cookie consists of text data that is stored either in memory or on disk on the web browser. It is associated with a specific host and possibly a path on that host's website. As a basic protection mechanism, web servers can only set cookies on clients which apply their own host names. This is known as the same-origin policy. For example, http://www.microsoft.com/ can only set a cookie for its own domain and cannot set a cookie for http://www.google.com/.

Once a cookie is set on the web browser, it is sent by the web browser with each HTTP request as an HTTP header as seen in the example below.

```
GET /somepage.html HTTP/1.1\r\n
Host: www.example.org\r\n
Cookie: sessionid=someuniquevalue\r\n
Accept: */*\r\n\r\n
```

HTTP Redirect

When an HTTP client or a web browser receives an HTTP response "301 Moved Permanently" from the web server, it will be automatically redirected to another site which is specified in the HTTP response header "Location". Similar behavior can be achieved by making use of HTTP response codes 302, 303 and 307[2]. Apart from these HTTP methods, web browsers can be redirected by Javascript and possibly various other methods.

In the example below, the client sends an HTTP request asking for a page on example.com and is redirected to another page on enablesecurity.com, which displays a “hello” message back to the web browser. The end user does not have to know that his or her browser was redirected to a different web server and HTML document since everything is done automatically by a standard web browser.

```
connection: client -> www.example.org:80
GET /somepage.html HTTP/1.1
Host: www.example.org
Accept: */*

HTTP/1.1 301 Moved Permanently
Location: http://www.enablesecurity.com/
Cache-Control: private
Content-Length: 0



connection: client -> www.enablesecurity.com:80

GET /somepage.html HTTP/1.1
Host: www.enablesecurity.com
Accept: */*



HTTP/1.1 200 OK
Cache-Control: private
Content-Length: 5

hello
```

Underlying protocols

While HTTPS is considered to be a safe protocol for secure connections, there are times when underlying protocols such as HTTP cookies can break paradigms. Cookies have a flag called “secure”, which when enabled will only transmit the cookie to encrypted sites. However many HTTPS sites do not make use of this flag and therefore the cookie is not aware of the security context of the traffic.

The result is that even though the traffic between the server and client is transported over a secure protocol, an attacker sitting in between the victim client and victim web server, can launch a downgrade attack to reveal the session cookie.

What is Surf Jacking?

The following is a scenario of how the attack can take place:

- Victim logs into the secure web service at https://somesecurebank.com/.
- The secure site issues a session cookie as the client logs in.
- While logged in, the victim opens a new browser window and goes to http://www.example.org/
- An attacker sitting on the same network is able to see the clear text traffic to www.example.org.
- The attacker sends back a "301 Moved Permanently" in response to the clear text traffic to www.example.org. The response contains the header “Location: http://somesecurebank.com/”, which makes it appear that www.example.org is sending the web browser to somesecurebank.com. Notice that the URL scheme is HTTP not HTTPS.
- The victim's browser starts a new clear text connection to http://somesecurebank.com and sends an HTTP request containing cookie in the HTTP header in clear text
- The attacker sees this traffic and logs the cookie for later (ab)use.

The attacker can also choose to be more stealthy and take the following extra steps:

- The attacker can then send another "301 Moved Permanently" sending the victim back to the original website http://www.example.org/
- In this case, the victim's browser goes on to render the original destination site without much visual indication that anything happened.
- The attacker sets the session id on his or her browser and gains access to the victim’s session.

The attack makes use of a TCP connection hijack, which can be launched when an attacker that can see the victim’s outgoing packets. There are various cases where this is possible:

- Insecure wireless networks
- Switched networks allowing for ARP poisoning
- Compromised routers
- Tunnel networks such as TOR
- DNS poisoned domain names

Proof of Concept

An attack tool was developed to easily demonstrate this issue. Together with this paper we are publishing surfjack.py which is a python script that makes use of the excellent scapy [3] as its backend. Since most of the times this tool is running on the same network as the victim, it will send its own spoofed response packets before the real web server does. Therefore the victim receives the attacker’s packets rather than the legitimate ones. This is of course not necessarily the case. For example, a DNS cache poisoning attack allows attackers to target victims on other networks. However in the case of a DNS cache poisoning attack, the victim never sends the HTTP requests to the real web server, but rather to the attacker’s IP address. Therefore, unlike the previous examples, the attacker is not racing against a real web server at all!

Apart from hijacking open TCP connections, surfjack.py is able to handle web servers that do not have port 80 open. This means that surfjack.py can easily overcome cases where for example an E-Banking site has closed port 80 to prevent victims from accessing the web server using insecure HTTP connections. Sufjack.py is able to bypass this protection by responding to the victim’s SYN packets.

Surfjack.py supports both Ethernet networks and Wireless. It can send wireless (802.11) frames when it is configured to listen on a Wireless NIC card which is set to monitor mode. The wireless network card has to support injection. However

Surfjack.py also supports making use of separate wireless cards, one for injection and the other for monitoring. This feature allows testers to attack various wireless clients which may be connected to different wireless networks. It also supports WEP keys which may used to decrypt and encrypt packets on the fly.

When the tool runs on a normal Ethernet network, or on a wireless interface which is associated to an access point, it will send Ethernet frames instead of Wireless packets. In many cases, the tester will need to perform an ARP Poisoning or DNS Cache poisoning to be able to see the victim's network traffic.

Surfjack.py has been tested in the following lab environments:

- WiFi networks without any key
- WiFi networks with a WEP key
- Ethernet switched network while performing an ARP poisoning attack

Once the tester has gathered the victim's cookies, he or she can set the web browser to pass through a local proxy which is part of the Surfjack tool and listens on Localhost port 8080. This proxy redirects the attacker to all the sites that Surfjack.py has collected the cookie for while using the "Set-Cookie" header to set the cookie on the attacker's web browser of choice. The tester can then simply browse to the hijacked websites as the victim and make use of both HTTP or HTTPS as need be.

Prior Art

The Side Jacking [4] attack demonstrated by Errata Security at Blackhat 2007 showed how easy it is to hijack HTTP sessions when the victim is not making use of secure HTTP. Side Jacking is a passive attack and relies on the victim browsing to specific websites while the attacker is monitoring the traffic. The same security issue was mentioned previously in various places [5].

Another interesting attack tool presented at Defcon 12 was “airpwn” [6]. This tool watches traffic on wireless and actively sends HTTP responses to the web browser. In reality the attack performed by this tool is very similar to the attack described in this paper. The difference is that in the case of “Surf Jacking” the attacker gains access to the victim’s session, even those passing through HTTPS, and does this in a stealth manner.

In 2007 [7] a Swedish security professional posted the usernames and passwords for various nation’s embassies and political parties. This was the result of a TOR node passively sniffing out-going traffic. As one can imagine, there is nothing stopping someone applying “Surf Jacking” to TOR.

A post on the popular “`Bugtraq” mailing list by Mike Perry [8] described how Gmail is vulnerable to this attack because it does not make use of the “secure” flag in the cookie and therefore is transmitted for both “http” and “https” sites.

A similar post was published on the International Computer Science Institute’s blog [9] published which includes an excellent detailed description of how Gmail is vulnerable.

Vulnerable Services

During our research we tested various services. Although we would have liked to test more online banking services, this was severely limited to the collective amount of bank accounts that we have access to.

The following table summarizes the results:

Web Service	Tested vulnerable	Notes
Google (Gmail / Checkout / Adsense)	Yes	Contacted. Gmail now has the option to force HTTPS.
Anonymous Bank 1	Yes	Contacted and fixed the issue
Anonymous Bank 2	Yes	Contacted and working on a fix
Salesforce	Yes	Fixed the issue silently
Anonymous Online Book Store	Yes	Contacted - no response
Skype website	Yes	Contacted - no response
Godaddy	Yes	Contacted - responded asking for details.

The following is a timeline of contacting different parties that were found vulnerable to this security flaw.

Date	Action
01 May 2008	Contacted Google's security team. Was informed that they are already working on a fix.
16 May 2008	Meeting with Anonymous Bank 1
12 Jul 2008	Contacted GoDaddy and started conversation
15 Jul 2008	Skype: filled out the form to submit a security concern. Received automated response: "Thank you for your interest in partnering with Skype; the world's fastest-growing internet communication offering."
24 Jul 2008	Gmail implemented the "Always use https" option which fixes the flaw.
24 Jul 2008	Sent full details of the vulnerability
27 Jul 2008	Sent an email to security@skype.com and secure@skype.com. Same automated response.
29 Jul 2008	GoDaddy asks if there is a specific URL, which I send. I explain that other web interfaces might be vulnerable as well.
31 Jul 2008	Meeting with Anonymous Bank 2
01 Aug 2008	SalesForce was tested again and found to have silently fixed the problem

Suggestions

For web developers

Cookies can have a flag called “secure”, which when present causes the browser cookie to be sent only through encrypted channels. When the web browser is redirected to a clear text channel (HTTP rather than HTTPS), such cookies are not included in the HTTP request. This behavior solves the problem described and can be easily implemented on web services that separate services that need encryption from those that do not. An of such services is E-banking, which is normally segregated on a website such as https://secure.bank.com/.

However there are cases where setting the Cookie secure flag is not an easy option. A web service such as Google shares a session and credentials across various domains and switches between HTTP and HTTPS depending on the service. In this case, the solution is not as easy as setting the Cookie to “secure” because that would not scale well with the rest of the infrastructure. Google appears to be have mitigated this for Gmail by providing the “use only HTTPS” option - but other Google services such as Google Docs remain vulnerable to attack.

For end users

Apart from releasing the proof of concept tool, we are working to create an easy way to detect sites which are vulnerable to this attack. This tool will be called “Surf Jack Checker” and tests if the site supports HTTPS and if any of the cookies have the secure flag set. Apart from detecting vulnerable web services, the tool will allows end users to protect themselves against this attack by forcing the secure flag when a domain is added to the list of vulnerable websites.

About EnableSecurity

EnableSecurity is dedicated to providing high quality Information Security Consultancy, Research and Development. EnableSecurity is focused on analysis of security challenges and providing solutions to such threats. EnableSecurity works on developing custom targeted security solutions, as well as working with existing off the shelf security tools to provide the best results for their customers. More information about EnableSecurity can be found at enablesecurity.com.

Disclaimer

The information within this document may change without notice. Use of this information constitutes acceptance for use in an AS IS condition. There are NO warranties with regard to this information. In no event shall the author be liable for any consequences whatsoever arising out of or in connection with the use or spread of this information. Any use of this information lies within the realm of the user's responsibility.

Redistribution

---

[1] Internet Study 2007 - http://tinyurl.com/6zbt7r

[2] HTTP/1.1 - RFC 2616

[3] Scapy is a powerful interactive packet manipulation program

[4] Side Jacking http://www.webopedia.com/TERM/S/SideJacking.html

[5] Cookie of Death http://tinyurl.com/58sakb

[6] airpwn - http://airpwn.sourceforge.net/

[7] Embassy leaks highlight pitfalls of Tor http://tinyurl.com/56sm4h

[8] Active Gmail "Sidejacking" - https is NOT ENOUGH http://tinyurl.com/6dlmnt

[9] Sidejacking, Forced Sidejacking and Gmail - http://tinyurl.com/6gu9ba