**VA/PT**

**Portal**: <http://ctf.0x10.info/>

**APK**: <http://ctf.0x10.info/1.apk>

**Network Scanning:** 38.109.218.93

**NOTE THAT ALL ANSWERS SHOULD BE KEPT within respective folder.**

Note that this will leverage free-tier to avoid any cost. In case you need assistance, IRC Live Chat: <http://webchat.freenode.net/?channels=hackerearth&uio=d4>

Reference VA/PT Help Doc: [hck.re/VAPT](http://hck.re/VAPT) | You get a time to submit this by 9th 11:55 PM

To make it easier for you to answer, you may download this DOC and fill up the answers and images as required, keep the filled submission in respective directory and archive/upload it.

To fill up subjective answers, please make a copy of this document: File > Download As > DOCX or ODT

Do not forget to **archive** and **upload** after you’re done with maximum tasks.

**Network** / **Infrastructure**

Q1: Enumerate the **services** running in the box.

ANS:

Q2: Find the **fuzzy port** number running SSH

ANS:

**Web App**

ADD MORE **DIFFICULTY** and ADD MORE **INTERESTING** TASKS.

Q3: Find the web server’s **version** and the **operating system** of the box.

ANS: Apace 2.4.7 (Web Server) and Ubuntu (Operating System)

Q4: Describe the **major vulnerability** in the **home page** of the given website based on OWASP TOP 10?

ANS: The major vulnerability in the home page is [A2 Broken Authentication and Session Management](https://www.owasp.org/index.php/Top_10_2013-A2-Broken_Authentication_and_Session_Management) which is for the URI <http://ctf.0x10.info/1.apk>.

The response received contains the *max-age* parameter having really long expiry time (~1 year) which makes the session vulnerable:

*\_\_cfduid=d1a350b145da6610f86760847f3dc80501499452091; expires=Sat Jul 7 11:28:11 2018; path=/; domain=.0x10.info; max-age=31536459; httponly*

Q5: **[**Fuzzing**]** Reveal all the common files/artifacts which provide information on the site architecture and overall structure.

ANS: The site is vulnerable to [A4-Insecure Direct Object References](https://www.owasp.org/index.php/Top_10_2013-A4-Insecure_Direct_Object_References), [A5-Security Misconfiguration](https://www.owasp.org/index.php/Top_10_2013-A5-Security_Misconfiguration), [A6-Sensitive Data Exposure](https://www.owasp.org/index.php/Top_10_2013-A6-Sensitive_Data_Exposure) and [A7-Missing Function Level Access Control](https://www.owasp.org/index.php/Top_10_2013-A7-Missing_Function_Level_Access_Control) which makes it easy to retrieve the information and directory structure. The file structure hierarchy is as given below and it is possible to do directory browsing on each of these folders:

*ctf.0x10.info/images*

*ctf.0x10.info/js*

*ctf.0x10.info/uploads*

*ctf.0x10.info/common*

*ctf.0x10.info/includes*

*ctf.0x10.info/backup*

*ctf.0x10.info/css*

*ctf.0x10.info/test*

Q6: **[**Remote Code Execution**]** Find the **kernel version** of the victim operating system, and identify privilege escalation vulnerability associated with it.

ANS: The kernel version of the victim Ubuntu OS is 3.13

Q7: **[**Crypto/SQL Injection**]** After logging into the member area, perform the following functions -

1. Find the **master hash**

2. **Dump** all user's

3. Find the **email ID** and **password** of saved users

ANS:

Q8: Bypass the **member authentication** and explain how you did so?

ANS:

Q9: Bypass authentication in **admin area** and **redirect** site to hackerearth.com

ANS:

**Mobile** / **Infrastructure [**Level 1 / 2**]**

**Q10: [**Reverse Engineering**] Mobile App Analysis**

There are various applications and compiled components in given APK.

Review each and find the possible exploitable bugs (with patch or steps to rectify the loopholes)

\* vulnerabilities in db/sqlite components

\* information leakage

\* improper authentication validation

\* packet Inspection and possible tampering in network communication

\* improper storage Injection

**[Deliverables]**

1. Find the **cipher hash** hidden in the APK and ELF binary.
2. Find the **platform** and **architecture** binaries are compiled in.
3. Extract the resources available inside the APK and PE and list their names.

ANS:

1. By **cipher hash** I’m assuming you’re referring to the key which was used to sign the apk. The process followed was to decompile the apk using [apktool](https://ibotpeaches.github.io/Apktool/) with the command below:

*Java –jar apktool\_2.2.3.jar d 1.apk*

This generates the files and folders and the certificate info is in the *META-INF/CERT.RSA* file.

Now use openssl to make this into a .pem file:

*Openssl pkcs7 –in CERT.RSA –inform DER –print\_certs –out cert.pem*

It’s now easy to get the public key from this pem file:

*Openssl x509 –in cert.pem –pubkey –noout*

-----BEGIN PUBLIC KEY-----

MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDHzDPjonOa/Yn1YxXmRx45NA/U

zZhN+mOK+FhlGxsnyWRNx+yYFYfRET+6wl7o9E0HsFu+oL4Mh8JJaFcii8V5IP5y

46JuUSxIqiX6xInGDsO2fUBZFnVuWCQH5zPlrpFoxT2WbPVXhwO2Czbd2HrzK/PI

Zw67bPDExW6kwTbwbwIDAQAB

-----END PUBLIC KEY-----

2. We can use the apktool again and then look at the generated apktool.yaml file for this section info:

*sdkInfo:*

*minSdkVersion:* **'14'**

*targetSdkVersion:* **'14'**

Matching this version number can be done in [here](https://developer.android.com/guide/topics/manifest/uses-sdk-element.html#ApiLevels)

Looking at the page, we can see that this app has been built for Android 4.0/4.0.1/4.0.2 (Ice Cream Sandwich)

However, there’s no lib folder under the apk file, which means the Gradle may have spun a generic build that supports all architecture format (x86, ARM etc…)

3. All the resources under *1.apk* have been extracted under the files section. They are as listed below:

*AndroidManifest.xml*

*classes.dex*

*resources.arsc*

*META-INF/CERT.RSA*

*META-INF/CERT.SF*

*META-INF/MANIFEST.MF*

*res/drawable-hdpi-v4/ic\_launcher.png*

*res/drawable-xhdpi-v4/bulbicon.png*

*res/drawable-xhdpi-v4/shevirah.png*

*res/layout/bluetooth.xml*

*res/layout/commands.xml*

*res/layout/data.xml*

*res/layout/fuctionlist.xml*

*res/layout/getagents.xml*

*res/layout/getdata.xml*

*res/layout/main.xml*

*res/layout/nfc.xml*

*res/layout/remoteattack.xml*

*res/layout/socialattack.xml*

*com/bulbsecurity/framework/app/BluetoothScanner$1.smali*

*com/bulbsecurity/framework/app/BluetoothScanner$2.smali*

*com/bulbsecurity/framework/app/BluetoothScanner$mytask.smali*

*com/bulbsecurity/framework/app/BluetoothScanner.smali*

*com/bulbsecurity/framework/app/BuildConfig.smali*

*com/bulbsecurity/framework/app/CommandHandler.smali*

*com/bulbsecurity/framework/app/FrameworkAndroidApp.smali*

*com/bulbsecurity/framework/app/FrameworkAndroidAppActivity$1$1.smali*

*com/bulbsecurity/framework/app/FrameworkAndroidAppActivity$1.smali*

*com/bulbsecurity/framework/app/FrameworkAndroidAppActivity$2.smali*

*com/bulbsecurity/framework/app/FrameworkAndroidAppActivity.smali*

*com/bulbsecurity/framework/app/InternetPoll$1.smali*

*com/bulbsecurity/framework/app/InternetPoll.smali*

*com/bulbsecurity/framework/app/NFCSend$1.smali*

*com/bulbsecurity/framework/app/NFCSend.smali*

*com/bulbsecurity/framework/app/R$array.smali*

*com/bulbsecurity/framework/app/R$attr.smali*

*com/bulbsecurity/framework/app/R$drawable.smali*

*com/bulbsecurity/framework/app/R$id.smali*

*com/bulbsecurity/framework/app/R$layout.smali*

*com/bulbsecurity/framework/app/R$string.smali*

*com/bulbsecurity/framework/app/R.smali*

*com/bulbsecurity/framework/app/ServiceAutoStarter.smali*

*com/bulbsecurity/framework/app/SMSService.smali*

*com/bulbsecurity/framework/app/WebUploadService$1.smali*

*com/bulbsecurity/framework/app/WebUploadService.smali*

*com/bulbsecurity/framework/app/WebUploadService3$1.smali*

*com/bulbsecurity/framework/app/WebUploadService3.smali*

**Q11:** **[**Manual Bug Finding**]** Given a source code, find the well-known SSL vulnerability.

* Find heartbleed vulnerability in the source code.

ANS:

[The Heartbleed bug](http://heartbleed.com/) is not strictly a broken SSL/TLS vulnerability but more of an incorrect implementation of the famous ubiquitous protocol. In a code review of the source code, we should be doing a bounds check to make sure we’re crossing the 16 KB maximum heartbeat request size including one byte to signal that this is a **TLS1\_HB\_REQUEST**, and two bytes to denote the length of the payload data in the request.

Rather than doing a blind copy, we should check if the length does not cross the threshold (something like below a sample C snippet):

if (1 + 2 + payload + 16 > s->length)

        return 0; /\* silently discard \*/

As described in the link above, heartbleed was not affected for openSSL version 1.0.1g and above. You could have also mitigated it with using the compiler option -DOPENSSL\_NO\_HEARTBEATS if you’re building from scratch.

So as the idiom says, *“look before you leap”*, the heartbleed has taught programmers to *“check before you copy”*

**[**NOTE**]**

Write answer in subjective manner, feel free to add external link in case you want to present findings in more elaborative way.