

Secure Session Management for Web Applications

Chris Palmer
iSEC Partners, Inc.
<https://www.isecpartners.com/>

Who Am I?

iSEC Partners: software security engineering consultants.

We are former developers and ops engineers; security researchers. I have been a web app developer and Staff Technologist at the Electronic Frontier Foundation.

We help software vendors (huge and small) make their products safer. We find bugs, explain the impact, identify the solutions.

We work on: native apps, web apps, operating systems. Software you have heard of and are using now.

Some bugs show up again and again. Hopefully this talk will help you avoid them!

Who Are You?

Jobs you do:

- Software development
- Software testing
- UI/UX development
- Management, business development
- Application server deployment

Testing tools you use:

- Wireshark
- WebScarab
- Perl/Python/Ruby
- Firefox extensions: Add 'n' Edit Cookies, Web Developer, Firebug

Who Are You?

Languages and platforms you use:

- Java
- Ruby
- .NET
- Unix/Linux
- Windows
- Python
- Functional hipster language (Scala, Haskell, Erlang, Miranda)
- C/C++ (D?!)
- SQL
- C-Store
- Amazon S3
- Other storage

Terminology

Session token

- Anything you use to link separate requests into a single session: HTTP cookie, Flash LSO, magic GET or POST parameters, magic URL components.
- HTTP cookies are perhaps the most common mechanism.

MITM and active network attacks

- Monkey (or Man, or Person) In The Middle: the attacker controls the network infrastructure somewhere between legitimate client and legitimate server.
- Various ways to achieve this attack.
- **If everything goes well**, TLS v1/SSL v3 prevents it.

Terminology

Passive network attack

- When the attacker does not need to own the network infrastructure.
- The passive attacker is assumed to have something like Wireshark (<http://www.wireshark.org/>).
- The passive attacker may also control their own web content, like <http://www.cybervillains.com/>.
- An active attacker is usually assumed to also have passive attack capability.

Terminology: People

Alice and Bob

- Legitimate, honest users of example.com.
- Mutually trusting... mostly.

Mallory

- Malicious user of example.com.
- Runs cybervillains.com.
- May have software development skills.

Dave

- Devilish user of example.com.
- May collude with Mallory.
- May have software development skills.

Eve

- Can passively eavesdrop on Alice and Bob.

The Internet Security Model

"Hopefully, some packets will get to some host, some of the time. They might not even be too badly damaged when they arrive!"

Pop-up blocked. To see this pop-up or additional options click here...



Secure this connection



Feeding issues are tough. Addressing them can be easy.



Quiets colic symptoms* fast, often within 48 hours.**



Designed for babies with fussiness or gas.



Clinically proven to reduce spit up by more than 40%**

FOR A FREE FORMULA SAMPLE [CLICK HERE >](#)

Close X



*Due to cow's milk protein allergy. **Based on clinical studies of the same formula before the addition of LIPIL.

[Web](#) [Images](#) [Video](#) [News](#) [Maps](#) [Gmail](#) [more](#)

Download Google Toolbar



Google Search I'm Feeling Lucky

[Advanced Search](#)
[Preferences](#)
[Language Tools](#)

[Advertising Programs](#) - [Business Solutions](#) - [About Google](#)

[Make Google Your Homepage!](#)

©2007 Google

MITM Is A Viable Business Model

Gee, do you think I should have clicked on the Download Google Toolbar button?

- Trademark and/or copyright violation goodness in addition to potential malware.

See also recent excitement:

[http://blog.washingtonpost.com/securityfix/2008/04/
when_monetizing_isp_traffic_go.html](http://blog.washingtonpost.com/securityfix/2008/04/when_monetizing_isp_traffic_go.html)

http://www.doxpara.com/DMK_Neut_toor.ppt

Monkey-in-the-middle is so reliable, it can be depended on for revenue. Attackers rely on it just as much as "misguided" marketing departments. Defend your users!

Most New Applications Are Web-based

Most web applications, especially new ones, are unsafe for use on the internet.

Really.

The web is one of the most complex application platforms ever, and every application is connected to every other application in the world.

Despite this, it is possible to build secure applications -- but not necessarily easy. Today we'll talk about session management, a key to building secure applications.

Problems **and** solutions!

Security Assertions We Want to Make

- Alice cannot read or write Bob's data unless he wants her to.
- Mallory cannot buy stuff with Alice's money.
- Dave cannot cause the service to be unavailable for Alice and Bob.
- When Alice reads PopularBlog.com, code on the page cannot read the contents of the other tab she has open: AjaxMail.com.
- Nor should PopularBlog be able to make requests to AjaxMail against Alice's will.
- Dave can not take control of the server.

Users who discover they are not safe on your site are unhappy users.

If assertions like these do not hold for your application, your users might be attacked. They might take their money, clicks, mindshare, or friends elsewhere.

Security Costs Money

Developing secure applications now is cheaper than dealing with the fallout later.

Just ask iSEC's clients!

One of our clients is Smart RIA Startup. I tested their app, and they neatly avoid many common pitfalls.

Another of our clients is named Large Success Story That Pays Too Much For Remediation Now. (Better security would equal better profit margins!)

Session Management Underlies Many Security Assertions

We need to know that **this request** really came from the **person linked with the Alice account** before we can make security decisions regarding Alice's data.

Have we been talking to the same person, linked with the same account, for the whole session? Could anyone have hijacked control of the real Alice's session?

What about single sign-on solutions (multiple applications accessed from a single session)? Do we downgrade the security of the most sensitive app to accommodate an insecure app in an SSO situation?

Session Management Is Often Weak in Real-world Applications

It's surprisingly hard to get right -- there are many little-known gotchas.

1. Lack of HTTPS, incomplete use of HTTPS
2. Broadly-scoped cookies in a domain with many applications
3. Guessable session IDs
4. Cross-site scripting
5. Cross-site request forgery
6. Session fixation
7. Session state stored directly in the cookie, unsafely
 1. Integrity is not the same as confidentiality!
 2. Fake crypto is not crypto!
 3. Static strings in the source code are not secret keys!

The Cookie Is Equivalent to a Password (Temporarily)

Everyone recognizes that we need to keep users' passwords secret. We often use HTTPS for the login sequence for that reason.

After login, how do we know who the user is? Their cookie.

Therefore, the cookie is the key to the user's session -- and assets! Kind of like a password, right?

So we must protect the cookie just as we protect the password.

Unless your app uses HTTPS for login and all post-login traffic, **your users are not safe**. Passive attackers can steal users' data -- and cookies, allowing them to impersonate the real user.

Really?!

Yes, I know SuperImportant.com and MegaMajorAwesome.com don't use HTTPS throughout.

No, their users are not safe, either.

This is a market opportunity for you: Undercut them on operations costs and provide a better user experience at the same time.

You'll need to eliminate cross-site scripting and cross-site request forgery too, of course. Then there's phishing...

HTTPS is a baseline, bare-minimum level of security.

The Internet Security Model (Again)

HTTPS (HTTP with TLS) provides features we need:

- Server authentication (via trusted third party)
- Integrity protection (damaged/mangled packets are detected and discarded; computationally infeasible for an active attacker to damage the packets undetectably)
- Confidentiality protection (computationally infeasible for a passive attacker to decrypt the packets)
- Optional client authentication (again via trusted third party)

Use the *Secure* Attribute on Cookies

If you use HTTPS all the time, that's good -- but not yet unbeatable. If you use HTTP cookies, use the *Secure* attribute (boolean). This tells the browser "Only send this cookie over the network using HTTPS, never HTTP."

Use the *Secure* Attribute

Why? Dave embeds these tags in a page he controls (perhaps by MITM), and which the victim may view (e.g. <http://forum.example.com/>):

```

```

```

```

The browser will send the cookie when making those requests, unless *Secure* is set!

Use the *Secure* Attribute

Bad:

```
Set-Cookie: id=123
```

```
Set-Cookie: id=123; Domain=finance.example.com
```

Good:

```
Set-Cookie: id=183274920182152134; Secure
```

```
Set-Cookie: id=183274920182152134; Secure;  
Domain=finance.example.com
```

Scope the Cookie Narrowly

We also need to set the *Domain* attribute as narrowly as possible. Leaving it blank is best!

Imagine two applications: `https://finance.example.com` and `http://wiki.example.com`.

If the cookie is set with *Domain* `".example.com"`, the wiki will get the cookies for the financial app. What if wiki contains JavaScript supplied by Dave? What if Eve observes the unencrypted traffic to wiki? Oops.

(Note that if *Secure* is set on the finance cookie, the cookie will not go to wiki over HTTP, but still will over HTTPS.)

Scope the Cookie Narrowly

Bad:

```
Set-Cookie: id=123; Domain=.example.com
```

```
Set-Cookie: id=123; Secure; Domain=.example.com
```

Good:

```
Set-Cookie: id=183274920182152134; Secure;
```

```
Set-Cookie: id=183274920182152134; Secure;
```

```
Domain=finance.example.com
```

WebScarab - conversation 2

Previous Next 2 - POST https://labs.isecpartners.com:443/chris/expo-secure/ 302 Found

Parsed Raw

POST https://labs.isecpartners.com:443/chris/expo-secure/ HTTP/1.1
Host: labs.isecpartners.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:1.8.1.14) Gecko/20080404 Firefox/2.0.0.14
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: https://labs.isecpartners.com/chris/expo-secure/
Content-Type: application/x-www-form-urlencoded
Content-length: 29

username=chris&password=hello

Parsed Raw

HTTP/1.1 302 Found
Date: Wed, 23 Apr 2008 23:12:42 GMT
Server: Apache/2.0.55 (Ubuntu) PHP/5.1.6 mod_ssl/2.0.55 OpenSSL/0.9.8b
X-Powered-By: PHP/5.1.6
Set-Cookie: session=fba759fc31d0556fc3b90e1b65a9c382; expires=Wed, 23-Apr-2008 23:22:42 GMT; path=/chris/expo-secure; secure
Location: /chris/expo-secure/welcome.php
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
X-Transfer-Encoding: chunked
Content-Type: text/html; charset=UTF-8
Content-length: 0

128 bits

**Secure is set,
Domain is not**

WebScarab - conversation 12

Previous Next 12 - GET https://labs.isecpartners.com:443/chris/expo-secure/welcome.php 200 OK

Parsed Raw

```
GET https://labs.isecpartners.com:443/chris/expo-secure/welcome.php HTTP/1.1
Host: labs.isecpartners.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:1.8.1.14) Gecko/20080404 Firefox/2.0.0.14
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: https://labs.isecpartners.com/chris/expo-secure/index.php
Cookie: session=cc1f4a44819ed351c4d77cfc052f925a
```

Cookie: header sent because this is an HTTPS connection.

Parsed Raw

```
HTTP/1.1 200 OK
Date: Wed, 23 Apr 2008 23:28:31 GMT
Server: Apache/2.0.55 (Ubuntu) PHP/5.1.6 mod_ssl/2.0.55 OpenSSL/0.9.8b
X-Powered-By: PHP/5.1.6
Content-length: 91
Keep-Alive: timeout=15, max=99
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
```

```
<h1>Thanks for logging in! Here's the kitty.</h1>

```

We get the content because we are authenticated.

WebScarab - conversation 13

Previous Next 13 - GET https://www.isecpartners.com:443/chris/expo-secure/welcome.php 404 Not Found

Parsed Raw

```
GET https://www.isecpartners.com:443/chris/expo-secure/welcome.php HTTP/1.1
Host: www.isecpartners.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:1.8.1.14) Gecko/20080404 Firefox/2.0.0.14
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

**Cookie: header not sent:
Domain does not match
"labs.isecpartners.com"**

Parsed Raw

```
HTTP/1.1 404 Not Found
Date: Wed, 23 Apr 2008 23:29:07 GMT
Server: Apache
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
X-Transfer-Encoding: chunked
Content-Type: text/html; charset=iso-8859-1
Content-length: 223

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>404 Not Found</TITLE>
</HEAD><BODY>
```

WebScarab - conversation 7

Previous Next 7 - GET http://labs.isecpartners.com:80/chris/expo-secure/welcome.php 302 Found

Parsed Raw

```
GET http://labs.isecpartners.com:80/chris/expo-secure/welcome.php HTTP/1.1
Host: labs.isecpartners.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:1.8.1.14) Gecko/20080404 Firefox/2.0.0.14
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
```

No Cookie: header, because the cookie was marked Secure and this is an HTTP request.

Parsed Raw

```
HTTP/1.1 302 Found
Date: Wed, 23 Apr 2008 23:23:33 GMT
Server: Apache/2.0.55 (Ubuntu) PHP/5.1.6 mod_ssl/2.0.55 OpenSSL/0.9.8b
X-Powered-By: PHP/5.1.6
Location: https://labs.isecpartners.com/chris/expo-secure/index.php
X-Transfer-Encoding: chunked
Content-Type: text/html; charset=UTF-8
Content-length: 0
```

We are redirected to the login page, because we did not provide a cookie!

Session Tokens Must Be Unpredictable

If the attacker can *guess* the value of a valid session token, that's just as bad as if the attacker *discovered* it. Perhaps worse -- the attacker doesn't have to be near the victim!

Session Tokens Must Be Unpredictable

Bad tokens:

- Auto-incrementing DB table IDs
- Pseudo-random numbers that come from deterministic functions
- Small numbers
 - The cardinality of the set of tokens must be huge relative to the average number of active sessions
- The current time in (milli)seconds
- base64(username)

Session Tokens Must Be Unpredictable

Good tokens:

- 128-bits of entropy from (examples):
 - `/dev/random` (Unix),
 - `java.Security.SecureRandom` (Java),
 - `os.urandom` (Python),
 - `System.Security.Cryptography.RNGCryptoServiceProvider` (C#/.NET)

Session Tokens Must Be Unpredictable

This gives us a huge set (2^{128}) of possible session tokens, evenly distributed, generated in an unpredictable sequence.

That's it, you're done -- no other mangling necessary.

A good web app framework handles this for you (recent .NET, recent Java -- but PHP disables the good behavior by default!).

PHP: Runtime Configuration - Manual - Mozilla Firefox

File Edit View History Bookmarks Tools Help

php http://us2.php.net/manual/en/session.configuration.php

session.gc_maxlifetime	"1440"	PHP_INI_ALL	
session.serialize_handler	"php"	PHP_INI_ALL	
session.cookie_lifetime	"0"	PHP_INI_ALL	
session.cookie_path	"/"	PHP_INI_ALL	
session.cookie_domain	""	PHP_INI_ALL	
session.cookie_secure	""	PHP_INI_ALL	Available since PHP 4.0.4.
session.cookie_httponly	""	PHP_INI_ALL	Available since PHP 5.2.0.
session.use_cookies	"1"	PHP_INI_ALL	
session.use_only_cookies	"1"	PHP_INI_ALL	Available since PHP 4.3.0.
session.referer_check	""	PHP_INI_ALL	
session.entropy_file	""	PHP_INI_ALL	
session.entropy_length	"0"	PHP_INI_ALL	
session.cache_limiter	"nocache"	PHP_INI_ALL	
session.cache_expire	"180"	PHP_INI_ALL	
session.use_trans_sid	"0"	PHP_INI_ALL	PHP_INI_ALL in PHP <= 4.2.3. PHP_INI_PERDIR in PHP < 5. Available since PHP 4.0.3.
session.bug_compat_42	"1"	PHP_INI_ALL	Available since PHP 4.3.0. Removed in PHP 6.0.0.
			Available since

Cross-site Request Forgery

Nothing stops cybervillains.com from formulating requests to example.com.

Even requests that cause a destructive or sensitive action to happen!

If Alice is authenticated with example.com (has a cookie), and visits cybervillains.com in another tab, cybervillains.com can cause her browser to make a dangerous request to example.com.

Example.com will honor the request -- she is authenticated, after all!

Cross-site Request Forgery

Embedded in cybervillains.com/index.html:

```

```

This works because the structure of the delete profile action is predictable.

example.com will see that Alice is authenticated.

example.com has no way to know that Alice didn't intend to do that! It is in every way a valid request.

Cross-site Request Forgery

Dave doesn't need to steal Alice's cookie in order to abuse the authority it confers to her!

How can we stop this from happening?

Consider the common "change password" action: You require the user to provide their current as well as their new password.

Why?

Cross-site Request Forgery

But we can't make Alice re-type her password for every action the application does!

We need to require a parameter to the action that Dave cannot predict or discover, which is unique to Alice's session and to the action.

Answer: an entropic, unique token stored in a hidden field.

```
<form action="https://example.com/deleteProfile.jsp">  
<input type=hidden name=token value=92821729927581921821>  
<input type=submit value="Delete Profile!">
```

Cross-site Request Forgery

The server remembers the token value.

When the server receives a request to delete Alice's profile, it checks that the expected unique, unpredictable token is in the request.

If it is, the request is valid: formulated by the app itself.

If it is not, the request is forged: formulated by someone who didn't know the token value!

Do not honor forged requests.

Conclusion

Security is cheaper now than it is later.

Your competitor is paying too much for remediation. What about you?

Session management underlies crucial security assertions and business requirements.

Conclusion

More nerdy details on session management will soon be available at

<https://www.isecpartners.com/publications.html>

Conclusion

Thank you for coming!

iSEC is hiring:

careers@isecpartners.com

I love questions:

chris@isecpartners.com