Exploring Android Security: Penetration Testing, Vulnerabilities, and Exploitation Techniques

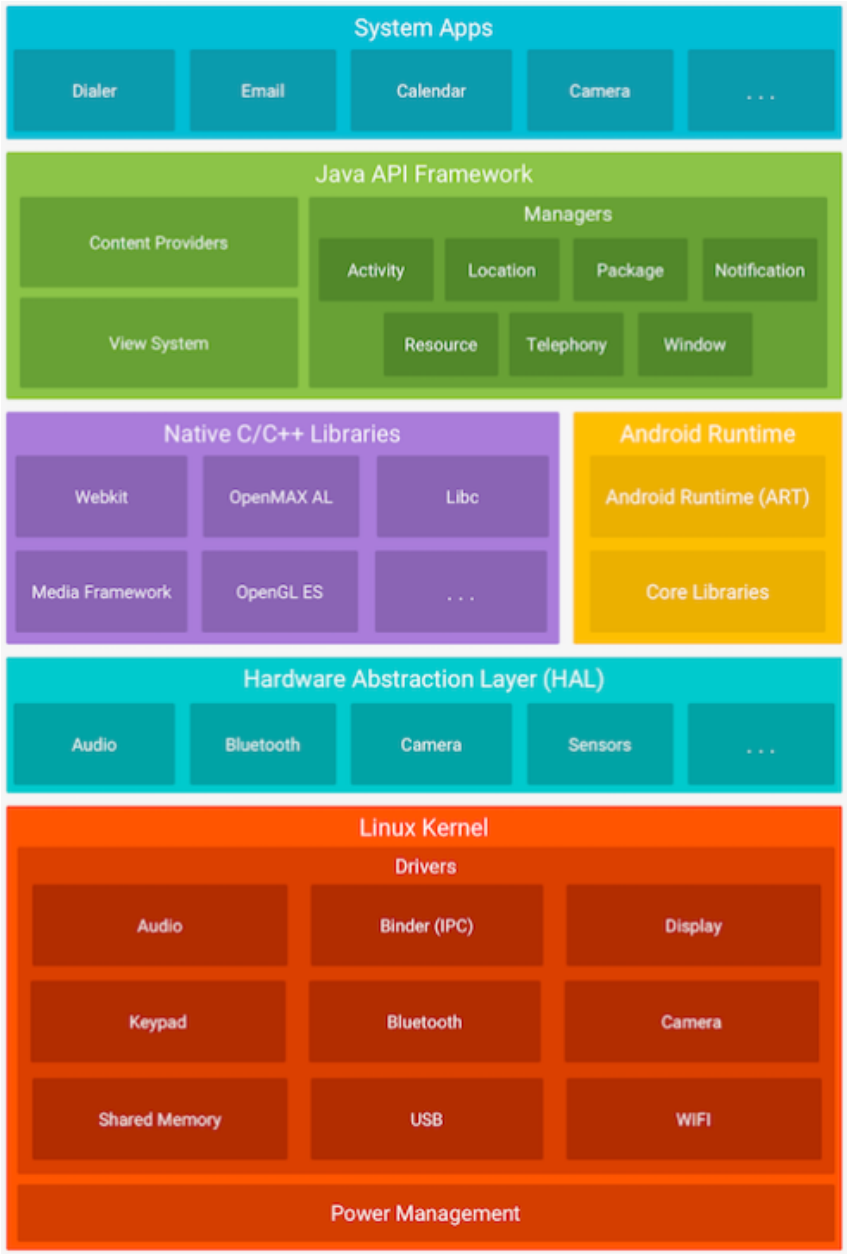
Rahul Meena(230832)
Chaitanya Bramhapurikar (230305)

Indian Institute of Technology Kanpur

April 16, 2025

Course: Undergraduate Project(CS496)
Professor: Prof. Subhajit Roy

Android Platform Overview and it's Security Features



Android Platform Overview and it's Security Features

- Linux Based, it uses a modified version of Linux kernel.
- Process Seperation, sandboxing: each process has a unique user id, storage and memory.
- Permission System - runtime and install-time permissions
- App Signing - All apps must be cryptographically signed
- Verified Boot - Cryptographic verification of OS integrity during startup
- Hardware-backed Keystore - Separate Hardware for cryptographic operations
- TLS by Default - Internet Communication only through TLS protocol
- ASLR, KASLR, PIE and DEP : standard protections against buffer-overflow exploits.
- Secure IPC Mechanisms - Framework for controlled inter-process/app communication

Testing Part 1: Android Apps and their attack surfaces

- Android uses sandboxing to isolate apps using separate UIDs and data directories, preventing unauthorized access by default.
- But apps often need to communicate internally and with outside. For this android provides secure ways and interfaces for IPC. This opens many attack surfaces.
- Weak security measure, bad code and other such factors opens these surfaces to exploits and attacks.
- **Identifying these attack surfaces is the first step in testing**
- To spread awareness about these potential risks, the Open Web Application Security Project(OWASP), a nonprofit foundation, releases it's widely recognized list of the 10 most critical mobile application security risks.

Owasp Top 10 Mobile security risks 2024

- Improper Credential Usage
- Inadequate Supply Chain Security
- Insecure Authentication/Authorization
- Insufficient Input/Output Validation
- Insecure Communication
- Inadequate Privacy Controls
- Insufficient Binary Protections
- Security Misconfiguration
- Insecure Data Storage
- Insufficient Cryptography

Common Android App Attack Surfaces

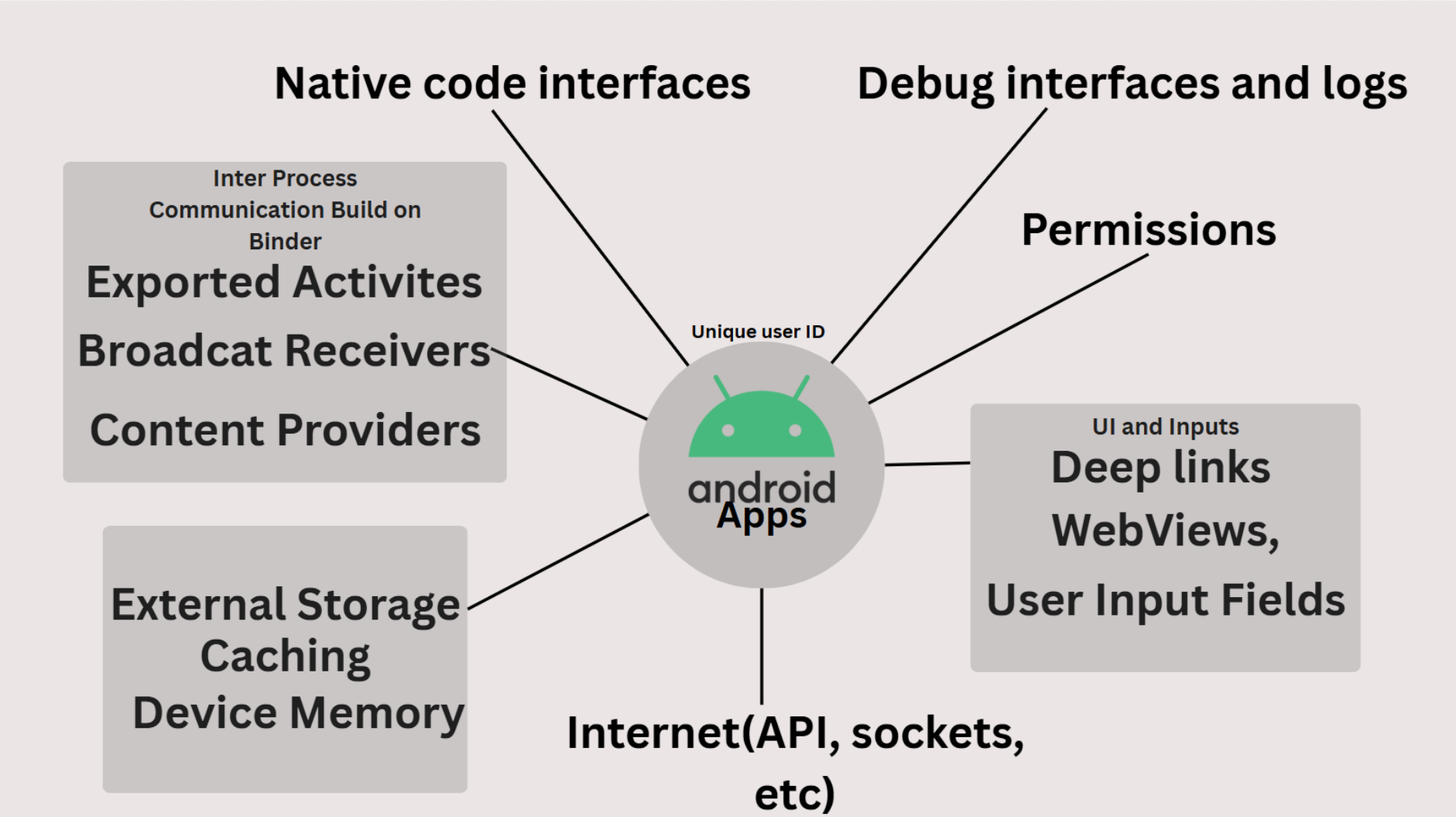


Figure: common android attack surfaces

Common Android Platform/OS Attack Surfaces

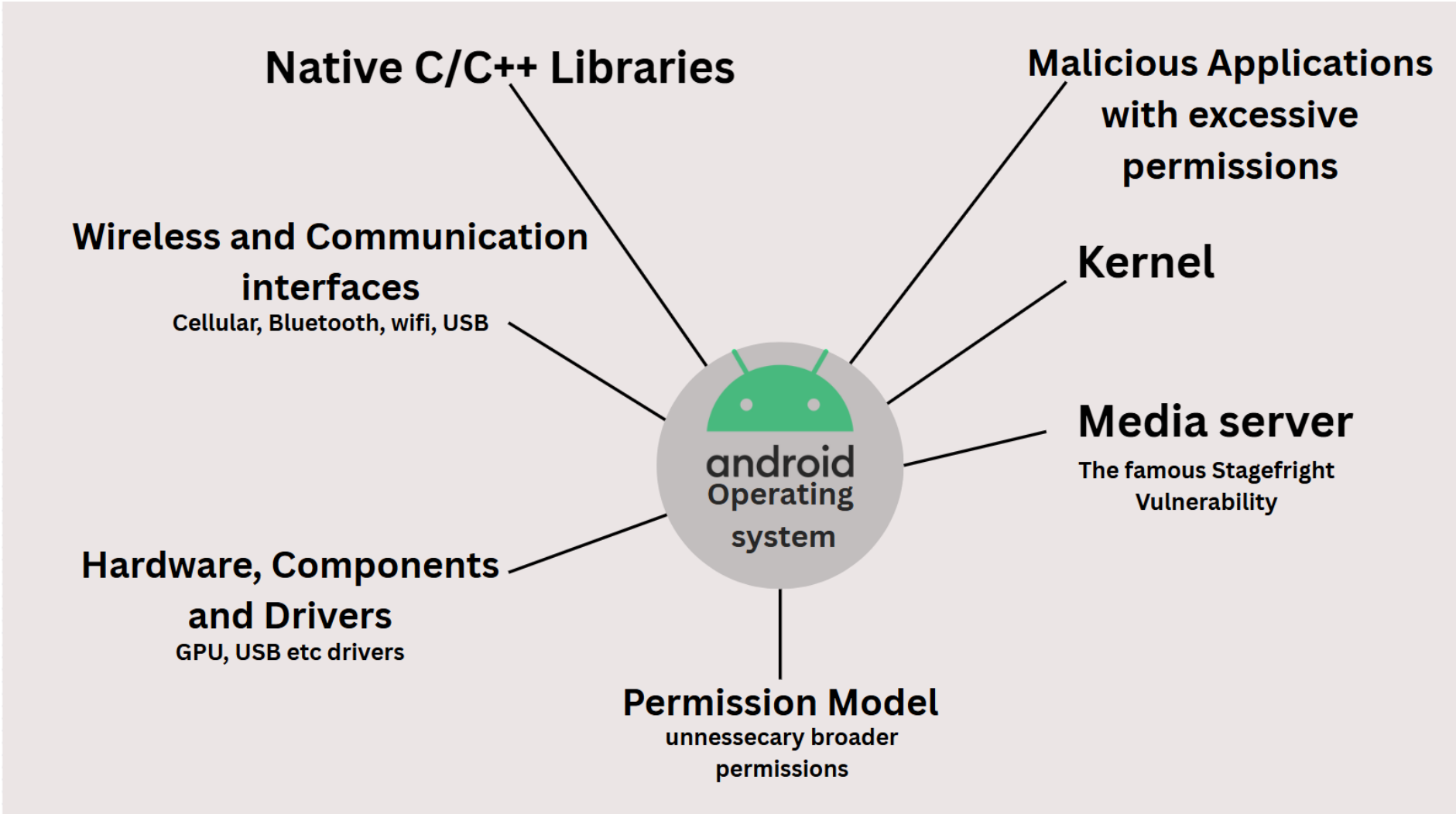


Figure: common android os attack surfaces

Attack Surfaces

How to reaveal more atack surfaces?

Techniques like Fuzzing : Feeding varied and often random data inputs like wifi packets, corrupted media files into an application's input interfaces, observer crashes and logs. Crashes often suggest a presence of potentially bad code or a condition the code did not took account of.

Testing Part 2: Tools and Techniques, Static Analysis

- Review the source code or binary without executing it.
- Ensure appropriate implementation of security controls
- Automatic scans and tools can catch bastic insecure practices like Hardcoded secrets, insecure cryptography, exported components, WebView vulnerabilities, Outdated Libraries with known CVEs
- the human tester can explore the code base with specific usage contexts in mind
- when the code is technically secure but logically flawed
- Tools : MobSF, APKTool, Jadx, AndroBugs, Radare2(for native binary analysis)

Static Analysis : Java/Kotlin Code Decompilation using jadx (Un-obfuscated code)

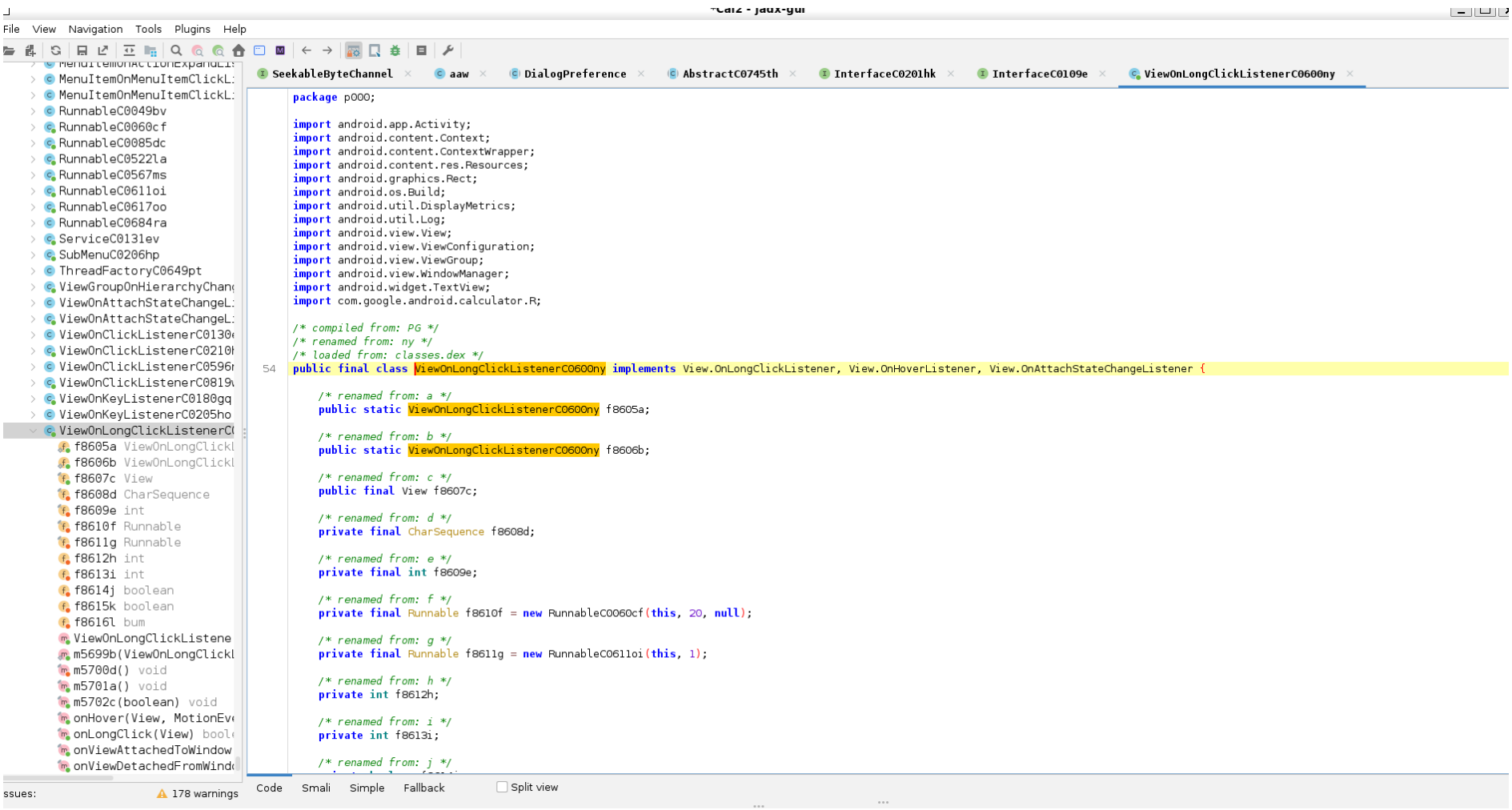


Figure: Using Jadx to decompile .dex files into java

Static Analysis: Native C/C++ Code Disassembly using radare2

Android provides Java Native Interface (JNI), which defines a way for Java code to interact with native code written in C/C++. As on other Linux-based operating systems, native code is packaged (compiled) into ELF dynamic libraries (*.so), which the Android app loads at runtime. Mishandling of these interfaces introduces traditional C/C++ vulnerabilities like Memory Corruption.

```
➜ FRIDA r2 demo
WARN: Relocs has not been applied. Please use `-e bin.relocs.apply=true` or `-e bin.cache=true` next time
[0x00001080]> aaa
INFO: Analyze all flags starting with sym. and entry0 (aa)
INFO: Analyze imports (af@@@i)
INFO: Analyze entrypoint (af@ entry0)
INFO: Analyze symbols (af@@@s)
INFO: Analyze all functions arguments/locals (afva@@@F)
INFO: Analyze function calls (aac)
INFO: Analyze len bytes of instructions for references (aar)
INFO: Finding and parsing C++ vtables (avrr)
INFO: Analyzing methods (af @@ method.*)
INFO: Recovering local variables (afva@@@F)
INFO: Type matching analysis for all functions (aaft)
INFO: Propagate noreturn information (aanr)
INFO: Use -AA or aaaa to perform additional experimental analysis
[0x00001080]> afl
0x00001030    1      6 sym.imp.puts
```

Testing Part 3: Tools and Techniques, Dynamic Analysis

- A typical android Application can have thousands of classes and methods. Looking through all of them through can be hard. Dynamic Analysis helps us narrow our search.
- Running the application in a test environment, observe behaviors under a variety of conditions. This can help identify, logic flaws, and unintended interactions, how an app interacts with inputs, systems, and networks while it's running.
- Trace function calls, change their behaviour and inputs at runtime. Can be used to bypass security checks like root checks, SSL Pinning, Login and Premium checks.
- Also helpful when dealing with obfuscated code, where it is hard to understand the logic of the code.

Dynamic Analysis: Objection

Demo : SSL Unpinning. SSL Pinning ensures an app only trusts a pre-defined server certificate (or its public key/hash) embedded within the app. Instead of relying solely on the device's trust store (system-level certificates), the app compares the server's certificate to the pinned value during handshakes. This features prevents anyone from intercepting the communication. A big problem while testing network realated vulnerabilities. Objection(a tool built on frida) can bypass this.

```
(new) PS D:\SEMESTER_4\TERMINAL\new\Scripts> objection -g "tech.httptoolkit.pinning_demo" explore
Using USB device `Android Emulator 5554`
Agent injected and responds ok!

     _   _         _   _
 ___| |_|_|___ ___| |_|_|___ ___
| . | . | | -_|  _|  _| | . |   |
|___|___| |___|___|_| |_|___|_|_|
      |___|(object)inject(ion) v1.11.0

     Runtime Mobile Exploration
        by: @leonjza from @sensepost

[tab] for command suggestions
tech.httptoolkit.pinning_demo on (google: 15) [usb] # android sslpinning disable
(agent) Custom TrustManager ready, overriding SSLContext.init()
(agent) Found okhttp3.CertificatePinner, overriding CertificatePinner.check()
(agent) Found okhttp3.CertificatePinner, overriding CertificatePinner.check$okhttp()
(agent) Found com.android.org.conscrypt.TrustManagerImpl, overriding TrustManagerImpl.verifyChain()
(agent) Found com.android.org.conscrypt.TrustManagerImpl, overriding TrustManagerImpl.checkTrustedRecursive()
(agent) Registering job 519211. Type: android-sslpinning-disable
tech.httptoolkit.pinning_demo on (google: 15) [usb] # (agent) [519211] Called SSLContext.init(), overriding TrustManager with empty one.
(agent) [519211] Called SSLContext.init(), overriding TrustManager with empty one.
(agent) [519211] Called (Android 7+) TrustManagerImpl.checkTrustedRecursive(), not throwing an exception.
(agent) [519211] Called SSLContext.init(), overriding TrustManager with empty one.
tech.httptoolkit.pinning_demo on (google: 15) [usb] #
```

Figure: Objection with frida gadget on non rooted device

Dynamic Analysis: Tracing function calls in Instagram

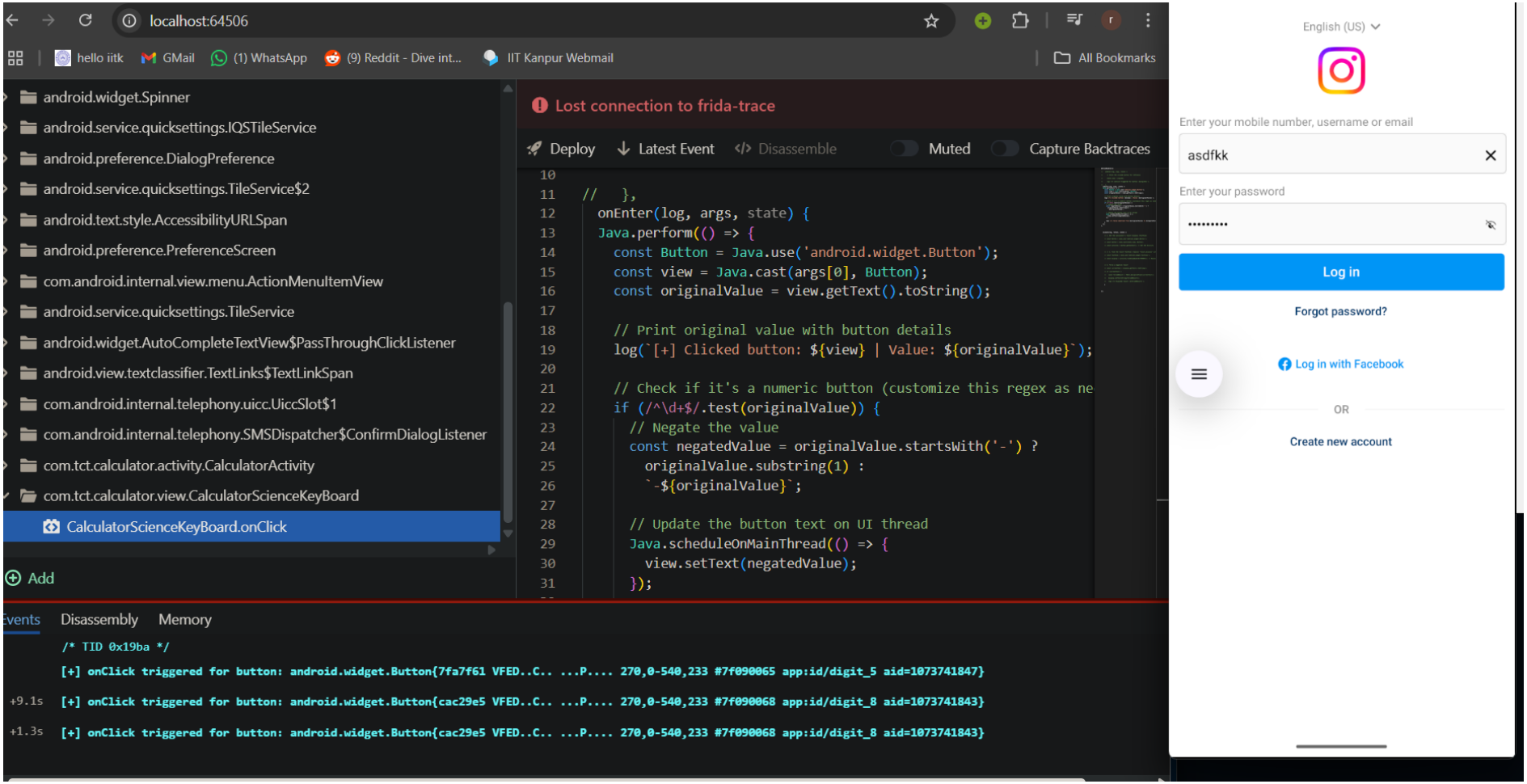


Figure: Frida Tracing Web UI

Attack Vectors and Exploitation

- Exploits are the specific actions taken to take advantage of vulnerabilities identified through those vectors.
- Vulnerabilities are harmless unless someone tries to exploit them, attackers use many pathways to gain access to the system to exploit these vulnerabilities, these pathways are called attack vectors.
- Vulnerabilities in itself are harmless unless an attacker tries to exploit them. Common Attack vectors are
  - Malicious Applications
  - Network-based Attacks(MITM, open wifis)
  - Phishing Attacks(fake emails, sms, login pages)
  - Native Code Vulnerabilities by Malformed input to native methods or media files.

Stagefright: A Short Case Study

- Arguably the most severe known Vulnerability discovered in android till date.
- Found by Zimperium zLabs (2015) in Android's libstagefright (media engine)
- Buffer overflow in MP3/MP4 parsing let to Remote Code Execution in Android 5/5.1.
- The media processor would process these files automatically without user interaction
- Zero-click attack possible—no user interaction needed, exploited just by sending a carefully crafted MMS or MP4 file.
- Lessons:
  - Memory safety is critical(C/C++ risks)
  - Automatic parsing and Weak input validations = high risk.
  - Avoid Unnecessary Permissions(the media enginer was running as a very elevated process)

Communication Between Computer and Network

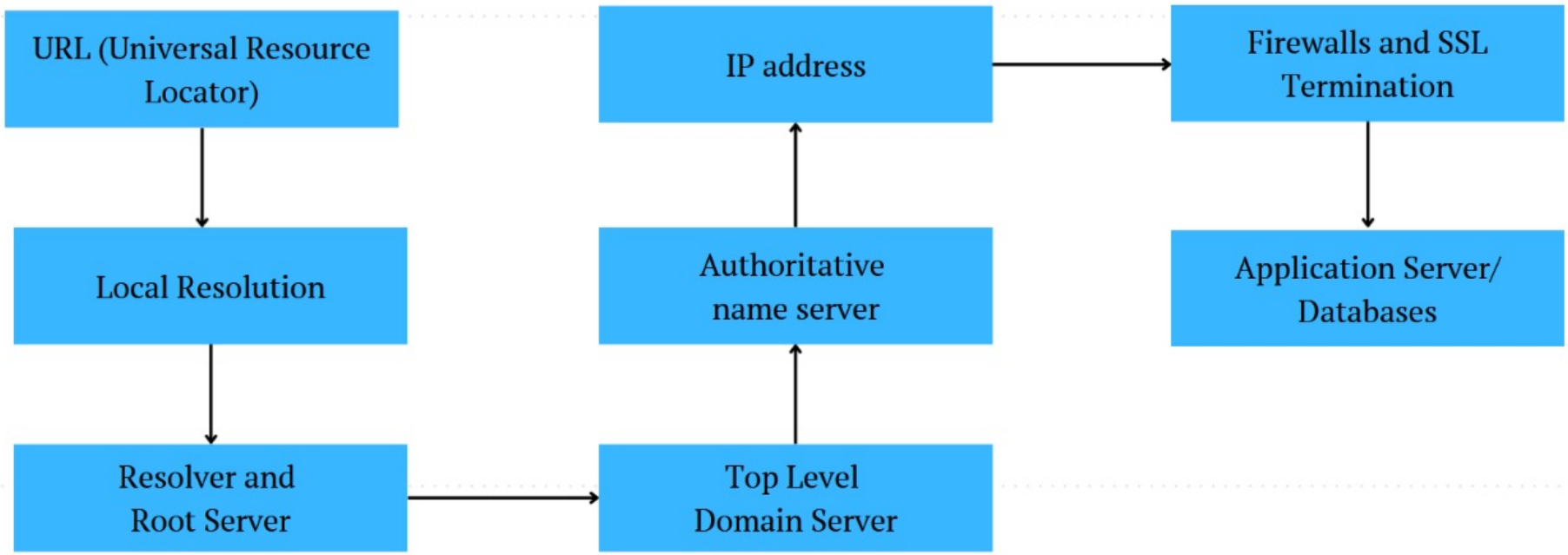


Figure: network flowchart

Examples of common vulnerabilities in network and tools to exploit them

- Weak Authentication
  - john the ripper
  - hydra
  - aircrack-ng
  - metasploit framework
  - Burpsuite
- Misconfigured Firewalls
  - wafw00f
  - nmap
  - metasploit framework
- Open Ports and Services
  - nmap
  - metasploit framework
  - masscan
  - nikto

Examples of common vulnerabilities in network and tools to exploit them

- Weak encryption
  - hashcat
  - john the ripper
  - wireshark
- Phishing and Social Engineering
  - Zphisher
  - metasploit

nmap

Nmap is short for Network Mapper. It is an open-source Linux command-line tool that is used to scan IP addresses and ports in a network.

- Basic Scan:

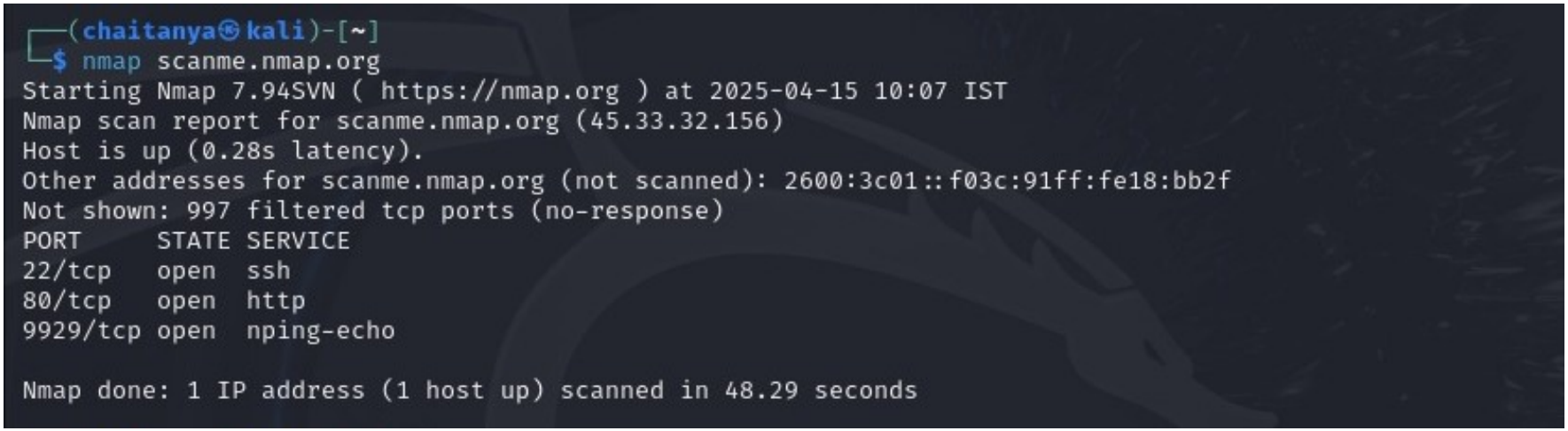


Figure: nmap basic scan

- Stealth Scan:
  - Stealth scanning is performed by sending an SYN packet and analyzing the response. If SYN/ACK is received, it means the port is open, and you can open a TCP connection.
  - However, a stealth scan never completes the 3-way handshake, which makes it hard for the target to determine the scanning system.

```
┌──(chaitanya㉿kali)-[~]
└─$ sudo nmap -sS scanme.nmap.org
[sudo] password for chaitanya:
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-04-15 10:17 IST
RTTVAR has grown to over 2.3 seconds, decreasing to 2.0
RTTVAR has grown to over 2.3 seconds, decreasing to 2.0
RTTVAR has grown to over 2.3 seconds, decreasing to 2.0
RTTVAR has grown to over 2.3 seconds, decreasing to 2.0
Nmap scan report for scanme.nmap.org (45.33.32.156)
Host is up (3.4s latency).
Other addresses for scanme.nmap.org (not scanned): 2600:3c01::f03c:91ff:fe18:bb2f
Not shown: 993 closed tcp ports (reset)
PORT      STATE    SERVICE
22/tcp    open     ssh
80/tcp    open     http
445/tcp   filtered microsoft-ds
514/tcp   filtered shell
8008/tcp  open     http
9929/tcp  open     nping-echo
31337/tcp open     Elite

Nmap done: 1 IP address (1 host up) scanned in 814.22 seconds
```

Figure: nmap stealth scan

- Version Scan:

```
┌──(chaitanya㉿kali)-[~]
└─$ nmap -sV scanme.nmap.org
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-04-15 10:23 IST
Nmap scan report for scanme.nmap.org (45.33.32.156)
Host is up (0.29s latency).
Other addresses for scanme.nmap.org (not scanned): 2600:3c01::f03c:91ff:fe18:bb2f
Not shown: 998 filtered tcp ports (no-response)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.13 (Ubuntu Linux; protocol 2.0)
80/tcp open  http    Apache httpd 2.4.7 ((Ubuntu))
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 56.46 seconds
```

Figure: nmap version scan

- Aggressive Scan:
  - Nmap has an aggressive mode that enables OS detection, version detection, script scanning, and traceroute.
  - Aggressive scans provide far better information than regular scans. However, an aggressive scan also sends out more probes, and it is more likely to be detected during security audits.

```
┌──(chaitanya㉿kali)-[~]
└─$ nmap -A scanme.nmap.org
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-04-15 10:35 IST
Nmap scan report for scanme.nmap.org (45.33.32.156)
Host is up (0.28s latency).
Other addresses for scanme.nmap.org (not scanned): 2600:3c01::f03c:91ff:fe18:bb2f
Not shown: 997 filtered tcp ports (no-response)
PORT     STATE SERVICE    VERSION
22/tcp   open  ssh        OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.13 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   1024 ac:00:a0:1a:82:ff:cc:55:99:dc:67:2b:34:97:6b:75 (DSA)
|   2048 20:3d:2d:44:62:2a:b0:5a:9d:b5:b3:05:14:c2:a6:b2 (RSA)
|   256 96:02:bb:5e:57:54:1c:4e:45:2f:56:4c:4a:24:b2:57 (ECDSA)
|_  256 33:fa:91:0f:e0:e1:7b:1f:6d:05:a2:b0:f1:54:41:56 (ED25519)
80/tcp   open  http       Apache httpd 2.4.7 ((Ubuntu))
|_http-title: Go ahead and ScanMe!
|_http-server-header: Apache/2.4.7 (Ubuntu)
|_http-favicon: Nmap Project
9929/tcp open  nping-echo Nping echo
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 66.82 seconds
```

Figure: nmap aggressive scan

spiderfoot

SpiderFoot is an open-source, cross-platform OSINT tool designed to automate the process of gathering intelligence about domains, IP addresses, email addresses, and other digital assets.

- spiderfoot web-ui:

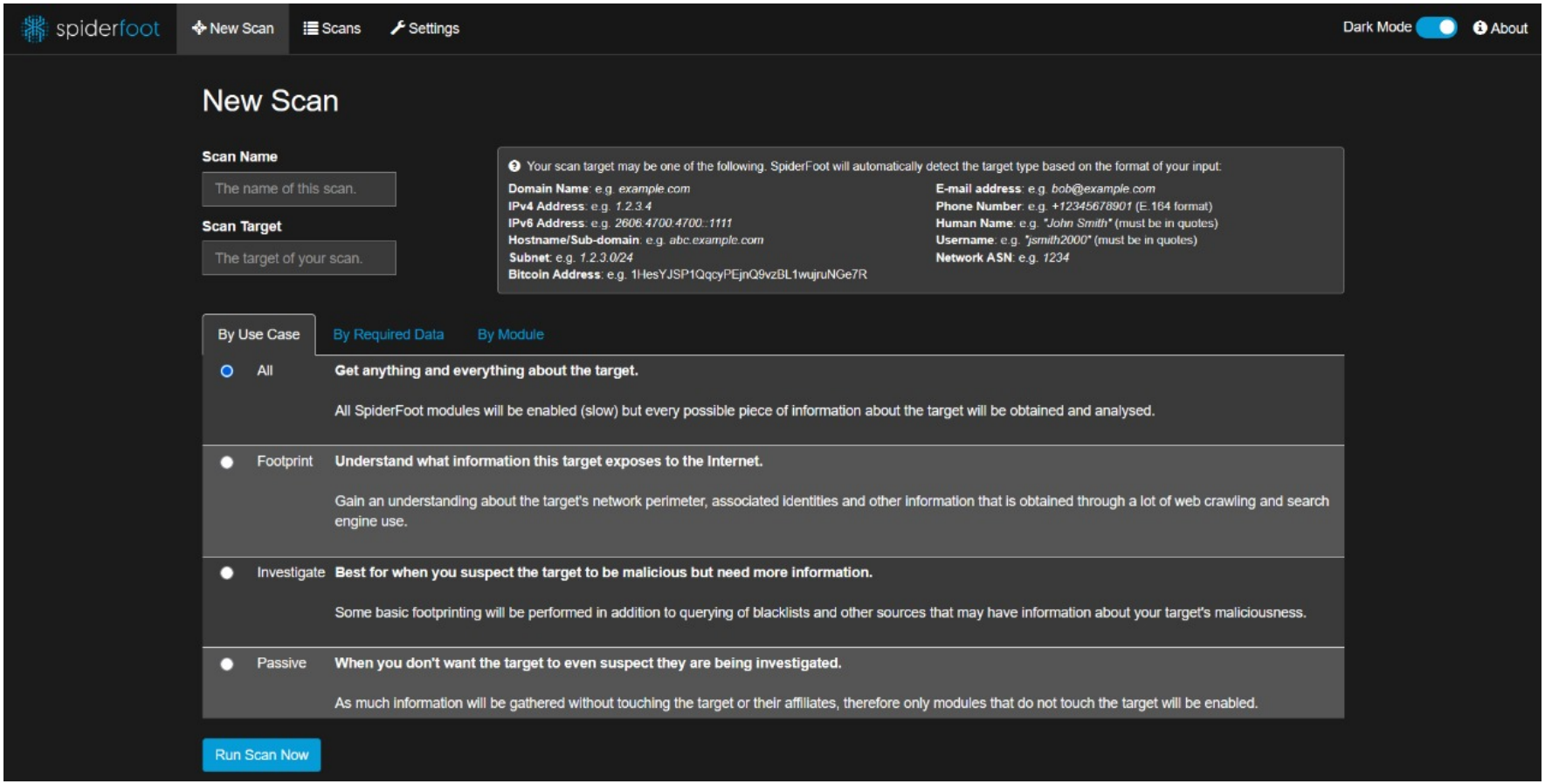


Figure: spiderfoot sample use

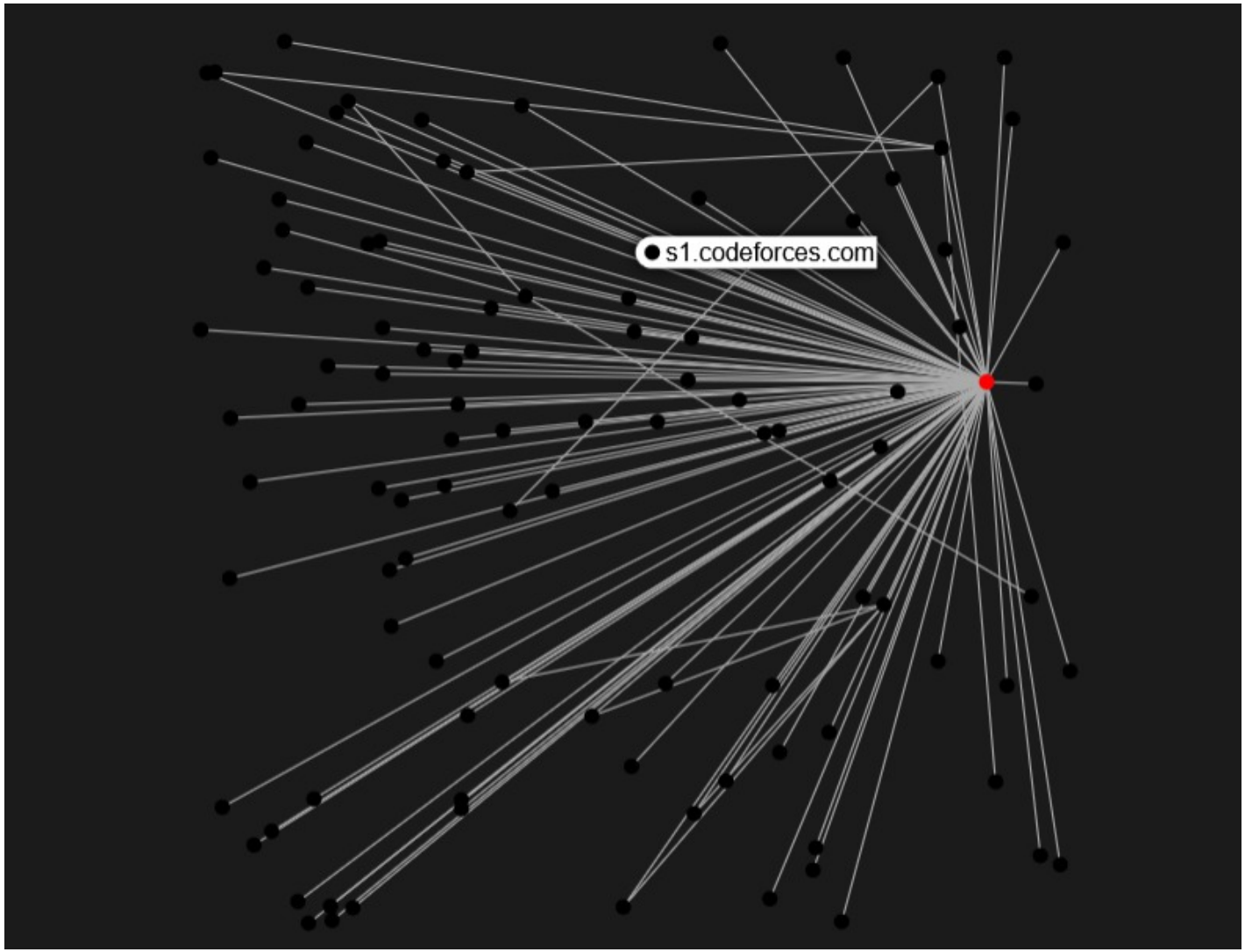


Figure: codeforces network graph

- spiderfoot cli:

```
sf> help

Command             Description
------------------+-------------------------------------------------------
help [command]    | This help output.
debug             | Enable/Disable debug output.
clear             | Clear the screen.
history           | Enable/Disable/List command history.
spool             | Enable/Disable spooling output.
shell             | Execute a shell command.
exit              | Exit the SpiderFoot CLI (won't impact running scans).
ping              | Test connectivity to the SpiderFoot server.
modules           | List available modules.
types             | List available data types.
correlationrules  | List available correlation rules.
set               | Set variables and configuration settings.
scans             | List all scans that have been run or are running.
start             | Start a new scan.
stop              | Stop a scan.
delete            | Delete a scan.
scaninfo          | Scan information.
data              | Show data from a scan's results.
correlations      | Show correlation results from a scan.
summary           | Scan result summary.
find              | Search for data within scan results.
query             | Run SQL against the SpiderFoot SQLite database.
logs              | View/watch logs from a scan.
```

Figure: spiderfoot cli

wafw00f

This tool identifies and fingerprints Web Application Firewall (WAF) products using the following logic:

- Sends a normal HTTP request and analyses the response; this identifies a number of WAF solutions.
- If that is not successful, it sends a number of (potentially malicious) HTTP requests and uses simple logic to deduce which WAF it is.
- If that is also not successful, it analyses the responses previously returned and uses another simple algorithm to guess if a WAF or security solution is actively responding to the attacks.

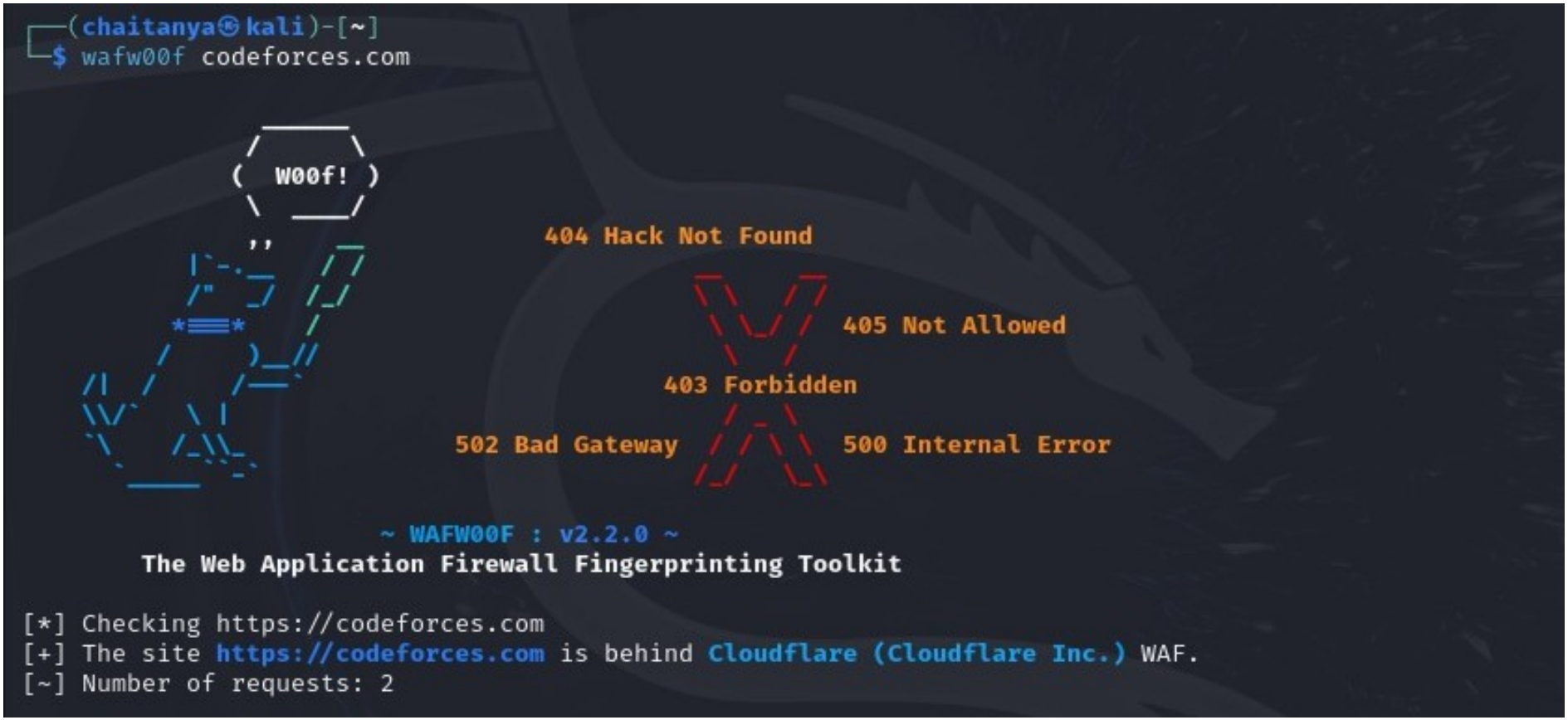


Figure: wafw00f sample use

fierce

- Fierce is a semi-lightweight scanner that helps locate non-contiguous IP space and hostnames against specified domains
- It's really meant as a pre-cursor to nmap, unicornscan, nessus, nikto, etc, since all of those require that you already know what IP space you are looking for.
- It is meant specifically to locate likely targets both inside and outside a corporate network.

Figure: fierce sample use

Referenes and Acknowledgements

- https://www.kali.org
- https://medium.com
- https://www.itjones.com/blogs
- https://github.com/OWASP/owasp-mastg
- Stagefright: Scary Code in the Heart of Android

Thank You!!!