Department of Computer Science
Faculty of Computing and Information Technology

Vulnerability Research and Exploit Development for Android Kernel

Version 1.0

TABLE OF CONTENTS

Vulnerability Research and Exploit Development for Android Kernel

1. Introduction

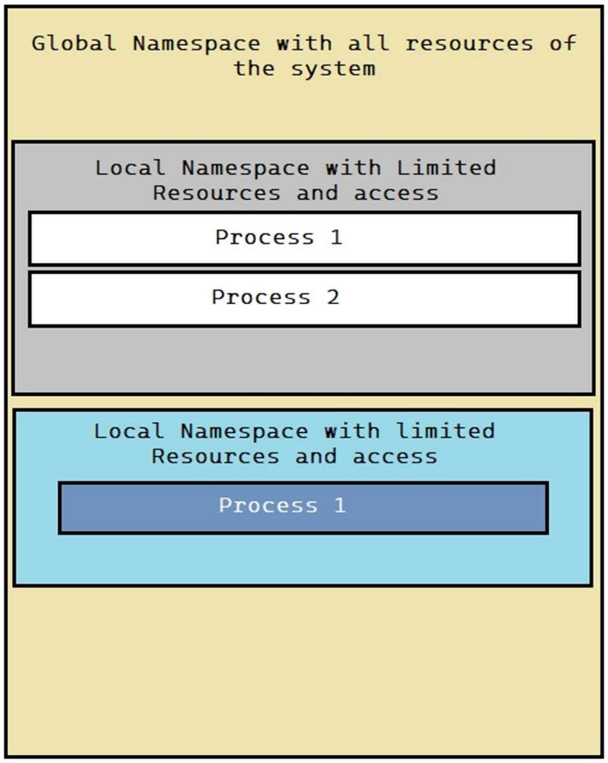
Cyber security is the practice of defending computers, servers, mobile devices, electronic systems, networks, and data from malicious attacks. It's also known as information technology security or electronic information security.

Vulnerability research is the process of researching vulnerabilities to determine if any of them affects your organization's systems. While monitoring vulnerability sources, you must research the vulnerabilities that appear and determine if any affect your organization's systems.

Vulnerability research methodologies are the commonly used principles of auditing systems for vulnerabilities. The process of source code research begins with searching the source code for error-prone directives such as **strcpy**, **gets** and **sprintf**. Another method is the line-by-line review of source code by the person auditing the program, which is a comprehensive audit of the program through all of its execution sequences. Discovery through difference is another method, using the **diff** utility on different versions of the same software to yield information about security fixes. The method of undertaking binary research can involve various utilities such as tracing tools, debuggers, guideline-based auditing, and sniffers. An auditing source code review involves the search for error-prone functions and line-by-line auditing methodologies.

Binary exploitation involves taking advantage of a bug or vulnerability in order to cause unintended or unanticipated behaviour in the program. Memory corruption is a common form of challenge seen in the Binary Exploitation category.

Sandboxing is a cybersecurity practice where you run code, observe and analyze code in a safe, isolated environment on a network that mimics end-user operating environments. Sandboxing is designed to prevent threats from getting on the network and is frequently used to inspect untested or untrusted code. Nowadays even trusted Applications run inside the sandbox in order to minimize the impact of malicious attacks.


Global Namespace with all resources of the system
Local Namespace with Limited Resources and access
Process 1
Process 2
Local Namespace with limited Resources and access
Process 1

Sandboxing is extremely effective mitigation. We need at least two sets of vulnerabilities to escape the sandbox. A set for exploiting the sandboxed process and another set for escaping the sandbox. If sandboxes are correctly configured then we might need vulnerabilities in the kernel in order to escape the sandbox.

Once we have all the necessary sets of vulnerabilities we need to trigger it this is done via writing the Exploit. An exploit is a piece of software, a chunk of data, or a sequence of commands that takes advantage of a bug or vulnerability in order to cause unintended or unanticipated behaviour to occur on computer software, hardware, or something electronic (usually computerized). Such behaviour frequently includes things like gaining control of a computer system, allowing privilege escalation, or a denial-of-service attack.

1.1 Project Title

The title is **‘‘Vulnerability Research and Exploit Development on Android Kernel’’**.

1.2 Project Overview Statement

Triggering the Vulnerability in the Android kernel IPC binder System for the elevation of privilege from an application to the Linux Kernel.

1.3 Project Overview Statement Template

Project Title:
Vulnerability Research and Exploit Development for Android Kernel

Group Leader:

Project Members:

Name	Registration #	Email Address	Signature
Ali Raza	BCF19M513	Bcf19m513@pucit.edu.pk	
Tehreem Iqbal	BCSF19M522	bcsf19m522@pucit.edu.pk	
Faran Abdullah	BCSF19M534	bcsf19m534@pucit.edu.pk	
Arslan Ahmed Qureshi	BCSF19M556	bcsf19m556	

Project Goal:
The goal of this project is to learn the process of vulnerability discovery and then write its exploit to trigger that vulnerability. Our project mainly focuses on **CVE-2019-2215** which is the **Use After Free** vulnerability in Android Kernel’s Binder **IPC** subsystem.
Note: Kernel Version 3.4x

Objectives:

Sr.#	Objective
1	Understanding program interaction and misusing them.
2	Learning the art of crafting the **shell code**.
3	Learning different techniques and principles for implementing effective **sandbox**.
4	Learning the tools and techniques for **reverse engineering**.
5	Understanding different types of **memory errors** and analyzing how they can lead to malicious **code execution**.
6	Learning to Exploit the different mitigation techniques using techniques such as **Rop** and **JIT Spraying.**
7	Understanding the **Dynamic Allocators Misuse**.
8	Setting up the Environment and tools chain for the Kernel Exploitation
9	Understanding the **Race Conditions** in the file system, different kernel processes and Memory. Understanding how race conditions can be used for exploitation.
10	Understanding the Kernel Securities and mitigation techniques such as **SECCOMP**, **SMEP/SMAP**, **KPTI** and **KASLR** and bypassing them
11	Triggering the Vulnerability in Android Kernel **binder IPC system** as per the **CVE-2019-2215**
12	Root Cause Analysis
13	Writing the Exploit for Privilege Escalation

Project Success criteria:

By being able to trigger the vulnerability as per defined in **CVE-2019-2215** and having a firm understanding of the techniques and patterns.

Assumptions, Risks and Obstacles:

Exploiting the vulnerability discussed here requires an understanding of kernel securities and techniques to bypass them resources related to them are rare. Bypassing techniques which require more in-depth knowledge can be time-consuming and disturb the whole time line.

Organization Address (if any):

Faculty of Computing and Information Technology

Type of project: ✓Research ✓Development

Target End users:

Organizations, Institutions, Academic Researchers, Students.

Development Technology: □Object Oriented ✓Structured

Platform: ✓Mobile based

Suggested Project Supervisor: Dr. Muhammad Arif Butt

Approved By:

Date:

1.4 Project Goals & Objectives

1.4.1 Program Interaction and Its Misuse:

Every Linux process has state (running, waiting, stopped, zombie), priority (and other scheduling information), parent, siblings, children and shared resources. Processes can communicate with other processes using different IPC mechanisms provided by the Linux kernel. In Linux, processes propagate by mitosis!

Fork and (more recently) clone are system calls that create a nearly exact copy of the calling process: a parent and a child. Later, the child process usually uses the **execve** syscall to replace itself with another process.

When it comes to privilege, it is important that the processes and applications should only be granted whatever is required for them to carry out their respective tasks. Additional permissions that are not required or necessary can lead to misuse of these permissions and we'll learn how we can misuse them.

1.4.2 Crafting Shellcode:

A shellcode is a small piece of code used as the payload in the exploitation of a software vulnerability. It is called "shellcode" because it typically starts a command shell from which the attacker can control the compromised machine, but any piece of code that performs a similar task can be called a shellcode.

1.4.3 Sandboxing:

Sandboxing is a cybersecurity practice where you run code, observe and analyze code in a safe, isolated environment on a network that mimics end-user operating environments. We'll see several strong mitigations that are so effective that a second vulnerability is needed to bypass the mitigation and make the first vulnerability useful.

1.4.4 Reverse engineering:

Software Reverse Engineering is a process of recovering the design, requirement specifications and functions of a product from an analysis of its code. The purpose of reverse engineering is to understand how a system works and how it is built.

1.4.5 Exploiting mitigation techniques:

Mitigation techniques are the security measures that are used to reduce security threats. These techniques are based on Hardware, Operating system and Compiler-level. In this module, we will be exploiting different mitigation techniques such as Address Space Layout Randomization (ASLR) and Canary.

1.4.6 Dynamic Allocator Misuse:

Heap is the memory area that a process can request dynamically (at runtime). In this module, we will be learning about different vulnerabilities such as use after free and how these vulnerabilities can be exploited.

1.4.7 Kernel Securities and Mitigation:

Since user land has different mitigation techniques, the Operating system kernel also has some security measures to protect the kernel land. These security measures include Supervisor Mode Access Prevention (SMAP) and Supervisor Mode Execution Prevention (SMEP), Kernel Page Table Isolation (KPTI) and Kernel Address Space Layout Randomization (KASLR).

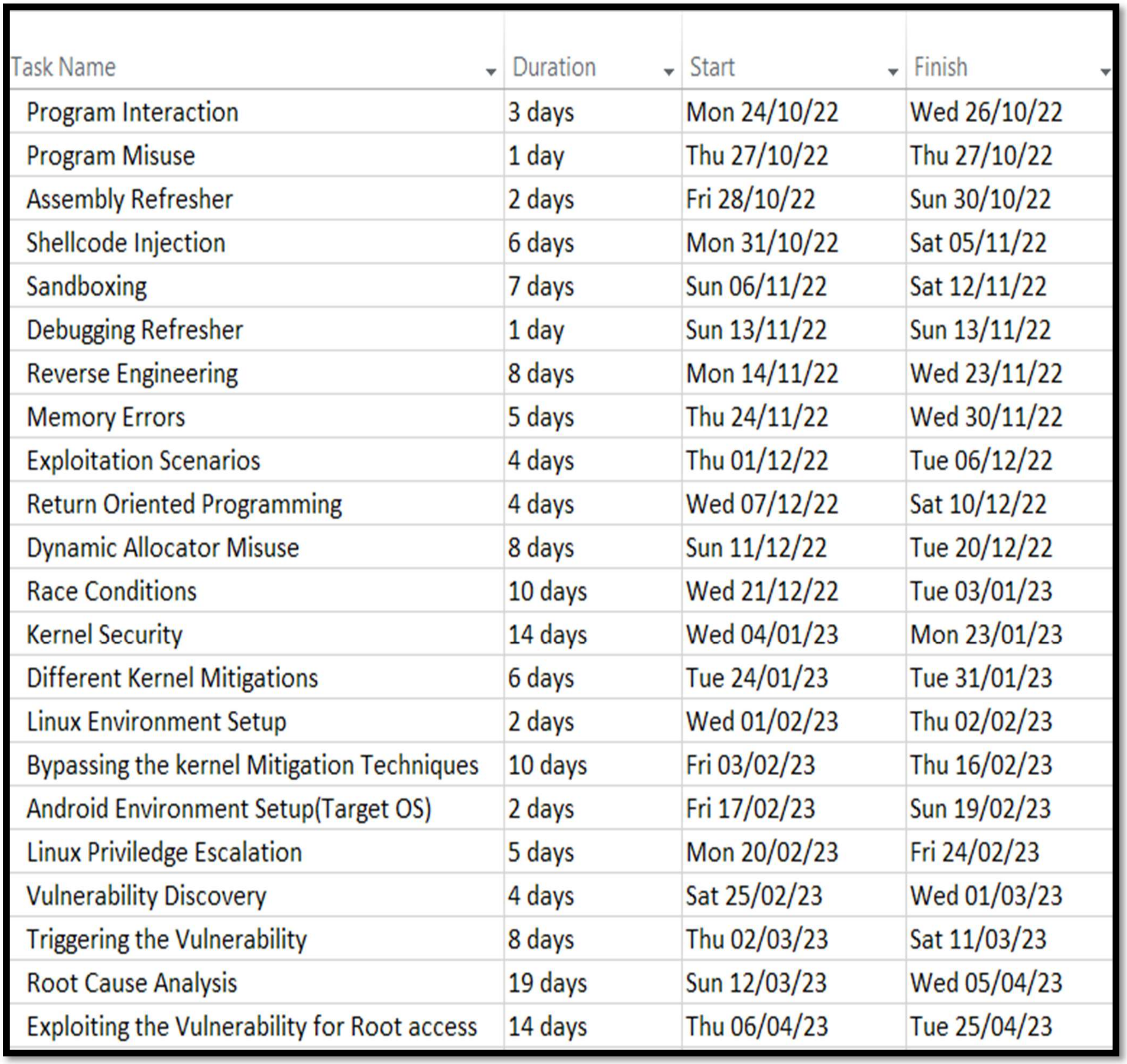
1.4.8 Android Kernel binder IPC system Vulnerability and Its Exploit:

We are going to look at CVE-2019-2215 which is a Use after Free vulnerability in the Binder IPC subsystem. This is a very severe vulnerability because the binder subsystem is reachable from the Chrome sandbox and can lead to privilege escalation if chained with a renderer exploit.

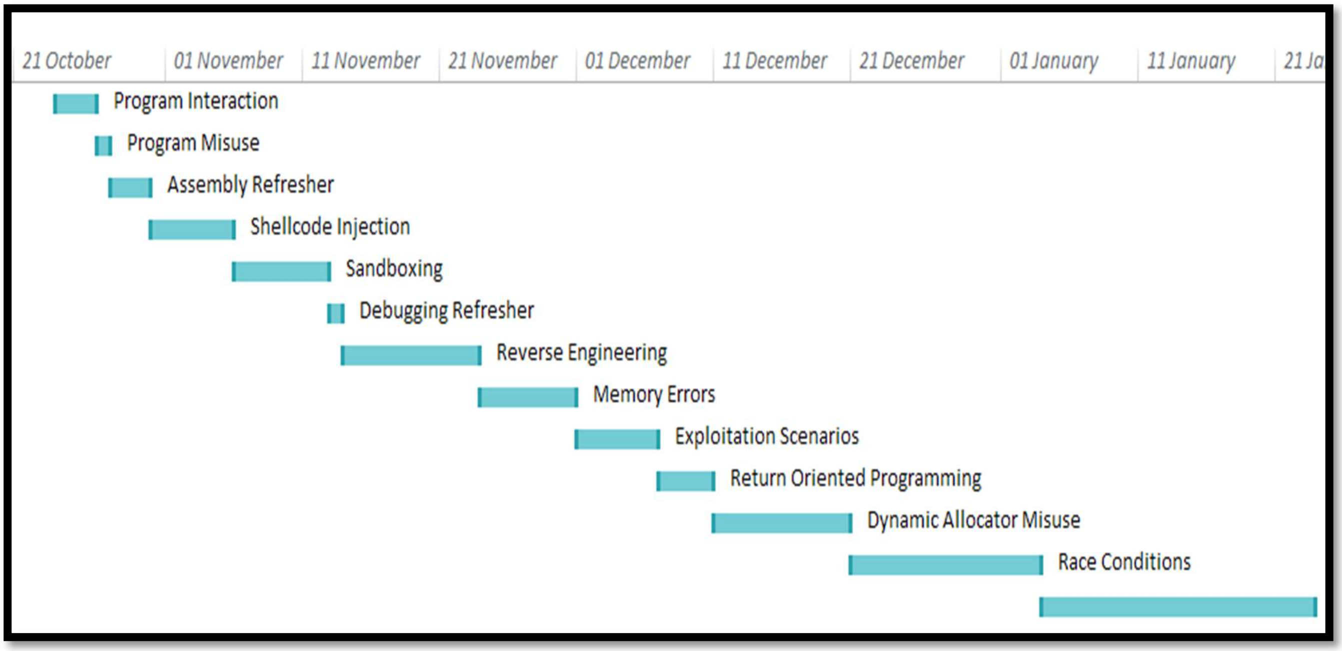
1.5 Exclusions:

We will try to set up physical Android Device and try rooting it remotely.

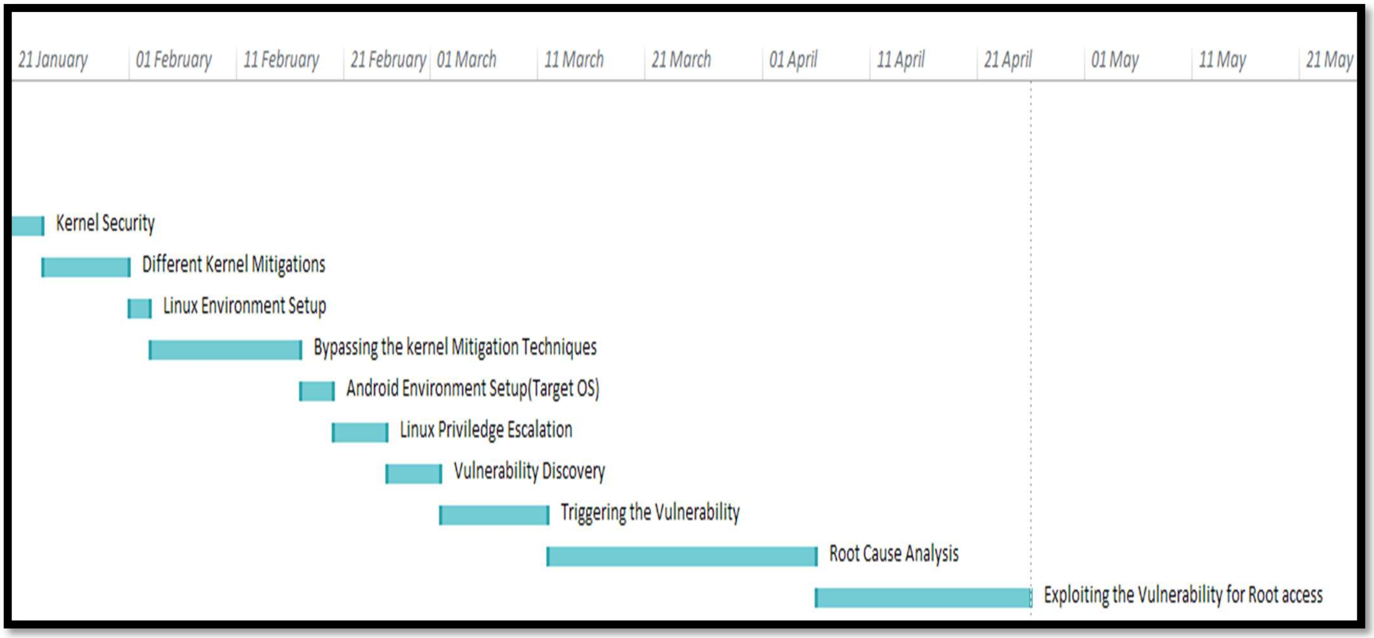
1.6 Duration chart & Gantt chart:



Task Name	Duration	Start	Finish
Program Interaction	3 days	Mon 24/10/22	Wed 26/10/22
Program Misuse	1 day	Thu 27/10/22	Thu 27/10/22
Assembly Refresher	2 days	Fri 28/10/22	Sun 30/10/22
Shellcode Injection	6 days	Mon 31/10/22	Sat 05/11/22
Sandboxing	7 days	Sun 06/11/22	Sat 12/11/22
Debugging Refresher	1 day	Sun 13/11/22	Sun 13/11/22
Reverse Engineering	8 days	Mon 14/11/22	Wed 23/11/22
Memory Errors	5 days	Thu 24/11/22	Wed 30/11/22
Exploitation Scenarios	4 days	Thu 01/12/22	Tue 06/12/22
Return Oriented Programming	4 days	Wed 07/12/22	Sat 10/12/22
Dynamic Allocator Misuse	8 days	Sun 11/12/22	Tue 20/12/22
Race Conditions	10 days	Wed 21/12/22	Tue 03/01/23
Kernel Security	14 days	Wed 04/01/23	Mon 23/01/23
Different Kernel Mitigations	6 days	Tue 24/01/23	Tue 31/01/23
Linux Environment Setup	2 days	Wed 01/02/23	Thu 02/02/23
Bypassing the kernel Mitigation Techniques	10 days	Fri 03/02/23	Thu 16/02/23
Android Environment Setup(Target OS)	2 days	Fri 17/02/23	Sun 19/02/23
Linux Priviledge Escalation	5 days	Mon 20/02/23	Fri 24/02/23
Vulnerability Discovery	4 days	Sat 25/02/23	Wed 01/03/23
Triggering the Vulnerability	8 days	Thu 02/03/23	Sat 11/03/23
Root Cause Analysis	19 days	Sun 12/03/23	Wed 05/04/23
Exploiting the Vulnerability for Root access	14 days	Thu 06/04/23	Tue 25/04/23



21 October
01 November
11 November
21 November
01 December
11 December
21 December
01 January
11 January
21 Ja
Program Interaction
Program Misuse
Assembly Refresher
Shellcode Injection
Sandboxing
Debugging Refresher
Reverse Engineering
Memory Errors
Exploitation Scenarios
Return Oriented Programming
Dynamic Allocator Misuse
Race Conditions



21 January
01 February
11 February
21 February
01 March
11 March
21 March
01 April
11 April
21 April
01 May
11 May
21 May
Kernel Security
Different Kernel Mitigations
Linux Environment Setup
Bypassing the kernel Mitigation Techniques
Android Environment Setup(Target OS)
Linux Priviledge Escalation
Vulnerability Discovery
Triggering the Vulnerability
Root Cause Analysis
Exploiting the Vulnerability for Root access

1.7 Hardware and Software used during Research

1.7.1 Hardware:

- Desktop/Laptop (8GB+ RAM, amd64 with Virtualization Support)
- ARM based Mobile Device

1.7.2 Software:

- Linux(source required too)/Android(source code required)/Windows
- C, Python(subprocess modules, OS modules and more..), JAVA, KOTLIN
- GDB + (peda/gef/pwndbg)
- Ghidra
- IDA
- angr-management
- Ropper
- Rappel
- Visual Studio Code
- Android Studio