Analysis Results

Lightning Browser

Report Date

2022-07-19 08:53:57

Report Author

Classification Method

OWASP Mobile Top 10 2016

Product Version

3.11.3

CONFIDENTIALITY NOTE

This report is intended only for the person(s) or entity to which it is addressed and contains confidential and privileged information. If you are not the intended recipient, you must not view, use, copy, disclose, or otherwise disseminate this report or any part of it. Doing so is strictly prohibited, and may result in legal proceedings. If you received this in error, please notify the sender immediately and destroy any copies of this information.

TABLE OF CONTENTS

PROJECT INFORMATION

Project	Lightning Browser
UUID	ebb0ba76-01cd-4dd8-a89b-ffeec08ced60
Go To Results In	DerScanner

Source code
https://play.google.com/store/apps/details?id=acr.browser.barebones&hl=en&gl=US

Security Level Dynamics

The app score is calculated on a scale from 0 to 5. Score is calculated based on the number of critical and medium level vulnerabilities. The impact of critical vulnerabilities is greater than that of medium level vulnerabilities, and does not take into account the amount of code. Medium level vulnerabilities are taken into account based on their frequency and total number of source code lines.

Vulnerability Dynamics

Vulnerabilities are divided by severity level: critical, medium, low and info.

1. **Critical vulnerabilities** are likely to compromise sensitive data and system integrity.

2. **Medium level vulnerabilities** are less likely to compromise confidential data and system integrity, or are less serious security breaches.

3. **Low level vulnerabilities** can be a potential security threat.

4. **Info level vulnerabilities** signal a violation of good programming practices.

We highly recommend to focus on critical and medium-level vulnerabilities first.

Scan History

Number	Date and Time	Status	Languages	Lines of Code	Number of Vulnerabilities					Score
					Critical	Medium	Low	Info	Total	
1/1	2022-07-18 09:31:55	completed	Java, Config files, PL/SQL	294 599	3	156	358	400	917	2.7/5.0

ABOUT OWASP MOBILE TOP 10 2016

Report classifies the level of vulnerability by **OWASP Mobile Top 10 2016**. The Open Web Application Security Project (OWASP) is an online community which creates freely-available articles, methodologies, documentation, tools, and technologies in the field of web application security. One of its main projects is OWASP Top 10 aiming to raise awareness about application security by identifying some of the most critical risks that organizations face. The Top 10 project is referenced by many standards, books, tools, and organizations, including MITRE, PCI DSS, DISA, FTC, and many more.
Note that some vulnerabilities may belong to the number of categories or to none at all. To see the full list of vulnerabilities. choose the **Bv severitv** classification method.

M1 Improper Platform Usage

This category covers misuse of a platform feature or failure to use platform security controls. It might include Android intents, platform permissions, misuse of TouchID, the Keychain, or some other security control that is part of the mobile operating system.

M2 Insecure Data Storage

Insecure data storage vulnerabilities occur when development teams assume that users or malware will not have access to a mobile device's filesystem and subsequent sensitive information in data stores on the device. Filesystems are easily accessible. When data is not protected properly, specialized tools are all that is needed to view application data.

M3 Insecure Communication

Mobile applications frequently do not protect network traffic. They may use SSL/TLS during authentication but not elsewhere. This inconsistency leads to the risk of exposing data and session IDs to interception. Also, this category includes all communications technologies that a mobile device might use: TCP/IP, Wi-Fi, Bluetooth/Bluetooth-LE, NFC, audio. infrared. GSM. 3G. SMS. etc.

M4 Insecure Authentication

Poor or missing authentication schemes allow an adversary to anonymously execute functionality within the mobile application or backend server used by the mobile application. Weaker authentication for mobile applications is fairly prevalent due to a mobile device's input form factor. The form factor highly encourages short passwords that are often purelv based on 4-digit PINs.

M5 Insufficient Cryptography

In order to exploit this weakness, an adversary must successfully return encrypted code or sensitive data to its original unencrypted form due to weak encryption algorithms or flaws within the encryption process.

M6 Insecure Authorization

This is a category to capture any failures in authorization (e.g., authorization decisions on the client side, forced browsing, etc.). It is distinct from authentication issues. If the app does not authenticate users at all in a situation where it should (e.g., granting anonymous access to some resource or service when authenticated and authorized access is required), then that is an authentication failure. not an authorization failure.

M7 Poor Code Quality

Code quality issues are fairly prevalent within most mobile code. Most code quality issues are fairly benign and result in bad programming practice. This category would capture things like buffer overflows, format string vulnerabilities, and various other code-level mistakes where the solution is to rewrite some code that's running on the mobile device.

M8 Code Tampering

Once the application is delivered to the mobile device, the code and data resources remain there. An attacker can either directly modify the code, change the contents of memory dynamically, change or replace the system APIs that the application uses, or modify the application's data and resources. This can provide the attacker a direct method of subverting the intended use of the software for personal or monetary gain.

M9 Reverse Engineering

Generally, all mobile code is susceptible to reverse engineering. Some apps are more susceptible than others. Code written in languages or frameworks that allow for dynamic introspection at runtime (Java, .NET, Objective C, Swift) are particularly at risk for reverse engineering.

M10 Extraneous Functionality

There is a high likelihood that any given mobile app contains extraneous functionality that is not directly exposed to the user via the interface. Most of this additional code is benign in nature and will not give an attacker any additional insight into backend capabilities. However, some extraneous functionality can be very useful to an attacker.

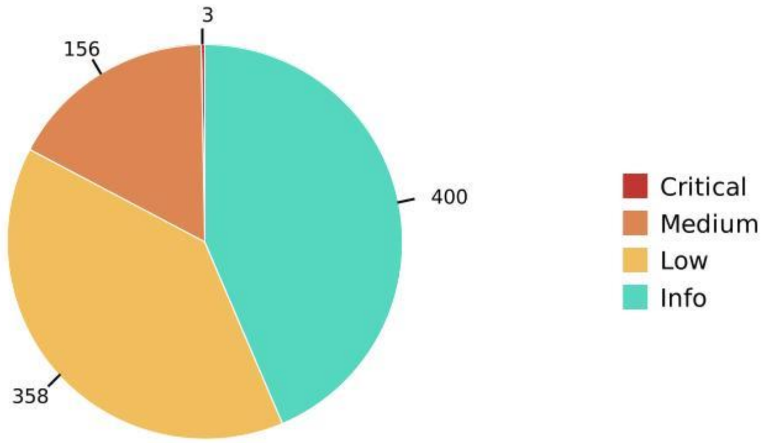
SCAN INFORMATION

1/1 2022-07-18 09:31:55

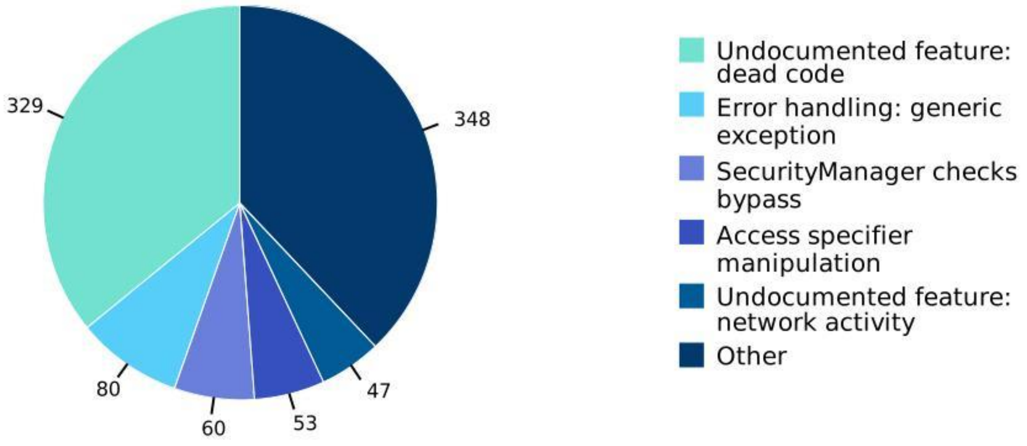
Scan Statistics

Status	completed				
Score	2.7/5.0				
Duration	1:55:15				
Lines of Code	294 599				
Vulnerabilities	Critical **3**	Medium **156**	Low **358**	Info **400**	Total **917**

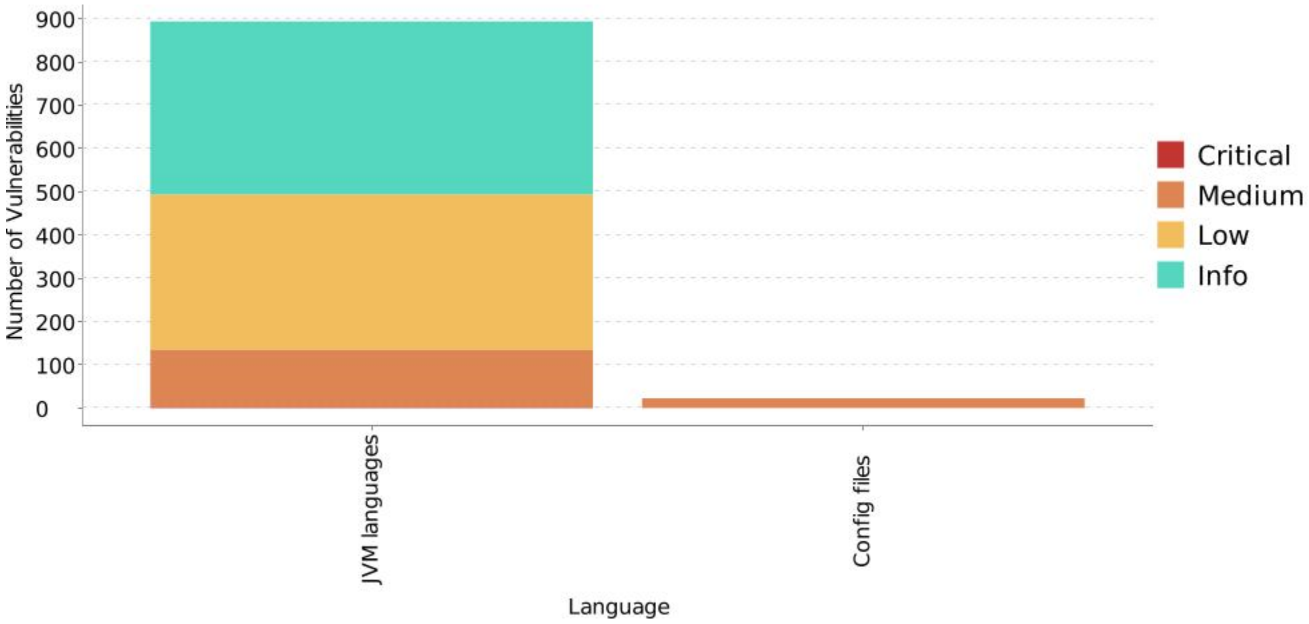
Found Vulnerabilities



Vulnerability Types

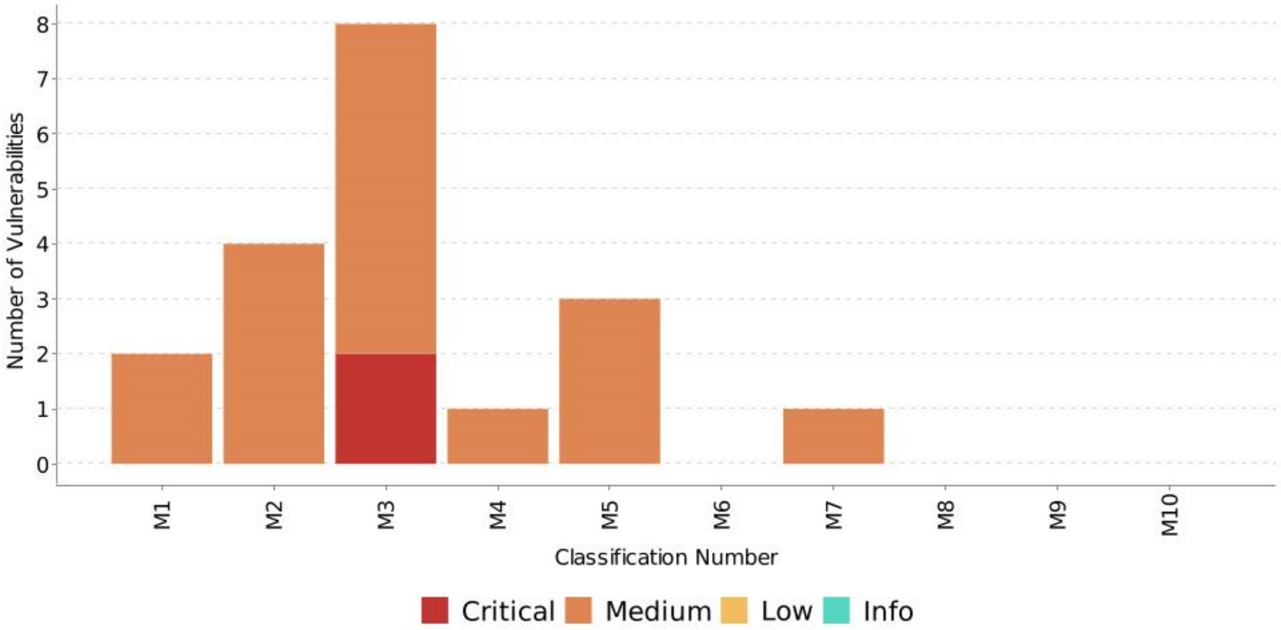


Language Statistics



Language	Status	Duration	Lines of Code	Number of Vulnerabilities				
				Critical	Medium	Low	Info	Total
JVM languages	completed	1:55:07	236 160	3	133	358	400	894
Config files	completed	0:00:05	58 277	0	23	0	0	23
PL/SQL	completed	0:00:02	162	0	0	0	0	0

Classification by OWASP Mobile Top 10 2016



	Vulnerabilities					Occurrences				
	Critical	Medium	Low	Info	Total	Critical	Medium	Low	Info	Total
M1	0	2	0	0	2	0	2	0	0	2
M2	0	4	0	0	4	0	29	0	0	29
M3	2	6	0	0	8	2	32	0	0	34
M4	0	1	0	0	1	0	3	0	0	3
M5	0	3	0	0	3	0	26	0	0	26
M6	0	0	0	0	0	0	0	0	0	0
M7	0	1	0	0	1	0	1	0	0	1
M8	0	0	0	0	0	0	0	0	0	0
M9	0	0	0	0	0	0	0	0	0	0
M10	0	0	0	0	0	0	0	0	0	0

Vulnerability List

Vulnerabilities are displayed accordingly to export settings: **37** selected

Actual: **37** of **917**

M1	Improper Platform Usage

Medium vulnerabilities 2*

Receiver without permissions	Android	1
🟧 androidx/appcompat/app/j0.java:47		Not processed
Broadcast sender without permissions	Android	1
🟧 acr/browser/lightning/n0/n.java:39		Not processed

M2	Insecure Data Storage

Medium vulnerabilities 29*

External storage usage	Android	3
🟧 acr/browser/lightning/settings/fragment/BookmarkSettingsFragment.java:63		Not processed
🟧 acr/browser/lightning/settings/fragment/BookmarkSettingsFragment.java:66		Not processed
🟧 androidx/core/content/FileProvider.java:94		Not processed
Unsafe SSL/TLS versions	Java	1
🟧 org/jsoup/helper/HttpConnection$Response.java:303		Not processed
HTTP usage	Java	2

* Rejected vulnerabilities are not taken into account

M2	Insecure Data Storage

Medium vulnerabilities

HTTP usage	Java	
🟧 acr/browser/lightning/reading/HtmlFetcher.java:51		Not processed
🟧 acr/browser/lightning/reading/HtmlFetcher.java:359		Not processed
HTTP usage	Config files	23
🟧 META-INF/CHANGES:791		Not processed
🟧 META-INF/CHANGES:902		Not processed
🟧 META-INF/CHANGES:905		Not processed
🟧 META-INF/CHANGES:927		Not processed
🟧 META-INF/CHANGES:930		Not processed
🟧 META-INF/CHANGES:982		Not processed
🟧 META-INF/CHANGES:985		Not processed
🟧 META-INF/CHANGES:989		Not processed
🟧 META-INF/CHANGES:995		Not processed
🟧 META-INF/CHANGES:999		Not processed
🟧 META-INF/CHANGES:1003		Not processed
🟧 META-INF/CHANGES:1006		Not processed
🟧 META-INF/CHANGES:1009		Not processed
🟧 META-INF/CHANGES:1012		Not processed
🟧 META-INF/CHANGES:1015		Not processed
🟧 META-INF/CHANGES:1019		Not processed
🟧 META-INF/CHANGES:1022		Not processed
🟧 META-INF/CHANGES:1025		Not processed
🟧 META-INF/CHANGES:1044		Not processed

* Rejected vulnerabilities are not taken into account

M2 Insecure Data Storage

Medium vulnerabilities

HTTP usage	Config files	
🟧 META-INF/CHANGES:1063		Not processed
🟧 META-INF/README.md:6		Not processed
🟧 META-INF/README.md:19		Not processed
🟧 META-INF/README.md:22		Not processed

M3 Insecure Communication

Critical vulnerabilities 2*

Unsafe custom SSL implementation (trivial)	Android	1
🟥 org/jsoup/helper/e.java:8#22		Not processed
No hostname verification	Android	1
🟥 org/jsoup/helper/d.java:14		Not processed

Medium vulnerabilities 32*

External storage usage	Android	3
🟧 acr/browser/lightning/settings/fragment/BookmarkSettingsFragment.java:63		Not processed
🟧 acr/browser/lightning/settings/fragment/BookmarkSettingsFragment.java:66		Not processed
🟧 androidx/core/content/FileProvider.java:94		Not processed
Unsafe custom SSL implementation (non-trivial)	Android	1
🟧 acr/browser/lightning/reading/f.java:9#31		Not processed
Unsafe SSL/TLS versions	Java	1

* Rejected vulnerabilities are not taken into account

M3	Insecure Communication

Medium vulnerabilities

Unsafe SSL/TLS versions	Java	
🟧 org/jsoup/helper/HttpConnection$Response.java:303		Not processed
HTTP usage	Java	2
🟧 acr/browser/lightning/reading/HtmlFetcher.java:51		Not processed
🟧 acr/browser/lightning/reading/HtmlFetcher.java:359		Not processed
HTTP usage	Config files	23
🟧 META-INF/CHANGES:791		Not processed
🟧 META-INF/CHANGES:902		Not processed
🟧 META-INF/CHANGES:905		Not processed
🟧 META-INF/CHANGES:927		Not processed
🟧 META-INF/CHANGES:930		Not processed
🟧 META-INF/CHANGES:982		Not processed
🟧 META-INF/CHANGES:985		Not processed
🟧 META-INF/CHANGES:989		Not processed
🟧 META-INF/CHANGES:995		Not processed
🟧 META-INF/CHANGES:999		Not processed
🟧 META-INF/CHANGES:1003		Not processed
🟧 META-INF/CHANGES:1006		Not processed
🟧 META-INF/CHANGES:1009		Not processed
🟧 META-INF/CHANGES:1012		Not processed
🟧 META-INF/CHANGES:1015		Not processed
🟧 META-INF/CHANGES:1019		Not processed

* Rejected vulnerabilities are not taken into account

M3	Insecure Communication

Medium vulnerabilities

HTTP usage	Config files	
🟧 META-INF/CHANGES:1022		Not processed
🟧 META-INF/CHANGES:1025		Not processed
🟧 META-INF/CHANGES:1044		Not processed
🟧 META-INF/CHANGES:1063		Not processed
🟧 META-INF/README.md:6		Not processed
🟧 META-INF/README.md:19		Not processed
🟧 META-INF/README.md:22		Not processed
Location data usage	Android	2
🟧 androidx/appcompat/app/c1.java:25		Not processed
🟧 androidx/appcompat/app/c1.java:26		Not processed

M4	Insecure Authentication

Medium vulnerabilities 3*

External storage usage	Android	3
🟧 acr/browser/lightning/settings/fragment/BookmarkSettingsFragment.java:63		Not processed
🟧 acr/browser/lightning/settings/fragment/BookmarkSettingsFragment.java:66		Not processed
🟧 androidx/core/content/FileProvider.java:94		Not processed

* Rejected vulnerabilities are not taken into account

M4	**Insecure Authentication**

M5	**Insufficient Cryptography**

Medium vulnerabilities 26*

Unsafe SSL/TLS versions	Java	1
🟧 org/jsoup/helper/HttpConnection$Response.java:303		Not processed
HTTP usage	Java	2
🟧 acr/browser/lightning/reading/HtmlFetcher.java:51		Not processed
🟧 acr/browser/lightning/reading/HtmlFetcher.java:359		Not processed
HTTP usage	Config files	23
🟧 META-INF/CHANGES:791		Not processed
🟧 META-INF/CHANGES:902		Not processed
🟧 META-INF/CHANGES:905		Not processed
🟧 META-INF/CHANGES:927		Not processed
🟧 META-INF/CHANGES:930		Not processed
🟧 META-INF/CHANGES:982		Not processed
🟧 META-INF/CHANGES:985		Not processed
🟧 META-INF/CHANGES:989		Not processed
🟧 META-INF/CHANGES:995		Not processed
🟧 META-INF/CHANGES:999		Not processed
🟧 META-INF/CHANGES:1003		Not processed

* Rejected vulnerabilities are not taken into account

M5	Insufficient Cryptography

Medium vulnerabilities

HTTP usage	Config files	
🟧 META-INF/CHANGES:1006		Not processed
🟧 META-INF/CHANGES:1009		Not processed
🟧 META-INF/CHANGES:1012		Not processed
🟧 META-INF/CHANGES:1015		Not processed
🟧 META-INF/CHANGES:1019		Not processed
🟧 META-INF/CHANGES:1022		Not processed
🟧 META-INF/CHANGES:1025		Not processed
🟧 META-INF/CHANGES:1044		Not processed
🟧 META-INF/CHANGES:1063		Not processed
🟧 META-INF/README.md:6		Not processed
🟧 META-INF/README.md:19		Not processed
🟧 META-INF/README.md:22		Not processed

M7	Poor Code Quality

Medium vulnerabilities 1*

JavaScript execution allowed in WebView	Android	1
🟧 acr/browser/lightning/view/c0.java:834		Not processed

* Rejected vulnerabilities are not taken into account

M7	Poor Code Quality

* Rejected vulnerabilities are not taken into account

Detailed Results

No hostname verification (Android)

Description

The verify() method defined in a class that implements the HostnameVerifier interface always returns true. When establishing a secure connection the application does not check the authenticity of the domain. This can lead to a loss of data confidentiality.
Within the establishing of a protected connection (handshake) server sends its public key and certificate, which are a cryptographic proof that the public key belongs to the owner of the server, to the client. The authenticity of certificates is provided by Certification Authority.
The correspondence between the certificate and the public key transferred to the client within the handshake does not guarantee the security of the connection. The client must make sure that the public key and the certificate come from the domain to which the connection is requested. Such check is not provided at the level of SSL / TLS protocol . In its absence at the application level, the attacker can violate the connection confidentiality by redirecting the user traffic through the attacker's server and presenting a certificate that is valid for the attacker's domain.
To check whether the requested domain matches the certificate received in response Android applications have the HostnameVerifier interface. The developer can use one of the existing implementations of this interface (StrictHostnameVerifier, X509HostnameVerifier) or create his/her own. It is assumed that the verify() method of the class that implements HostnameVerifier returns true if the connection to this host is allowed within the current connection and return false otherwise. The verify() method always returning true means that the application trusts all owners of all valid certificates, regardless of the domain for which they were obtained.
A possible attack scenario:

1. The attacker enters the user's WLAN and redirects user's traffic through the attacker's server (for example, via a DNS cache poisoning attack).
2. The user initiates a connection to https://safeserver.example.com.via an SSL / TLS protocol.
3. Instead of the https://safeserver.example.com.public key an attacker sends the application his/her own public key and a valid certificate issued by the certification authority for the https://hackedserver.example.com.domain.
4. The app makes sure that the resulting certificate is valid (for https://hackedserver.example.com., ignoring the fact that the certificate has been issued not for the resulting domain for which the connection was originally requested.

Insecure Communication vulnerabilities take the third place in the "OWASP Mobile Top 10 2016" mobile application vulnerabilities ranking.

Example

In the following example, the verify() method of the BlindHostnameVerifier is redefined so that any domain is recognized as valid:

```
private class BlindHostnameVerifier implements HostnameVerifier {
   @Override
   public boolean verify(final String s, final SSLSession sslSession) {
        return true;
    }
}
```

Recommendations

- Check the authenticity of the certificate each time when establishing a connection via an SSL / TLS protocol.
- Do not use AllowAllHostnameVerifier (the HostnameVerifier implementation that allows connections with any domain), except for debugging during the application development.
- Use one of the standard classes that implement HostnameVerifier (e.g., BrowserCompatHostnameVerifier, StrictHostnameVerifier, X509HostnameVerifier) to authenticate the domain.
- Use your own implementation of HostnameVerifier only when absolutely necessary.

Links

1. Security with HTTPS and SSL - developer.android.com
2. OWASP Mobile Top 10 2014: Insufficient Transport Layer Protection
3. OWASP Mobile Top 10 2016-M3-Insecure Communication
4. CWE-297: Improper Validation of Certificate with Host Mismatch
5. Fixing Hostname Verification - Will Sargent / tersesystems.com
6. CWE-295

Vulnerability Entries

org/jsoup/helper/d.java:14

Level Critical

```
11
12    @Override
13    public boolean verify(final String s, final SSLSession sslSession) {

14        return true;

15    }
16 }
```

Unsafe custom SSL implementation (trivial) (Android)

Description

The class that implements the X509TrustManager or SSLSocketFactory interface may contain trivial methods. This can lead to a loss of confidentiality of the data transferred an SSL / TSL protocol.
Within the establishing of a protected connection (handshake) server sends its public key and certificate, which are a cryptographic proof that the public key belongs to the owner of the server, to the client. The authenticity of certificates is provided by Certification Authority.
If it is necessary for the application functioning to take the certificate that is not signed by a recognized certification authority (for example, a self-signed certificate), then developers create a class that implements the X509TrustManager or SSLSocketFactory interface. Often methods of this class are trivial (accepting all certificates), which makes the application vulnerable to man in the middle (MitM) attacks. By providing a valid self-signed certificate an attacker can violate the confidentiality of the connection and get an access to valuable data.
Even if the methods of the redefined class are not trivial, their implementation is likely to be contain mistakes leading to the same consequences.
A possible attack scenario:

1. The attacker enters the user's WLAN and redirects user's traffic through the attacker's server (for example, via DNS cache poisoning).
2. The user initiates a connection to https://safeserver.example.com.via an SSL / TLS protocol through the application.
3. The attacker sends his/her own public key and a self-signed certificate generated by him/herself for the https://safeserver.example.com.domain to the application .
4. The application verifies that the received certificate matches the requested domain, ignoring the fact that the received certificate is self-signed.

Insecure Communication vulnerabilities take the third place in the "OWASP Mobile Top 10 2016" mobile application vulnerabilities ranking

Example

In the following example, the BlindX509TrustManager class that does not validate a certificate is defined:

```
private class BlindX509TrustManager implements X509TrustManager
{
   @Override
   public void checkClientTrusted(final X509Certificate[] array, final String s) throws
CertificateException {
   }

   @Override
   public void checkServerTrusted(final X509Certificate[] array, final String s) throws
CertificateException {
   }

   @Override
   public X509Certificate[] getAcceptedIssuers() {
      return new X509Certificate[0];
   }
}
```

Recommendations

- Check the validity of the certificate each time when establishing a connection via a SSL / TLS protocol.
- Use standard implementations of X509TrustManager.
- If accepting self-signed certificates is necessary, generate your own X509TrustManager implementation using KeyStore. Explicitly specify the certificates that should be taken and reject all others.

Links

1. Security with HTTPS and SSL - developer.android.com
2. OWASP Mobile Top 10 2014-M3: Insufficient Transport Layer Protection
3. OWASP Mobile Top 10 2016-M3-Insecure Communication
4. CWE-295: Improper Certificate Validation
5. HTTPS with Client Certificates on Android - Rich Freedman / chariotsolutions.com
6. SSL on Android: The Basics - Mark Murphy / commonsware.com
7. Using self signed certificates in Android - Taneli Korri
8. Trusting all certificates using HttpClient over HTTPS - emmby, Bostone / stackoverflow.com

Vulnerability Entries

org/jsoup/helper/e.java:8#22

Level Critical

```
5
6 class e implements X509TrustManager
7 {


8    e() {
9        super();
10    }
11
12 ...
13
14    @Override
15    public X509Certificate[] getAcceptedIssuers() {
16        return null;


17    }
18 }
```

Broadcast sender without permissions (Android)

Description

The application sends a broadcast message without specifying the appropriate permissions for the receiving application.
Messages sent this way are available to any receiver. Valuable data contained in the message may be compromised.
Android uses broadcast messages for system events such as battery level, network connections, incoming calls, time zone changes, data connection status, incoming SMS messages or phone calls. Broadcast messages are also used to notify listeners of system or application events.
Broadcast messages make the application more open. By passing events using messages, you open the components of your applications to third-party applications, and third-party developers respond to events without having to modify your original application.
Senders of intents can make sure that the recipient has permission, specifying a non-zero permission when calling the method. Only the application with this permission will receive the intent. If data in broadcast intents can be sensitive, you should consider applying permissions to ensure that malicious applications can not register to receive these messages without the appropriate permissions. In these circumstances, you can also consider calling the recipient directly without performing the mailing.
Improper Platform Usage vulnerabilities take the first place in the “OWASP Mobile Top 10 2016” mobile application vulnerabilities ranking.

Example

In the following example, a broadcast message is sent insecurely:
context.sendBroadcast(intent);
A secure alternative:
context.sendBroadcast(intent, "permission.ALLOW_INCOMING");

Recommendations

- Explicitly specify permissions that the broadcast messages receiver must have.
- Avoid using broadcast messages for valuable data transmission.

Links

1. OWASP Top 10 2013-A5-Security Misconfiguration
2. Mobile Top 10 2016-M1-Improper Platform Usage
3. Context - developer.android.com
4. CWE-941

Vulnerability Entries

acr/browser/lightning/n0/n.java:39

Level Medium

```
36    this.c = c;
37 }
38


39 private void f(final Activity p0) {


40    //
41    // This method could not be decompiled.
42    //
```

External storage usage (Android)

M2

M3

Description

The application writes data to the external storage.
Files written to external storage device are readable by all applications and can be changed when the user connects the device to a computer in a USB drive mode. Besides, files stored in external storage will remain there even after the application is deleted. This can lead to a valuable data confidentialitv loss.

Example

In the following example, the application calls a method, which returns a reference to an external storage device:

```
private void WriteToFile(String what_to_write) {
  try {
    File root = Environment.getExternalStorageDirectory();
    if(root.canWrite()) {
      File dir = new File(root + "write_to_the_SDcard");
      File datafile = new File(dir, number + ".extension");
      FileWriter datawriter = new FileWriter(datafile);
      BufferedWriter out = new BufferedWriter(datawriter);
      out.write(what_to_write);
      out.close();
    }
    }
}
```

Secure alternative (internal memory used; files created by the application and stored in it are available only to this application):

```
String FILENAME = "hello_file";
String string = "hello world!";
FileOutputStream fos = openFileOutput(FILENAME, Context.MODE_PRIVATE);
fos.write(string.getBytes());
fos.close();
```

Recommendations

- Store files in the internal memory, then they will only be available to the application that stored them.
- Use SQLite database: override SQLiteOpenHelper class and OnCreate() method.

Links

1. OWASP: Insecure Storage
2. Storage Options - developer.android.com
3. CWE-250: Execution with Unnecessary Privileges
4. CWE-921: Storage of Sensitive Data in a Mechanism without Access Control

Vulnerability Entries

acr/browser/lightning/settings/fragment/BookmarkSettingsFragment.java:63

Level Medium

```
60 final StringBuilder sb = new StringBuilder();
61 sb.append(string);
62 sb.append(": ");

63 sb.append(Environment.getExternalStorageDirectory());

64 r.b((CharSequence)sb.toString());
65 if (file == null) {
66     file = new File(Environment.getExternalStorageDirectory().toString());
```

acr/browser/lightning/settings/fragment/BookmarkSettingsFragment.java:66

Level Medium

```
63 sb.append(Environment.getExternalStorageDirectory());
64 r.b((CharSequence)sb.toString());
65 if (file == null) {

66     file = new File(Environment.getExternalStorageDirectory().toString());

67 }
68 Label_0115: {
69     try {
```

androidx/core/content/FileProvider.java:94

Level Medium

```
91     parent = context.getCacheDir();
92 }
93 else if ("external-path".equals(name)) {

94     parent = Environment.getExternalStorageDirectory();

95 }
```

```
else if ("external-files-path".equals(name)) {
    final File[] b = androidx.core.content.b.b(context, null);
```

JavaScript execution allowed in WebView (Android)

Description

The setJavaScriptEnabled(true) method, allowing the execution of JavaScript code, is called for an instance of the WebView class (designed to download and display HTML pages). This behavior is prohibited by default. A setJavaScriptEnabled(true) call can contribute to the success of cross-site scripting (XSS) attacks. Among the possible consequences of such an attack there is the loss of confidentiality of application data, such as user session data.

Cross-site scripting is one of the most common types of attacks on web applications. XSS-attacks take the seventh place in the “OWASP Top 10 2017” list of ten most significant web application vulnerabilities. In the mobile application vulnerabilities “OWASP Top 10 Mobile Risks 2014” ranking, client side injection attacks, which include some XSS-attack, take the seventh place.

The main phase of any XSS-attack is an imperceptible for the victim execution of a malicious code in the context of the vulnerable application. For this purpose, the functionality of the client application (browser) is used that allows to automatically execute scripts embedded in web page code. In most cases, these malicious scripts are implemented in JavaScript. Thus, the setJavaScriptEnabled(true) call is one of the necessary conditions for an XSS attack.

Consequences of an XSS attack vary from violations of application functionality to complete loss of user data confidentiality. The malicious code can steal cookies during the XSS-attack, which gives an attacker the ability to make requests to the server on behalf of the user.

OWASP proposes the following classification of XSS-attacks.

Server type XSS attacks occur when data from an untrusted source is included into the response returned by the server. The source of such data can be both user input and server database (where it had been previously injected by an attacker who exploited vulnerabilities in the server-side application).

Client type XSS attacks occur when the raw data from the user input contains code that changes the Document Object Model (DOM) of the web page received from the server. The source of such data can be both the DOM and the data received from the server (e.g., in response to an AJAX-request).

The typical server type attack scenario:

1. Unchecked data, usually from the HTTP-request, get into the server part of the application.
2. The server dynamically generates a web page that contains the unchecked data.
3. In the process of generating a web page, server does not prevent the inclusion of an executable code that can be executed in the client (browser), such as JavaScript code language, HTML-tags, HTML-attributes, Flash, ActiveX, etc., in the page code.
4. The victim's client application displays the web page that contains the malicious code embedded using the help of data from an untrusted source.
5. Since malicious code is embedded in the web page coming from the known server, the client part of the application (browser) executes it with the rights specified for the application.
6. This violates same-origin policy, according to which the code from the one source must not get an access to resources from another source.

Client type attacks are executed in a similar way with the only difference that the malicious code is injected during the phase of the client application work with the document object model received from the server.

In the context of Android applications attention must be payed vulnerabilities that lead to DOM-based XSS attacks (a subset of client type XSS attacks). The difference between this type of attack and traditional XSS attacks is that in the case of DOM-based XSS malicious code is not sent to the server. Therefore, the server means of protection, such as escaping special characters in the output of the server application, in this case are useless.

Example

In the following example, the code enables the ability to perform JavaScript code in the WebView class instance and loads the web page from the URL obtained from an Intent object:

```
WebView webview = (WebView)findViewById(R.id.webview);
webview.getSettings().setJavaScriptEnabled(true);
String url = this.getIntent().getExtras().getString("url");
webview.loadUrl(url);
```

If the value of the url variable contains the javascript: prefix, then both this code and JavaScript-code will be executed in the context of the loaded web page. In this example, the malicious code is stored outside the application, is loaded during the processing of the loaded web page, and is included into the dynamic context of the application.

Recommendations

- Do not call setJavaScriptEnabled(true), if the JavaScript-code execution is not necessary for the application functioning.
- Make sure that the parameters used by the WebView instance to display web pages can be loaded only from trusted sources (possibly only local).
- Implement a validation mechanism that escapes potentially dangerous characters or character sequences in the parameters passed to an instance of the WebView class.

Links

1. OWASP: Cross-site Scripting (XSS)
2. CWE-79: Improper Neutralization of Input During Web Page Generation
3. OWASP: Types of Cross-Site Scripting
4. OWASP: XSS Prevention Cheat Sheet
5. OWASP: DOM-based XSS Prevention Cheat Sheet
6. OWASP Top 10 2013-A3-Cross-Site Scripting (XSS)
7. OWASP Mobile Top 10 2014-M7: Client Side Injection
8. OWASP Mobile Top 10 2016-M7-Poor Code Quality

Vulnerability Entries

acr/browser/lightning/view/c0.java:834

Level Medium

```
831     throw null;
832 }
833 if (o12.u()) {

834     settings.setJavaScriptEnabled(true);

835     settings.setJavaScriptCanOpenWindowsAutomatically(true);
836 }
837 else {
```

Location data usage (Android)

Description

The application uses the information about the device location received from GPS. When working incorrect with such information, the application can compromise user privacy. Applications that process GPS data must take precautions to prevent violation of the confidentiality of this information.

Example

In the following example, the application requests notification about the device location change: lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 1000, 0, locationListener);

Recommendations

- Follow the instructions for secure data storage in order to prevent violations of the confidentiality of information about the device location.

Links

1. Location Strategies - developer.android.com
2. OWASP Top 10 2013-A6-Sensitive Data Exposure
3. CWE-250: Execution with Unnecessary Privileges

Vulnerability Entries

androidx/appcompat/app/c1.java:25

Level Medium

```
22
23 private Location a(final String s) {
24     try {
25         if (this.b.isProviderEnabled(s)) {
26             return this.b.getLastKnownLocation(s);
27         }
28         return null;
```

androidx/appcompat/app/c1.java:26

Level Medium

```
23 private Location a(final String s) {
24     try {
25         if (this.b.isProviderEnabled(s)) {
26             return this.b.getLastKnownLocation(s);
27         }
28         return null;
29     }
```

Receiver without permissions (Android)

Description

The application registers a broadcast receiver without defining the requirements for the sender permissions.
The application will receive broadcast messages from any source, including malicious ones. This may lead to an application compromise.
BroadcastReceiver processes asynchronous requests initiated by Intent.
By default recipients are exported and can be called by any other application. If your BroadcastReceiver is intended for use by other applications, you can apply permissions to

recipients using the <receiver> element in the application manifest. This will prevent sending intents from applications without proper permissions to BroadcastReceiver.
Improper Platform Usage vulnerabilities take the third place in the “OWASP Mobile Top 10 2016” mobile application vulnerabilities ranking. This category includes vulnerabilities related to platform’s permissions, misuse of TouchID, the Keychain and other security control elements that are part of the mobile operating system.

Example

In the following example, the application registers a broadcast receiver that does not check sender permissions:
context.registerReceiver(broadcastReceiver, intentFilter);
Secure alternative:
context.registerReceiver(broadcastReceiver, intentFilter, "permission.ALLOW_BROADCAST", handler);

Recommendations

- Explicitly specify the permissions that the broadcast messages sender must have. Do not accept messages from senders that do not have these permissions.
- Implement validation mechanism for data contained in the received messages.

Links

1. OWASP Top 10 2013-A5-Security Misconfiguration
2. Context - developer.android.com
3. CWE-925

Vulnerability Entries

androidx/appcompat/app/j0.java:47

Level Medium

```
44          if (this.a == null) {
45              this.a = new i0(this);
46          }


47          this.b.e.registerReceiver(this.a, b);


48        }
49      }
50 }
```

Unsafe custom SSL implementation (non-trivial) (Android)

Description

The class that implements the X509TrustManager or SSLSocketFactory interface can contain trivial methods. This can lead to a loss of confidentiality of the data transferred via SSL / TSL protocol.
Within the establishing of a protected connection (handshake) server sends its public key and certificate, which are a cryptographic proof that the public key belongs to the owner of the server, to the client. The authenticity of certificates is provided by Certification Authority.
If it is necessary for the application functioning to take the certificate that is not signed by a recognized certification authority (for example, a self-signed certificate), then developers create a class that implements the X509TrustManager or SSLSocketFactory interface. Often methods of this class are trivial (accepting all certificates), which makes the application vulnerable to man in the middle (MitM) attacks. By providing a valid self-signed certificate an attacker can violate the confidentiality of the connection and get an access to valuable data.
Even if the methods of the redefined class are not trivial, their implementation is likely to be contain mistakes leading to the same consequences.
A possible attack scenario:

1. The attacker enters the user's WLAN and redirects user's traffic through the attacker's server (for example, via DNS cache poisoning).
2. The user initiates a connection to https://safeserver.example.com.via an SSL / TLS protocol through the application.
3. The attacker sends his/her own public key and a self-signed certificate generated by him/herself for the https://safeserver.example.com.domain to the application .
4. The application verifies that the received certificate matches the requested domain, ignoring the fact that the received certificate is self-signed.

Insufficient Transport Layer Protection vulnerabilities take the third place in the “OWASP Mobile Top 2014” mobile platforms vulnerabilities ranking.

Example

In the following example, the BlindX509TrustManager class that does not validate a certificate is defined:

```
private class BlindX509TrustManager implements X509TrustManager
{
  @Override
  public void checkClientTrusted(final X509Certificate[] array, final String s) throws
CertificateException {
  }

  @Override
  public void checkServerTrusted(final X509Certificate[] array, final String s) throws
CertificateException {
  }

  @Override
  public X509Certificate[] getAcceptedIssuers() {
    return new X509Certificate[0];
  }
}
```

Recommendations

- Check the validity of the certificate each time when establishing a connection via a SSL / TLS protocol.
- Use standard implementations of X509TrustManager.
- If accepting self-signed certificates is necessary, generate your own X509TrustManager implementation using KeyStore. Explicitly specify the certificates that should be taken and reject all others.

Links

1. Security with HTTPS and SSL - developer.android.com
2. OWASP Mobile Top 10 2014-M3: Insufficient Transport Layer Protection
3. OWASP Mobile Top 10 2016-M3-Insecure Communication
4. CWE-295: Improper Certificate Validation
5. HTTPS with Client Certificates on Android - Rich Freedman / chariotsolutions.com
6. SSL on Android: The Basics - Mark Murphy / commonsware.com
7. Using self signed certificates in Android - Taneli Korri
8. Trusting all certificates using HttpClient over HTTPS - emmby, Bostone / stackoverflow.com

Vulnerability Entries

acr/browser/lightning/reading/f.java:9#31

Level Medium

```
6
7 class f implements X509TrustManager
8 {

9    f(final e e) {
10       super();
11   }
12
13 ...
14
15   @Override
16   public X509Certificate[] getAcceptedIssuers() {
17       return null;

18   }
19 }
```

HTTP usage (Config files)

M2

M3

Description

Using HTTP rather than HTTPS allows “the man in the middle” attack. This can lead to a complete confidentiality loss of the transferred data.
Using HTTPS, which is based on HTTP and SSL / TLS, helps to protect the transferred data against unauthorized access and modification. It is recommended to use HTTPS for all cases of data transfer between the client and the server, in particular, for the login page and all pages that require authentication.

Example

In the following example, the application stores an address with HTTP protocol:
url = "http://example.com"

Recommendations

- Use only secure protocols (e.g., HTTPS) for the confidential data transfer between the client and the server.

Links

1. OWASP Top 10 2017-A3-Sensitive Data Exposure
2. Transport Layer Protection Cheat Sheet – OWASP
3. Web Security: Why You Should Always Use HTTPS – Mike Shema / Mashable
4. CWE-319: Cleartext Transmission of Sensitive Information
5. CWE CATEGORY: OWASP Top Ten 2017 Category A6 - Security Misconfiguration

Vulnerability Entries

META-INF/CHANGES:791

Level Medium

```
788
789 *** Release 1.6.0 [2011-Jun-13]
790  * HTML5 conformant parser. Complete reimplementation of HTML tokenisation and parsing, to implement the

791    http://whatwg.org/html spec. This ensures jsoup parses HTML identically to current modern browsers.


792
793  * When parsing files from disk, files are loaded via memory mapping, to increase parse speed.
794
```

META-INF/CHANGES:902

Level Medium

```
899    doc.select("iframe").remove();
900
901 * Fixed issue in Entities when unescaping &#36; ("$")

902   <http://github.com/jhy/jsoup/issues/issue/34>


903
904 * Added restricted XHTML output entity option
905   <http://github.com/jhy/jsoup/issues/issue/35>
```

META-INF/CHANGES:905

Level Medium

```
902    <http://github.com/jhy/jsoup/issues/issue/34>
903
904  * Added restricted XHTML output entity option

905    <http://github.com/jhy/jsoup/issues/issue/35>
```

```
906
907 *** Release 1.3.2 [2010-Aug-30]
908  * Treat HTTP headers as case insensitive in Jsoup.Connection. Improves compatibility
for HTTP responses.
```

META-INF/CHANGES:927

Level Medium

```
924 * Further speed optimisations for parsing and output generation.
925
926 * Fixed support for case-sensitive HTML escape entities.

927   <http://github.com/jhy/jsoup/issues/issue/31>

928
929 * Fixed issue when parsing tags with keyless attributes.
930   <http://github.com/jhy/jsoup/issues/issue/32>
```

META-INF/CHANGES:930

Level Medium

```
927    <http://github.com/jhy/jsoup/issues/issue/31>
928
929  * Fixed issue when parsing tags with keyless attributes.

930    <http://github.com/jhy/jsoup/issues/issue/32>

931
932 *** Release 1.2.3 [2010-Aug-04]
933  * Added support for automatic input character set detection and decoding. Jsoup now
automatically detects the encoding
```

META-INF/CHANGES:982

Level Medium

```
979 * Added :contains(text) selector, to search for elements containing the specified text
980
981 * Added :has(selector) pseudo-selector


982     <http://github.com/jhy/jsoup/issues/issue/20>


983
984 * Added Element#parents and Elements#parents to retrieve an element's ancestor chain
985     <http://github.com/jhy/jsoup/issues/issue/20>
```

META-INF/CHANGES:985

Level Medium

```
982     <http://github.com/jhy/jsoup/issues/issue/20>
983
984 * Added Element#parents and Elements#parents to retrieve an element's ancestor chain


985     <http://github.com/jhy/jsoup/issues/issue/20>


986
987 * Fixes an issue where appending / prepending rows to a table (or  to similar implicit
988     element structures) would create a redundant wrapping elements
```

META-INF/CHANGES:989

Level Medium

```
986
987 * Fixes an issue where appending / prepending rows to a table (or  to similar implicit
988     element structures) would create a redundant wrapping elements


989      <http://github.com/jhy/jsoup/issues/issue/21>


990
991 * Improved implicit close tag heuristic detection when parsing malformed HTML
992
```

META-INF/CHANGES:995

Level Medium

```
992
993 * Fixes an issue where text content after a script (or other data-node) was
994    incorrectly added to the data node.

995     <http://github.com/jhy/jsoup/issues/issue/22>

996
997 * Fixes an issue where text order was incorrect when parsing pre-document
998   HTML.
```

META-INF/CHANGES:999

Level Medium

```
996
997  * Fixes an issue where text order was incorrect when parsing pre-document
998    HTML.

999     <http://github.com/jhy/jsoup/issues/issue/23>

1000
1001 *** Release 1.1.1 [2010-Jun-08]
1002  * Added selector support for :eq, :lt, and :gt
```

META-INF/CHANGES:1003

Level Medium

```
1000
1001 *** Release 1.1.1 [2010-Jun-08]
1002  * Added selector support for :eq, :lt, and :gt

1003    <http://github.com/jhy/jsoup/issues/issue/16>

1004
1005  * Added TextNode#text and TextNode#text(String)
1006    <http://github.com/jhy/jsoup/issues/issue/18>
```

META-INF/CHANGES:1006

Level Medium

```
1003    <http://github.com/jhy/jsoup/issues/issue/16>
1004
1005 * Added TextNode#text and TextNode#text(String)

1006    <http://github.com/jhy/jsoup/issues/issue/18>

1007
1008 * Throw exception if trying to parse non-text content
1009    <http://github.com/jhy/jsoup/issues/issue/17>
```

META-INF/CHANGES:1009

Level Medium

```
1006    <http://github.com/jhy/jsoup/issues/issue/18>
1007
1008 * Throw exception if trying to parse non-text content

1009    <http://github.com/jhy/jsoup/issues/issue/17>

1010
1011 * Added Node#remove and Node#replaceWith
1012    <http://github.com/jhy/jsoup/issues/issue/19>
```

META-INF/CHANGES:1012

Level Medium

```
1009    <http://github.com/jhy/jsoup/issues/issue/17>
1010
1011 * Added Node#remove and Node#replaceWith

1012    <http://github.com/jhy/jsoup/issues/issue/19>
```

```
1013
1014 * Allow _ and - in CSS ID selectors (per CSS spec).
1015     <http://github.com/jhy/jsoup/issues/issue/10>
```

META-INF/CHANGES:1015

Level Medium

```
1012     <http://github.com/jhy/jsoup/issues/issue/19>
1013
1014 * Allow _ and - in CSS ID selectors (per CSS spec).

1015     <http://github.com/jhy/jsoup/issues/issue/10>

1016
1017 * Relative links are resolved to absolute when cleaning, to normalize
1018   output and to verify safe protocol. (Were previously discarded.)
```

META-INF/CHANGES:1019

Level Medium

```
1016
1017 * Relative links are resolved to absolute when cleaning, to normalize
1018   output and to verify safe protocol. (Were previously discarded.)

1019      <http://github.com/jhy/jsoup/issues/issue/12>

1020
1021 * Allow combinators at start of selector query, for query refinements
1022      <http://github.com/jhy/jsoup/issues/issue/13>
```

META-INF/CHANGES:1022

Level Medium

```
1019     <http://github.com/jhy/jsoup/issues/issue/12>
1020
1021 * Allow combinators at start of selector query, for query refinements

1022      <http://github.com/jhy/jsoup/issues/issue/13>

1023
1024 * Added Element#val() and #val(String) methods, for form values
1025      <http://github.com/jhy/jsoup/issues/issue/14>
```

META-INF/CHANGES:1025

Level Medium

```
1022      <http://github.com/jhy/jsoup/issues/issue/13>
1023
1024 * Added Element#val() and #val(String) methods, for form values

1025      <http://github.com/jhy/jsoup/issues/issue/14>

1026
1027 * Changed textarea contents to parse as TextNodes, not DataNodes,
1028    so contents visible to text() (and val(), as treated as form input)
```

META-INF/CHANGES:1044

Level Medium

```
1041     with attribute.
1042
1043  * Fixed assertion error when cleaning HTML with empty attribute

1044      <http://github.com/jhy/jsoup/issues/issue/7>

1045
1046 *** Release 0.2.2 (2010-Feb-07)
1047  * jsoup packages are now available in the Maven central repository.
```

META-INF/CHANGES:1063

Level Medium

```
1060
1061 *** Release 0.1.2 (2010-Feb-02)
1062  * Fixed unrecognised tag handler to be more permissive
1063    <http://github.com/jhy/jsoup/issues/issue/1>
1064
1065
1066 *** Release 0.1.1 (2010-Jan-31)
```

META-INF/README.md:6

Level Medium

```
3 **jsoup** is a Java library for working with real-world HTML. It provides a very convenient API for extracting and manipulating data, using the best of DOM, CSS, and jquery-like methods.
4
5
6 **jsoup** implements the [WHATWG HTML5](http://whatwg.org/html) specification, and parses HTML to the same DOM as modern browsers do.
7
8 * scrape and [parse](https://jsoup.org/cookbook/input/parse-document-from-string) HTML from a URL, file, or string
9 * find and [extract data](https://jsoup.org/cookbook/extracting-data/selector-syntax), using DOM traversal or CSS selectors
```

META-INF/README.md:19

Level Medium

```
16 See [**jsoup.org**](https://jsoup.org/) for downloads and the full [API documentation](https://jsoup.org/apidocs/).
17
18 ## Example
19 Fetch the [Wikipedia](http://en.wikipedia.org/wiki/Main_Page) homepage, parse it to a [DOM](https://developer.mozilla.org/en-
US/docs/Web/API/Document_Object_Model/Introduction), and select the headlin...


20
21 ```java
22 Document doc = Jsoup.connect("http://en.wikipedia.org/").get();
```

META-INF/README.md:22

Level Medium

```
19 Fetch the [Wikipedia](http://en.wikipedia.org/wiki/Main_Page) homepage, parse it to a
[DOM](https://developer.mozilla.org/en-
US/docs/Web/API/Document_Object_Model/Introduction), and select the headlin...
20
21 ```java


22 Document doc = Jsoup.connect("http://en.wikipedia.org/").get();


23 log(doc.title());
24 Elements newsHeadlines = doc.select("#mp-itn b a");
25 for (Element headline : newsHeadlines) {
```

HTTP usage (Java)

M2

M3

Description

Using HTTP rather than HTTPS allows “the man in the middle” attack. This can lead to a complete confidentiality loss of the transferred data.
Using HTTPS, which is based on HTTP and SSL / TLS, helps to protect the transferred data against unauthorized access and modification. It is recommended to use HTTPS for all cases of data transfer between the client and the server, in particular, for the login page and all pages that require authentication.

Example

In the following example, the application initiates a HTTP connection:

```
URL exampleUrl = new URL("http://www.example.org/");
URLConnection exampleConn = exampleUrl.openConnection();
```

Recommendations

- Use only secure protocols (e.g., HTTPS) for the confidential data transfer between the client and the server.

Links

1. OWASP Top 10 2017-A3-Sensitive Data Exposure
2. Transport Layer Protection Cheat Sheet – OWASP
3. Web Security: Why You Should Always Use HTTPS – Mike Shema / Mashable
4. CWE-319: Cleartext Transmission of Sensitive Information
5. CWE CATEGORY: OWASP Top Ten 2017 Category A6 - Security Misconfiguration

Vulnerability Entries

acr/browser/lightning/reading/HtmlFetcher.java:51

Level Medium

```
48
49 public HtmlFetcher() {
50     super();


51     this.a = "http://jetsli.de/crawler";


52     final StringBuilder a = d.a.a.a.a.a("Mozilla/5.0 (compatible; Jetslide; +");
53     a.append(this.a);
54     a.append(')');
```

acr/browser/lightning/reading/HtmlFetcher.java:359

Level Medium

```
356 }
357
358 private HttpURLConnection b(final String spec, final int n, final boolean b) {


359     final HttpURLConnection httpURLConnection = (HttpURLConnection)new
URL(spec).openConnection(Proxy.NO_PROXY);


360     httpURLConnection.setRequestProperty("User-Agent", this.b);
361     httpURLConnection.setRequestProperty("Accept", this.e);
362     if (b) {
```

Trace

"http://jetsli.de/crawler"

acr/browser/lightning/reading/HtmlFetcher.java:51

```
48
49 public HtmlFetcher() {
50     super();


51     this.a = "http://jetsli.de/crawler";
```

```
final StringBuilder a = d.a.a.a.a.a("Mozilla/5.0 (compatible; Jetslide; +");
a.append(this.a);
a.append(')');
```

URL.openConnection()

acr/browser/lightning/reading/HtmlFetcher.java:359

```
356 }
357
358 private HttpURLConnection b(final String spec, final int n, final boolean b) {

359     final HttpURLConnection httpURLConnection = (HttpURLConnection)new
URL(spec).openConnection(Proxy.NO_PROXY);

360     httpURLConnection.setRequestProperty("User-Agent", this.b);
361     httpURLConnection.setRequestProperty("Accept", this.e);
362     if (b) {
```

Unsafe SSL/TLS versions (Java)

M2

M3

Description

SSL connection uses insecure settings. The established connection is insecure and can cause a compromise of valuable data.
The SSLv2, SSLv23, SSLv3, TLSv1.0 and TLSv1.1 protocols contain several flaws that make them insecure, so they should not be used to transmit sensitive data.
The Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols provide a protection mechanism to ensure the authenticity, confidentiality and integrity of data transmitted between a client and web server. Both TLS and SSL have undergone revisions resulting in periodic version updates. Each new revision was designed to address the security weaknesses discovered in the previous versions. Use of an insecure version of TLS/SSL will weaken the strength of the data protection and could allow an attacker to compromise, steal, or modify sensitive information.
Weak versions of TLS/SSL may exhibit one or more of the following properties: * No protection against man-in-the-middle attacks * Same key used for authentication and

encryption * Weak message authentication control * No protection against TCP connection closing
The presence of these properties may allow an attacker to intercept, modify, or tamper with sensitive data.

Example

In the following example, the application installs insecure TLS settings:
SSLContext context = SSLContext.getInstance("TLSv1");

Recommendations

- Use the last version of the SSL/TLS protocol.

Links

1. OWASP Top 10 2017-A3-Sensitive Data Exposure
2. CWE CATEGORY: OWASP Top Ten 2017 Category A6 - Security Misconfiguration
3. Vulnerability Summary for CVE-2014-3566
4. CWE-757: Selection of Less-Secure Algorithm During Negotiation

Vulnerability Entries

org/jsoup/helper/HttpConnection$Response.java:303

Level Medium

```
300 if (HttpConnection$Response.o == null) {
301     final e e = new e();
302     try {

303         final SSLContext instance = SSLContext.getInstance("SSL");

304         instance.init(null, new TrustManager[] { e }, new SecureRandom());
305         HttpConnection$Response.o = instance.getSocketFactory();
306     }
```

WAF Configuration Guide

HTTP usage

Description

Using HTTP rather than HTTPS allows “the man in the middle” attack. This can lead to a complete confidentiality loss of the transferred data.
Using HTTPS, which is based on HTTP and SSL / TLS, helps to protect the transferred data against unauthorized access and modification. It is recommended to use HTTPS for all cases of data transfer between the client and the server, in particular, for the login page and all pages that require authentication.

Vulnerability Entries

1. META-INF/CHANGES:791
2. META-INF/CHANGES:902
3. META-INF/CHANGES:905
4. META-INF/CHANGES:927
5. META-INF/CHANGES:930
6. META-INF/CHANGES:982
7. META-INF/CHANGES:985
8. META-INF/CHANGES:989
9. META-INF/CHANGES:995
10. META-INF/CHANGES:999
11. META-INF/CHANGES:1003
12. META-INF/CHANGES:1006
13. META-INF/CHANGES:1009
14. META-INF/CHANGES:1012
15. META-INF/CHANGES:1015
16. META-INF/CHANGES:1019
17. META-INF/CHANGES:1022
18. META-INF/CHANGES:1025
19. META-INF/CHANGES:1044
20. META-INF/CHANGES:1063
21. META-INF/README.md:6
22. META-INF/README.md:19
23. META-INF/README.md:22

HTTP usage

Description

Using HTTP rather than HTTPS allows “the man in the middle” attack. This can lead to a complete confidentiality loss of the transferred data.
Using HTTPS, which is based on HTTP and SSL / TLS, helps to protect the transferred data against unauthorized access and modification. It is recommended to use HTTPS for all cases of data transfer between the client and the server, in particular, for the login page and all pages that require authentication.

Vulnerability Entries

1. acr/browser/lightning/reading/HtmlFetcher.java:51
2. acr/browser/lightning/reading/HtmlFetcher.java:359

Path manipulation

Description

Using data from an untrusted source when working with the file system may give an attacker access to important system files.
By manipulating variables that reference files with <)>> sequences and its variations or by using absolute file paths, it may be possible to access arbitrary files and directories stored on file system including application source code or configuration and critical system files.

Vulnerability Entries

1. c/h/b/a.java:43
2. c/h/b/a.java:120

Resource injection

Description

Using data from an untrusted source to identify the resource allows an attacker to view or modify protected system resources.
The injection when working with resources (resource injection) occurs when an attacker can specify the identifier to access the system resources (for example, the port number for the network resource access). This allows him/her in particular to transfer valuable data to a third party server.
Injection vulnerabilities take the first place in the “OWASP Top 10 2017” web-application vulnerabilities ranking.

Vulnerability Entries

1. acr/browser/lightning/reading/g.java:76
2. acr/browser/lightning/reading/HtmlFetcher.java:252
3. acr/browser/lightning/reading/HtmlFetcher.java:359
4. acr/browser/lightning/view/c0.java:573
5. k/i1/g/c.java:146
6. k/i1/g/c.java:196
7. org/jsoup/helper/HttpConnection.java:36
8. org/jsoup/helper/HttpConnection.java:36
9. org/jsoup/helper/StringUtil.java:211

Scan Settings

1/1 2022-07-18 09:31:55

Source code https://play.google.com/store/apps/details?id=acr.browser.barebones&hl=en&gl=US

Select files for Analysis **/*

Languages

- ☑ ABAP
- ☑ Apex
- ☑ C#
- ☑ C/C++
- ☑ COBOL
- ☑ Config files
- ☑ Dart
- ☑ Delphi
- ☑ Go
- ☑ Groovy
- ☑ HTML5
- ☑ Java, Scala, Kotlin
- ☑ JavaScript
- ☑ LotusScript
- ☑ Objective-C
- ☑ Pascal
- ☑ PHP
- ☑ PL/SQL
- ☑ Python
- ☑ Perl
- ☑ Ruby
- ☑ Rust
- ☑ Solidity
- ☑ Swift
- ☑ T-SQL
- ☑ TypeScript
- ☑ VB.NET
- ☑ VBA
- ☑ VBScript
- ☑ Visual Basic 6
- ☑ Vyper
- ☑ 1C

Java/Scala/Kotlin Specific Settings

☑ Do not build project (project is already built)

C/C++ Specific Settings

□ Visual Studio project

JavaScript Specific Settings

□ Analyze standard libraries

General Analysis Settings

□ Analyze libraries and nested archives

□ Incremental analysis

Source Code Charset UTF-8

Filename Charset UTF-8

Rule Sets —

Export Settings

Project Information

☑ Vulnerability Dynamics

Scan History

○ Do not export scan history

● Export entire scan history

○ Export the latest scans ...

Vulnerability Classification

OWASP Mobile Top 10 2016

Scan Information

☑ Detected vulnerabilities chart

☑ Vulnerability types chart

☑ Language statistics

□ Analyzed Files Statistics

☑ Scan error information

☑ Scan Settings

Issues Filter

Severity Level

- ☑ Critical
- ☑ Medium
- □ Low
- □ Info

Vulnerability Types

- ☑ Vulnerabilities in standard libraries
- ☑ Vulnerabilities in .class files that could not be decompiled
- ☑ With a task created in Jira
- ☑ Vulnerabilities without WAF configuration guide

Languages

- ☑ ABAP
- ☑ Android
- ☑ Apex
- ☑ C#
- ☑ C/C++
- ☑ COBOL
- ☑ Config files
- ☑ Dart
- ☑ Delphi
- ☑ Go
- ☑ Groovy
- ☑ HTML5
- ☑ Java
- ☑ JavaScript
- ☑ Kotlin
- ☑ LotusScript
- ☑ Objective-C
- ☑ Pascal
- ☑ PHP
- ☑ PL/SQL
- ☑ Python
- ☑ Perl
- ☑ Ruby
- ☑ Rust
- ☑ Scala
- ☑ Solidity
- ☑ Swift
- ☑ T-SQL
- ☑ TypeScript
- ☑ VB.NET
- ☑ VBA
- ☑ VBScript
- ☑ Visual Basic 6
- ☑ Vyper
- ☑ 1C

Vulnerability List

Vulnerability Statuses

☑ Not processed

☑ Confirmed

□ Rejected

List of Vulnerability Entries

○ Do not export

● Export all entries

○ Export no more than entries ...

Detailed Results

Vulnerability Statuses

☑ Not processed

☑ Confirmed

□ Rejected

List of Vulnerability Entries

○ Do not export

● Export all entries

○ Export no more than entries ...

Source code

○ Do not export source code

○ Export entire vulnerable source code file

● Export context in the number of lines of code 3

Trace

○ Do not export trace items

● Export only the first and last items

○ Export all items

Additional information

☑ Vulnerability comment

☑ Jira information

WAF Configuration Guide

Guide for vulnerability statuses

☑ Not processed

☑ Confirmed

☐ Rejected

Guide for WAF

☑ Imperva SecureSphere

☑ ModSecurity

☑ F5

General Report Settings

☑ Report Export Settings

☑ Table of Contents

☐ Showing Statuses