IBM

Active Man in the Middle Attacks

A SECURITY ADVISORY

A whitepaper from IBM Rational Application Security Group

Roi Saltzman

Adi Sharabani

February 27, 2009

Abstract

Surfing the internet through untrustworthy public networks (whether wired or wireless) has been known to be risky for a long time now. We all think twice before logging into our bank account or accessing any kind of sensitive information, but what about simply browsing our favorite news site?

In this paper, we revisit 'Man-in-the-Middle' attacks and examine in detail a frightening category of MitM attacks that targets WebApplications. We will show in detail how an attacker can steal users' private data *for any site the attacker chooses* when the victim uses a public network, even though all the victim does is read the latest news headlines on a harmless site. The attack methods we will describe work regardless of whether the wireless network is encrypted.

"Passive attack" is the term we will use to describe known methods in which an attacker intercepts (views or modifies) sensitive data sent to or received by a user from the router in an untrusted network, by deploying a Man–in-the-Middle attack (using techniques such as Arp Poisoning, Intercepting Wi-Fi traffic, Rogue Access Point, etc.).

The new category of attack that will be presented here enables an attacker to harm even a cautious user who avoids the risk of Passive attacks by surfing only "vanilla" sites such as news sites. We will call this category of attack "Active attacks".

Microsoft suggests various rules for using public wireless networks safely[i]. For example, the user is advised to use a firewall, not to connect to unencrypted networks, and not to submit sensitive information. These are the commonly known precautions for using a public network securely. They protect against Passive attacks, but are not enough to protect against Active ones.

Using our "active" methods we shall introduce, to attack cautious yet innocent users taking precautions against *Passive* attacks, an attacker could achieve the following goals while the user browses an "innocent" site:

1. Steal the victim's session cookies for any other site
2. Override Same Origin Policy for any other site (this has the same impact as XSS)
3. Steal the browser's saved passwords for any other site
4. Poison the browser cache for any other site (this will make the attack persistent)
5. Maybe even more…

Dealing with the issue of Active attacks is not a question of fixing specific bugs since, as we will show, the issue is the result of a fundamental design flaw. Browsing the Internet using a public network cannot be considered safe no matter which sites you visit or what information you submit.

We will also suggest some remediation actions you can perform to safeguard yourself from Active attacks.

Table of Contents

Passive and Active attacks

The Principle of Passive attacks

It is common knowledge that when using an untrusted network, such as Internet Café, your communication with the gateway (i.e. a router or access point) must always be treated as untrustworthy.

The methods available for intercepting users' traffic using a Man-in-the-Middle attack in order to initiate a Passive attack have already been widely discussed.

For instance, an attacker sets-up an access point with the same network name and MAC address as a genuine one. This gives the attacker complete control over the communication of everyone who mistakenly uses the rogue network instead of the genuine one. This attack is described in the book "Wireless Hacking: Breaking Through "[ii].

Although Passive attacks work well and constitute a serious risk when using a public network, users can protect themselves by following a few general security guidelines, such as:

1. Use SSL as much as possible:
   - Avoid logging in to any non-SSL sites
   - Make sure the site's SSL certificate is valid
2. If you must access a non-SSL site:
   - Do not supply sensitive information: Credit Cards, usernames and passwords.
   - Be aware that the integrity of the information you see on such a Web site might have been compromised
   - Do not execute applications downloaded from the site
   - Do not install software updates, ActiveX controls, browser extensions, etc.

Of course, some of the attack vectors are still valid even if these security guidelines are followed, but the attacker cannot cause much harm, since the user never surfs to any site that deals with sensitive information.

The Principle of Active attacks

In the Active attack scenario, a malevolent third party manipulates a response within a legitimate session in a way that tricks the client into issuing an unwanted request (unknown to the user) that discloses sensitive information. The attacker can then apply a regular Passive attack on this information. It is important to emphasize that this is made possible by a design flaw, not an implementation error or bug.

We describe this type of attack as "active" rather than "passive" because of two essential differences in the nature of the attack:

- It is initiated by the attacker rather than the victim
- The target is entirely controlled by the attacker, rather than being limited by the extent of the victim's browsing activity

Injecting an IFrame

Active attacks can be initiated in several ways. One way, which we will use throughout this paper, is by injecting a specially crafted object such as an IFrame. Injecting an IFrame in the response inside an HTTP session will cause the user's browser to send out a request for the SRC of the IFrame along with the site's credentials.

When the victim browses a news site, for example, the attacker might return a modified attack page that is identical to the original page, except for an extra line containing a malicious, and probably invisible, IFrame:

```
<HTML>
        <HEAD>
                <TITLE>World Wide News </TITLE>
        </HEAD>
        <BODY>
        World Wide News Original Content
                <IFRAME SRC="http://webmail.site" WIDTH="0" HEIGHT="0"></IFRAME>
        </BODY>
</HTML>
```

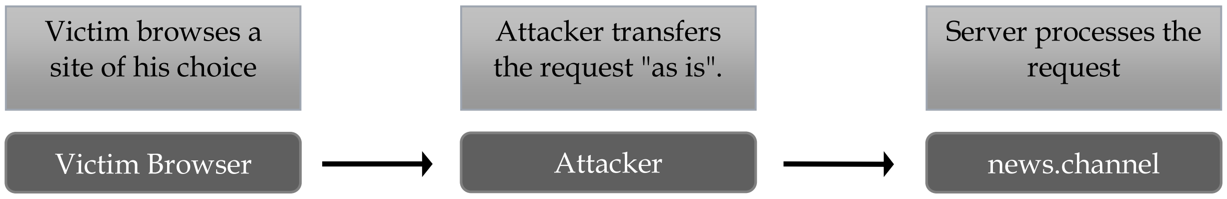
Figure 1: HTML page with an injected IFrame

When the browser renders the response, it will automatically send a request for the site specified as the SRC of the IFrame.

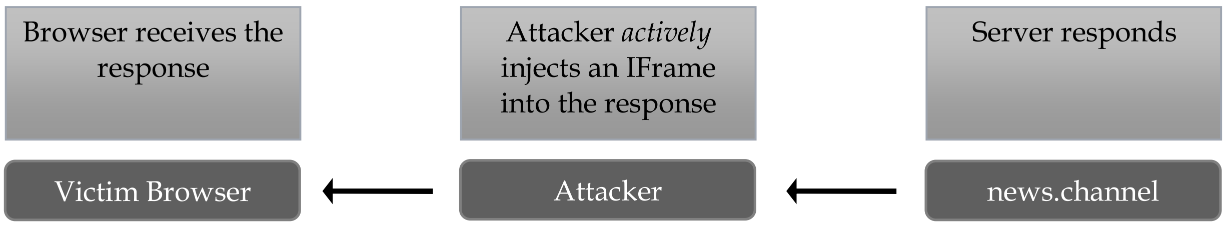
Active attack Flow

The technique described above can be illustrated with the following attack flow.

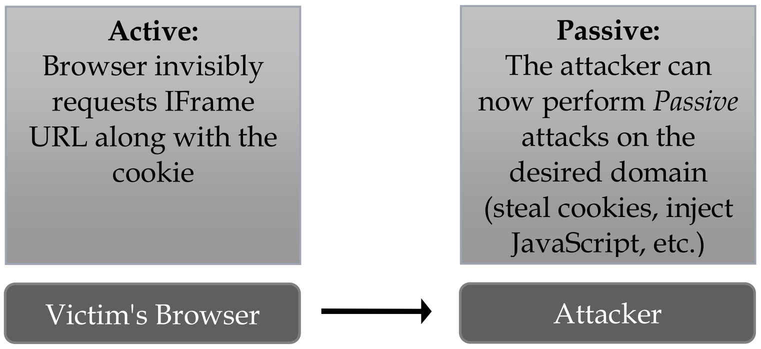
1: The victim browses his favorite news site (a request is sent to "http://news.channel")


Victim browses a site of his choice
Attacker transfers the request "as is".
Server processes the request
Victim Browser
Attacker
news.channel

2: The attacker intercepts the response from "http://news.channel", and injects an IFrame with SRC set to "http://webmail.site" (a site for which the victim has credentials that the attacker wants to steal).


Browser receives the response
Attacker *actively* injects an IFrame into the response
Server responds
Victim Browser
Attacker
news.channel

3: The user's browser renders the IFrame object in the response and sends an automatic request to the source of the IFrame (http://webmail.site) with the user credentials for the site (i.e. the user's cookies). The attacker has now obtained the user's credentials, and controls the response from http://webmail.site. The attacker can now perform Passive attacks such as impersonating the victim using the cookies he has obtained, Inject arbitrary JavaScript into the response, and execute transaction on behalf of the victim.


Active:
Browser invisibly requests IFrame URL along with the cookie
Passive:
The attacker can now perform *Passive* attacks on the desired domain (steal cookies, inject JavaScript, etc.)
Victim's Browser
Attacker

Attack Scenarios

In this section, we will discuss the various attacks possible, giving Passive and Active examples.

Stealing Session Cookies

Session cookies are set by sites as a means of identifying users. Because cookies usually function as authentication tokens, keeping them protected is essential for user privacy.

Passive attack

If a user logs in to a non-SSL website, his browser sends unencrypted HTTP requests that contain his session cookies for that domain.

An attacker with the ability to eavesdrop on the user's network communication can intercept these cookies and impersonate the victim, giving him access to the victim's sensitive information.

This attack is described by Robert Graham under the name "SideJacking", on his blog[iii].

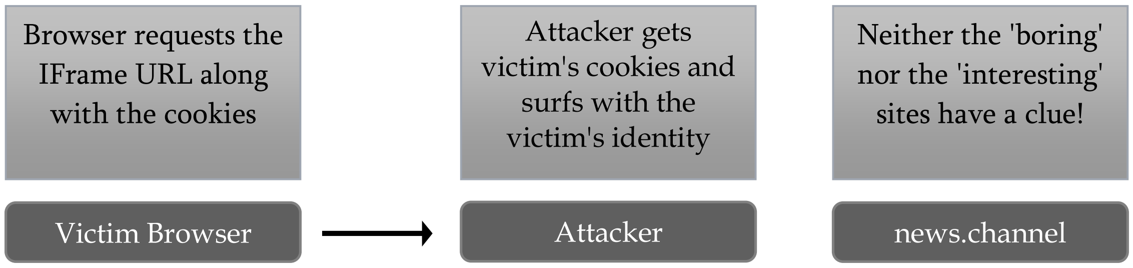
Active attack

In the Active version of the attack, whilst the victim browses some mundane site, the attacker intercepts the response and injects an IFrame that refers to a site of interest to the attacker. The user's browser renders the IFrame and sends a request for the URL along with the user's cookies for that site. A variant of this attack known as "Active SideJacking" has first been described by Mike Perry[iv] and later, under the name "SurfJacking", by Sandro Gauci from EnableSecyrity [v].

This attack has two major advantages, for the attacker, over a Passive attack:
1) The attacker initiates it
2) The attacker controls which sites to steal cookies from (as long as they exist)

These steps set the scene for the classic Passive Session Cookie Stealing attack that is described above. Using his ability to capture the victim's traffic, the attacker now has the victim's cookies for the IFrame source and can thereby access the target site using the victim's identity.

This is what the flow would look like after the injection of an IFrame with a source of "http://webmail.site":


Browser requests the IFrame URL along with the cookies
Attacker gets victim's cookies and surfs with the victim's identity
Neither the 'boring' nor the 'interesting' sites have a clue!
Victim Browser
Attacker
news.channel

This Active attack also enables the attacker to capture cookies that are marked as HttpOnly, in contrast to XSS attacks that are incapable of accessing such cookies through JavaScript code.

In general, an attacker will not be able to steal cookies that are marked secure, because the browser will not send them over an insecure connection such as HTTP. Some secure cookies could still be stolen by this attack if an SSL page refers to a non-SSL resource, such as a .JS file, as described in an article by Billy Rios[vi]. However, in such a case the browser will display a pop-up saying that the page contains both secure and insecure information, which limits the threat of such attacks.

Session cookies of pages behind VPN connections can also be stolen, if they refer to a resource outside of VPN LAN such as a .JS file. When the user's browser attempts to retrieve the external resource, it will send a request that an attacker can intercept and attack.

A further limitation of Passive attacks is evident when sites do not rely only on cookies, but employ additional means of user authentication. For example, a site may only allow connections from a list of trusted IP addresses, or employ a two-factor authentication mechanism. In these cases, the cookies stolen with the Passive attack are not enough to impersonate the victim. However, in the case of an Active attack, which originates from the user's browser, these means of user authentication are rendered ineffective.

Overriding Same Origin Policy

In order to understand the following attack we first need to understand a fundamental aspect of browser security: Same Origin/Domain Policy[vii].

Same Origin Policy is a well-known policy that prevents one domain from accessing information from another domain.

Simply put, "Same Domain Policy" means that a script that originates from one domain cannot access the contents of another domain, even if the user is currently logged-in to the other site using the same browser.

Cross-Site Scripting (commonly referred to as XSS) is an attack that usually occurs with Web applications that return unfiltered user input (usually given through a parameter) in their responses. If the attacker can embed a malicious script on the page and entice the victim to visit it (say by providing a link that causes the browser to send the parameter when clicked), the application will return the input and the victim's browser will execute the script *in the context of the vulnerable page*. This enables the attacker to extract and manipulate *any* information from this site in any ways that the victim's credentials allow. Cross-Site Scripting is described in detail in OWASP[viii].

Overriding Same Origin Policy has an impact similar to XSS, and may enable the attacker to perform actions such as stealing the victim's cookies, performing transactions on behalf of the victim, and stealing the victim's keystrokes using a JavaScript key logger - to name just a few.

Passive attacks

In the Passive version of this attack, the attacker is able to intercept site responses and can inject JavaScript code into the pages. The Web application is unaware of this. This enables the attacker to initiate Cross-Site Scripting attacks on any site the victim visits.

Active attack

Instead of waiting passively for the user to visit a site of interest to her, the attacker actively injects an IFrame during a mundane surfing session. When the user's browser renders the response, it sends a request for the site defined in the source of the IFrame, which the attacker allows to reach the server.

The attacker then injects JavaScript code into the response page and sends it to the user. The only limitations to what this JavaScript can do are the limitations of JavaScript itself. The JavaScript can potentially perform any actions that the victim is allowed to perform. Of course the victim sees none of this, because the injected IFrame window is invisible.

Stealing Saved Passwords

Isn't the automatic password filler, offered by browsers, a great feature? When you browse to a site that requires logging in you can just click the submit button without the hassle of having to remember and type in your data again. So useful!

This is how it usually works: A user visits a site, fills in the login form (usually user and password) and submits it. The user's browser suggests saving the information, and the user accepts. After some time, the user returns to the same page and the browser automatically fills the information in for him.

However, RSnake has pointed out[ix] an inherent problem. The problem lies in the fact that the browser fills the information directly to the input element value (based on its name or id attributes) *in clear text*. This allows an attacker to access and steal the values easily!

Although this attack has been known for some time, the latest versions of the common browsers Mozilla Firefox (3.0.6) and Google Chrome (1.0.154.46) are still vulnerable to it.

In the case of Internet Explorer, the attack works only partially. The username can be stolen because it is filled automatically, but the password cannot be invisibly stolen by an attacker, because user interaction with the user textbox element is required.

The attack does not work on SSL pages since it is generally impossible to modify an SSL session.

Passive attack

As we saw in the Same Origin Policy Attack, an attacker can inject any script into any site to which a victim browses. An attacker can then inject a login form into the Web page and a script that automatically extracts the filled values and sends them back to him.

Although using SSL for all transactions with a site effectively thwarts all possible attacks, both Passive and Active, the vast majority of Web sites choose to use SSL only for important transactions, such as login, due to performance considerations. Not only that, but frequently SSL is not used effectively even for login.

The login process is comprised of two phases:
- Getting the login page (pre-login)
- Sending the authentication data (performing login)

Using SSL for *both* of these phases effectively foils attacks, but on many sites the prelogin page is served in regular, unencrypted HTTP containing an SSL form, which POSTs the data to an HTTPS URL. This is very risky, since an attacker can inject content into the login page and steal the user's authentication data using a Same Origin Policy override attack (such as key logging the user's keystrokes, or changing the POST destination to the attacker's site).

Active attack

Using the method described above in The Principle of Active Attacks, in which the victim's browser is tricked into to sending an IFrame to a site of the attacker's choice, an attacker can target any site without the need to lure the victim into browsing to it. The attacker simply embeds an IFrame into a specific site and returns a script and a form as described in the Passive attack above.

The impact of this attack can be devastating. An attacker can steal all of the victim's saved usernames & passwords.

Poisoning the HTTP Cache

Classic Man-in-the-Middle attacks can only access information that exists at the time of the attack. Information the attacker steals (i.e. cookies) is either present pre-attack or generated during the attack, but the attacker has no access to future information. When the victim leaves the scope of the MITM attack, the attack ends. However, HTTP Cache poisoning makes the MITM attack persistent.

Passive attack

We have already seen that it is possible to modify the server's response to the user. Modifying the HTTP header's expiry date to one that's a long way off, ensures that each time the user visits a page the browser will load the *cached* page, and the attacker's malicious script will be executed, maintaining the attacker's control.

For example, setting the HTTP header to these values will tell the browser to cache the page for about a year:

```
Expires: Fri, 27 Feb 2010 14:19:41 GMT
Cache-Control: public
```

Figure 2: HTTP Headers that indicate a page should be cached

Active attack

As with the other Passive attacks, the limitation of the "HTTP cache poisoning" attack described above is that in order to perform it the attacker needs to wait for the victim to surf to a site of interest. This might take a long while or never happen at all.

An attacker can overcome this limitation by using the Active attack technique, and then attack the victim using the method described in the Passive attack section.

Furthermore, if the attacker chooses to cache-poison popular sites, or maybe even cache a poisoned copy of the Google Analytics JavaScript[x] (to which many sites refer) the victim will be vulnerable to the attacks mentioned above even after the victim stops using the attacker's compromised public network and connects to the Internet by some other means.

Simple Attack (Direct Poison)

An attacker can cache-poison a site the user is likely to use often. This will mean that in any subsequent visits to this site, the cached page, still containing the attacker's malicious code, will be loaded.

Example:

An attacker Injects an IFrame for "http://any.site/", and returns a cache-poisoned, JavaScript filled page. The user leaves the public network and connects to the Internet from home. When the victim accesses "http://any.site/" again, his browser loads the malicious cached copy of the page instead of the "live" version.

This cached version may open an IFrame to "http://any.site/index.html" (which is not poisoned) and hijack the victim's session with the site; seeing everything he does, manipulating his responses, capturing his keystrokes and so on.

Advanced Attack (Indirect Poison)

An attacker can cache-poison a site the user is less likely to use by cache-poisoning a popular one. On any subsequent visits to the "popular" site, the cached page will be loaded, and the cached page of the less "popular" site *will be loaded too.*

Example:

1) When a user browses "http://my.news.channel" an attacker can inject an IFrame that will refer to "http://www.any.site/evil.htm" (the site can exist in the victim's corporate internal LAN, and it is not necessary for him to be able to access it at the time of the attack).

2) Return a cache-poisoned response to the IFrame request, which contains a form and some JavaScript code.

3) Poison the cache of "http://my.news.channel" (or any other common site), so that whenever the user browses "http://my.news.channel" he will be delivered the IFrame that refers to "http://www.any.site/evil.htm".

Now, whenever the victim connects to "http://my.news.channel" he will be redirected to "http://www.any.site/evil.htm" and a malicious JavaScript will be executed within its domain.

This can allow the attacker to override Same Origin Policy for local and internal servers with common IP's like 192.168.0.1, or suchlike. This attack vector will persist on the victim’s machine, and the attacker can decide which sites to poison and which to execute the attack from.

Stealing the User's Cache

When a user fetches a page on a Web site, the server may tell the browser to cache a local copy of this page for future use. The server indicates whether a page should be cached or not by setting values in the response's HTTP header. Cached pages represent the user's browsing session and therefore a user should always treat them as sensitive information.

Using the method described in "Active Attack Flow" above, an attacker can inject an IFrame that refers to a site of interest to the attacker. The attacker then forges a response to this request that contains another IFrame referring to a page in the same domain. If a cached copy of this page exists, the browser will load the cached copy instead of requesting it from the server. The attacker can then extract information from the cached page, since it is in the same domain.

Cookie Fixation

Although we will not go into further details here, a "Session Fixation" attack is also possible.

Using the method described above in The Principle of Active Attacks, the victim is tricked into to sending an IFrame to a site of the attacker's choice, only this time the attacker records only the Session ID that the server had set for the user. As long as the server does not change this Session ID and is susceptible to a "Session Fixation" attack, the attacker can log in to the site using the SID he recorded earlier, and thus impersonate the user. A thorough explanation of this attack can be found at OWASP[xi]

Remediation

We have shown that users who follow all the commonly recommended security instructions are nonetheless vulnerable to Active attacks. However, there are some practical steps that - though they do not prevent these attacks - can limit their impact.

End users

"Clean Slate" Approach

As we have seen, Active attacks:

- Endanger even information that the user did *not* choose to expose during the current session
- Are persistent beyond the current session

To avoid the risk of such attacks, whenever we use an un-trusted network we should take care to connect and disconnect with a "clean slate". When we connect there should be no sensitive information on our computer that is accessible to the browser, and at the end of the session all potentially malicious information created should be deleted.

Suggested safe browsing practices:

Method 1

1. Before connecting to any un-trusted network, delete all browser cookies & cache files. That way there is nothing for an attacker to steal.

2. After disconnecting from any un-trusted network, delete all browser cookies & cache files. That way, if any of the cookies or cache files have been poisoned, the attack will not persist in future browsing.

Method 2

Another way of accomplishing the same thing is by always using two different browsers (not tabs or instances!), one for trusted networks and one for un-trusted networks. That way the browser used for untrusted networks will never have access to sensitive

Beware of Form Fillers

As seen above, it is quite easy to steal user information that is filled by automatic form fillers. You should either disable such form fillers or use form fillers that require some user interaction (such as entering a master password).

If you use the "Clean slate" approach, with separate browsers, you can of course allow the untrusted browser form-filler to work as usual.

Site Administrators

SSL-iziation

The heart of MITM attacks lies in eavesdropping on, or injecting into, unencrypted network traffic. However, many sites send and receive session cookies and other data via an unencrypted HTTP connection, for performance reasons. Attackers can therefore easily intercept the cookies of other users and impersonate those users on the relevant websites. By always using SSL for sensitive information, an administrator can be certain that his users are safe from some of the risks presented by both Passive or Active attacks, since the encrypted communication cannot be eavesdropped on or injected into.

Another countermeasure that can be applied is setting the Secure cookie attribute. This will render impossible any attempts to either leak the cookie to a third-party site or to leak the cookie and sniff it.

By setting the Secure attribute, we ensure that cookies are sent encrypted, denying attackers the possibility of intercepting them.

Industry

Trusted/Untrusted Networks Separation

Integrating a Trusted/Untrusted network separation feature into browsers could thwart the attacks we have discussed in this paper.

Through an interface in the browser, the user would define which networks are to be considered safe and which are not. Content received from one network would be allowed to access and modify cookies or cache pages of that specific network only.

Configured properly, trust-based separation of this kind would render the above-mentioned attacks useless.

Signed HTTP

For some time now, we have felt the need for a standard protocol that signs HTTP requests or responses.

While HTTPS allows the user to receive encrypted and signed information, in many cases there is no need for the encryption itself because the data transmitted is not interesting in its own right.

The common use-case is the release of application updates. Companies tend not to release updates through SSL, since it is too heavy on bandwidth and often crashes their servers. A Signed HTTP-protocol would be of great value in such cases. There have been some attempts to implement such a protocol, but, as far as we know, nothing has yet been standardized.

The great benefit of Signed HTTP is that it doesn't have the performance issues of SSL, yet still maintains the integrity of the data. SSL is just too heavy for many vendors to use fully. For this reason they do not release software updates through SSL (as they should) but instead release them in plain HTTP, implementing the integrity check themselves, or not at all. This integrity-check code is written individually by vendors, requiring development time as well as being bug-prone.

Standardizing this idea in an RFC - and then implementing it in browsers and Web servers - would be a great addition to current protocols and could be used easily by many vendors.

With Signed HTTP, we can mitigate the risk of an Active MITM attack without the overhead of SSL.

Acknowledgements

We are grateful to the following people for their contribution to this paper:

- Jonathan Afek
- Yair Amit
- Jonathan Cohen
- Guy Podjarny
- Danny Allan

References

[i] Microsoft "Public Wireless Networks - How To Stay Safe"
http://www.microsoft.com/protect/yourself/mobile/publicwireless.mspx

[ii] "Wireless Hacking: Breaking Through" Rogue Access Point Excerpt
http://www.informit.com/articles/article.aspx?p=353735&seqNum=7

[iii] SideJacking by Robert Graham
http://erratasec.blogspot.com/2008/01/more-sidejacking.html

[iv] "Active SideJacking" by Mike Perry
http://seclists.org/bugtraq/2007/Aug/0070.html

[v] "SurfJacking" by Sandro Gauci, ENABLESECURITY
http://resources.enablesecurity.com/resources/Surf%20Jacking.pdf

[vi] SurfJacking Secure Cookies by Billy Rios
http://xs-sniper.com/blog/2008/09/24/surf-jacking-secure-cookies/

[vii] Same Origin Policy Page in Wikipedia
http://en.wikipedia.org/wiki/Same_origin_policy

[viii] OWASP Page on Cross Site Scripting (XSS)
http://www.owasp.org/index.php/XSS

[ix] Stealing form filling information by RSnake
http://ha.ckers.org/blog/20060821/stealing-user-information-via-automatic-form-filling/

[x] Google Analytics JS
http://www.google-analytics.com/ga.js

[xi] OWASP Page on Session Fixation
http://www.owasp.org/index.php/Session_Fixation