Top Vulns & Where to Start

The big picture

OWASP Top 10 2013	±	OWASP Top 10 2017
A1 – Injection	→	A1:2017 – Injection
A2 – Broken Authentication and Session Management	→	A2:2017 – Broken Authentication and Session Management
A3 – Cross-Site Scripting (XSS)	↘	A3:2013 – Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	U	A4:2017 – XML External Entity (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017 – Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017 – Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	U	A7:2017 – Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017 – Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017 – Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017 – Insufficient Logging & Monitoring [NEW, Comm.]

The big picture

OWASP Top 10 2013	±	OWASP Top 10 2017
A1 – Injection	→	A1:2017 – Injection

Injection vulnerabilities

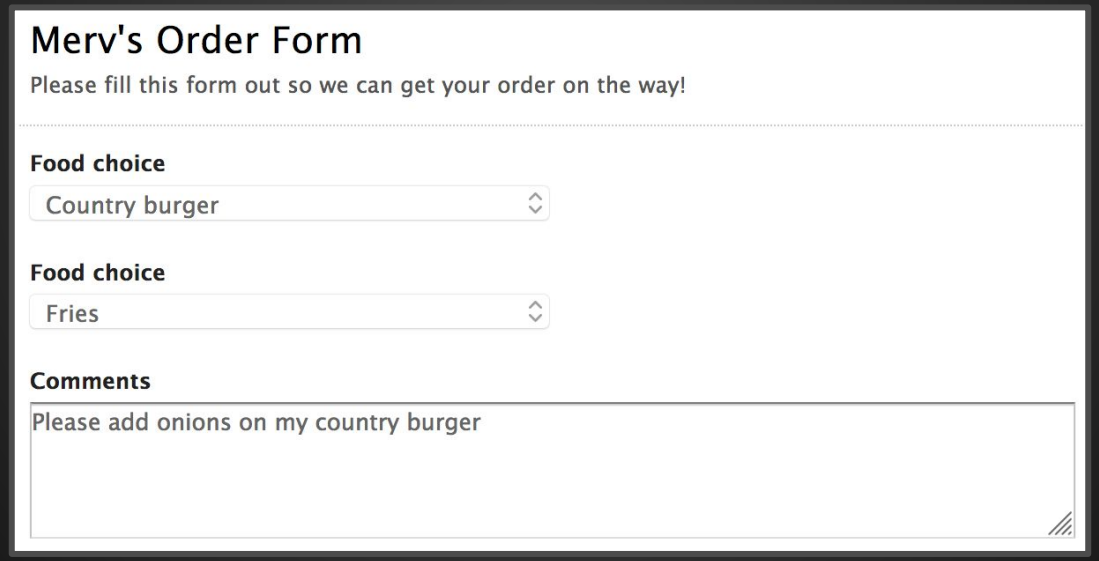
- There’s a common theme in many vulnerabilities you’ve probably heard in the news
- User input escaping its context

Injection vulnerabilities

- Let’s take a look at a simple example, no computers involved.
- Today we’ll learn how to do a burger injection vulnerability (BIV). Next meeting we’ll see BIVs are identical to real vulnerabilities such as BOF, SQLi, XSS, RCE, LFI, SSRF, CSRF, etc.

Burger injection

- Let’s order a hamburger from our favorite restaurant online.



Burger injection: Receipt

MERV'S MELTDOWN SHOP
825 RAILROAD AVE
TALLAHASSEE
FL
32310

CASHIER: MITCH
CUSTOMER: NATHAN

PURCHASE:

COUNTRY BURGER	$8.99
REG FRIES	$3.99
COUNTRY BURGER + ONIONS	$0.00
+% TAX:	$0.00

TOTAL: $12.98

PAYMENT METHOD: CREDIT CARD
TRANSACTION #1536624929 -001
DATE:10/09/2018 4:25:51 PM

THANK YOU

Burger injection

- One day I decided to get a *smoked burger* instead, and forgot to change my order comment about having onions on my *country burger*

Burger injection: Receipt

MERV'S MELTDOWN SHOP
825 RAILROAD AVE
TALLAHASSEE
FL
32310

CASHIER: MITCH
CUSTOMER: NATHAN

PURCHASE:

SMOKED BURGER	$8.99
REG FRIES	$3.99
COUNTRY BURGER + ONIONS	$8.00
+% TAX:	$8.00

TOTAL: $12.98

PAYMENT METHOD: CREDIT CARD
TRANSACTION #1536624870 -001
DATE:10/09/2018 4:25:51 PM

THANK YOU

Burger injection

- Result: we got two burgers, what just happened?

Burger injection: Receipt

Where does our *order* stop and the *comments* begin?

There's no line on our receipt that separates order comments!

```
MERV'S MELTDOWN SHOP
825 RAILROAD AVE
TALLAHASSEE
FL
32310
--------------------------------
CASHIER: MITCH
CUSTOMER: NATHAN
--------------------------------
PURCHASE:

SMOKED BURGER                $8.99
REG FRIES                    $3.99

COUNTRY BURGER + ONIONS      $0.00

 +%              TAX:        $0.00

--------------------------------
                      TOTAL: $12.98

PAYMENT METHOD: CREDIT CARD
TRANSACTION #1536624870 -001
DATE:10/09/2018 4:25:51 PM

        THANK YOU
```

Burger injection

- Takeaway: We got a free burger using our burger injection vulnerability (BIV)
- How do we protect ourselves against these vulnerabilities?

- A line to separate order comments (user input) from the rest of the order

Injection vulnerabilities

- Computers need that same line to delineate user data and what the instructions are.

Authentication & Session Management

When you visit a website how does the server know who you are?

A few ways are possible…

- Credentials (username / password)
- HTTP Cookies
- IP Address

What are HTTP Cookies?

HTTP is a stateless protocol!

- Cookies store information that a server wants web clients to send back in following requests
- HTTP Cookies are a way to facilitate Session Management
- Insecure implementations of Session Management is one way vulnerabilities can occur in computer programs

What are insecure HTTP Cookies?

- Predictable, non-hashed, values
- Reused values
- Very long (or non specified) expiration date
- A wider set of applicable domains than necessary
- Data that should be stored server-side

Examples of Session Cookies

'ASP.NET_SessionId=10x0kp5snfafxpua4vky0q0r; domain=.exampledomain.com; path=/; HttpOnly'

'PHPSESSID=44opurqc0btvdnajfj6qogsqr7; expires=Tue, 18-Sep-2018 01:02:50 GMT; Max-Age=604800; path=/'

‘AuthToken=123456;Username=admin;RoleId=1;loggedIn=True;’

Authentication & Session Management

When you visit a website how does the server know which resources you are allowed to access?

It is generally a good practice to establish roles for your users. Roles determine which resources a given user should be allowed to access.

Servers must be diligent to enforce access by role!

Authentication & Session Management

When you visit a website how does the server know which resources you are allowed to access?

Servers should not store user identifiers and/or user access permissions in the browser!

This can be done server-side after establishing a session

How to validate Authentication & Session Management?

- Check which values are being stored in HTTP cookies
  - *Is the Session Management scheme homebrewed?*

View HTTP Cookies in your browser

- Ctrl + Shift + I
- F12 -> Networking

Find the vulnerability!

Which HTTP Cookie values may be vulnerable?

'ASP.NET_SessionId=1kp5snfafxpua4vkyfjozfezpppaxq0r; userId=1508447;AjaxSessionKey=Z4PhX7uL31m2Y5fUv;MDID=H4sIAAAAAAAAEAGNkYGBgBGI2IGYCsfWBBINg'

Find the vulnerability!

Which HTTP Cookie values may be vulnerable?

‘AuthToken=123456;Username=admin;RoleId=1;loggedIn=True;’

(they all are!!)

How to validate Authentication & Session Management?

- Attempt to access sections of a site that should be limited to users with escalated privileges
  - Especially sections that should require you to be logged in!

How to validate Authentication & Session Management?

- Test for default credentials
- Check for weak password policies (including reset policies)

How to validate Authentication & Session Management?

- Try changing form input values beyond what the browser displays
- Check for anti Cross Site Request Forgery (CSRF) mechanisms
  - Unique AntiForgeryTokens within input forms