**PROPOSED PROGRAMMING PRACTICES TO ADDRESS BUFFER OVERFLOW VULNERABILITIES**

By;

MURITHI ROY KATHURIMA

TED/142/14

A PROJECT PROPOSAL SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF BACHELOR OF EDUCATION IN TECHNOLOGY EDUCATION

MOI UNIVERSITY

MAY 2019

# DECLARATION

**Declaration by the Student**

I hereby declare that this project proposal work is my original work in substance and form, and has not been submitted elsewhere by any other person for the award of Degree in Bachelor of Education in Technology Education to the best of my knowledge.

Signature………………………………………………….

Name………………………………………………………

Date………………………………………………………..

**Declaration by the Supervisor**

I confirmed that the work reported in this proposal was carried out by the candidate under my supervision.

Signature…………………………………………………

Name…………………………………………………….

Date……………………………………………………..

# DEDICATION

I dedicate this project to my mother, Ruth Murithi, for her unwavering support, both financially and psychologically and her grand sacrifices.

# ACKNOWLEDGEMENT

I’d like to appreciate my project’s supervisor Mr. Oloo for the unique perspective he accorded and elevating me to think of the bigger picture. Due gratitude also goes to the projects coordinator Dr. Keter for successful guidance and many directives on how to get necessary materials required for the project. I’d also like to appreciate all the lecturers in the department for all their advice and guidance throughout my stay in the departments; special thanks to Dr. Mutai and Mr. Kemei who have been pivotal in my understanding of the various daunting concepts in the Electrical department.

# ABSTRACT

In the recent years, the problem on vulnerabilities pertaining to the category of Binary exploitation have largely been averted. Various techniques such as non-executable stack, stack cookies/canaries and address space layout randomization etc. have made the injection of shell code almost infeasible and its execution impossible on modern systems due to the daunting computational complexity of the task. This project aims to introduce and explore a various ways of exploiting buffer overflows to run an attacker’s code in the modern system and finally suggest methods in which these vulnerabilities can be averted. There are various types of overflow vulnerabilities whose nature will be briefly explored. The methods to exploit these vulnerabilities will be considered chronologically in order to show how complex the task has gotten over time and that its still feasible even with the afore mentioned modern techniques notwithstanding. Various exploitation techniques for these vulnerabilities will be explored for example; injecting shell code, doing a return-to-lib attack, borrowed code chunks and finally explore modern techniques such as Return Oriented Programming. Return oriented programming (ROP) is a way of using the already available ‘innocent’ instructions in any typical program, undermining the developer’s trust assumptions, and making those instructions execute virtually anything based on the available ‘gadgets’ without having to inject any new code. A developer would create a complex system for a specific function, say a school management system, but in so doing casts shadows of weird machines, whose maliciousness is only limited by the creativity of the attacker. The project will also briefly examine Sigreturn oriented programming (SROP) and Jump Oriented Programming attacks that aims to make the attack more efficient and evade detection in modern techniques. It is to be noted that these techniques provide the attacker with a Turing complete machine for which an attacker can write a programming language for, to execute virtually anything on the target program.

# TABLE OF CONTENTS

[DECLARATION i](#_Toc12260479)

[DEDICATION ii](#_Toc12260480)

[ACKNOWLEDGEMENT iii](#_Toc12260481)

[ABSTRACT iv](#_Toc12260482)

[TABLE OF CONTENTS v](#_Toc12260483)

[CHAPTER 1 1](#_Toc12260484)

[1.0 Background information 1](#_Toc12260485)

[1.1 Statement of the problem 1](#_Toc12260486)

[1.2 Objectives of the study 1](#_Toc12260487)

[1.3 Research questions 1](#_Toc12260488)

[1.4 Justification and significance 1](#_Toc12260489)

[1.5 Merits and demerits 1](#_Toc12260490)

[1.6 Assumption and precautions 1](#_Toc12260491)

[CHAPTER 2 1](#_Toc12260492)

[2.0 Literature review 1](#_Toc12260493)

[CHAPTER 3 1](#_Toc12260494)

[3.1 Methodology 1](#_Toc12260495)

[3.2 Procedure 1](#_Toc12260496)

[3.3 Observations and result 1](#_Toc12260497)

[CHAPTER 4 1](#_Toc12260498)

[4.1 Data analysis 1](#_Toc12260499)

[CHAPTER 5 1](#_Toc12260500)

[5.1 Conclusions and recommendations 1](#_Toc12260501)

[5.2 Reference 1](#_Toc12260502)

# CHAPTER 1

## Background information

Most programmers, new into the field or otherwise, all exhibit the tendency to simply write code that performs its task. In most part, that is what a typical client requires. This tendency has been worsened by the development of frameworks or abstract constructs that aim to make development of software goal oriented. This is usually a productive approach but within it lies a fundamental. These programmers are usually aware of what they are doing in the layer of abstraction that they are working on. However, a lot of details are invisible to the programmer and this can have serious security implications. It is important for developers, especially those working in low levels languages like C, to learn how to write code that is secure not just one that works. This study aims at exploring the reasons as to why programmers should take a moment and appreciate the gravity of why it is important to write secure code. It can save an organisation millions because a single bug can be potentially catastrophic. Also, most modern day developers live in a blissful bubble where they reason that modern systems are usually secure and bugs are usually fixed at the hardware, OS, language and framework level. This statement is usually true but at the end of the day, these constructs depend on the human element in order to properly function. There is a general assumption that the developer understands the concept well enough to use the feature securely but is not usually the case.

Before delving into the subject matter however, it’s important to define a few terminologies in a bid to deter from ambiguity:-

* **Vulnerability: -** This is a flaw in a system’s security that can lead to an attacker in a manner other than the one that the designer intended. This can include impacting the availability of the system, elevating access privileges to an unintended level, complete control of the system by an unauthorized party among other possibilities.
* **Exploit: -** As a verb, this’ to take advantage of a vulnerability so that the target system reacts in a manner other than which it was intended for.

As a noun, it means the set of tools, set of instructions, or code that is used to take advantage of a vulnerability.

* **Buffer Overflow**: - A buffer overflow or buffer overrun is an anomaly where a program when writing data into memory exceeds the boundary of the buffer and writes data into the adjacent memory location.
* **Call Stack**: - This is a stack data structure that stores the information about the active subroutines of a computer program
* **Process**: - This is an instance of a program executing in memory.
* Inter-Process Communication (IPC): - This is a mechanism that allows processes to communicate with each other
* **Pipes**: - This is a unidirectional data channel used to implement IPC.
* **Pseudoterminal**: - This is a pair of pseudo-devices, one of which is the slave, emulates the hardware text terminal device, the other of which is the master, provides a way by which a terminal emulator process controls the slave. This is used in the paper to implement a bidirectional data channel for IPC.

In the earlier days, hacker would

## Statement of the problem

The problem when programming at a low level or using a low level programming language, the assumption is that you know what you are doing. The security features implemented by the operating system and the underlying hardware do not enforce their in-built security policies on the software but rather rely on the software to opt in. The programmer’s not only need to learn how to write code but also understand the programming constructs that they implement.

People also opt for cheaper solutions when acquiring software. What people do not understand is that cheap can become very expensive. Even very large companies such as Google, Microsoft and Amazon etc. invest a lot on the issue of security and even today with decades of experience and state of the art technology at their disposal their software is not immune to vulnerabilities. This should say a lot about code developed by an upstart.

## Objectives of the study

The objective of this paper is to demonstrate how simple misuse of a commonly used programming feature due to ignorance among other reasons can be used to take over a system. The project will illustrate the attack on a modern system and even demonstrate various methods in which the attack can be optimised. The paper then makes recommendations how these vulnerabilities can be thwarted by following proper programming constructs and demonstrated their effectiveness. The paper also aims at introducing the reader to a new attitude and instil values related to the way they should view the topic of security.

## Research questions

The questions to be asked in order to propel this study include;

* Are exploitable vulnerabilities obsolete in modern systems?
* Is it possible to write code that is vulnerable in a modern system?
* Do programmers understand all the programming constructs in the languages that they use?
* Why is it important to always update one’s system if the system is still vulnerable?
* Do modern systems provide a completely automated and reliable way of securing applications?
* Can programming techniques help to make applications more secure?

## Justification and significance

It is merely last month that Safaricom lost millions due to a flaw in their system, details of which were undisclosed, that allowed people to purchase any of their products (Airtime, Data Bundles, SMS or voice bundles) using Bonga points that the subscriber did not possess. The flaw was however eradicated and attempts to recover the losses were put in place. This is a clear demonstration that it is very important to not only write code that works but also one that is secure.

## 1.5 Merits and demerits

## 1.6 Assumption and precautions

This research is performed in Linux distributions and no attempt was made to extend the attacks to extend the exploit in other operating system platforms. Some of the attacks performed rely on the features of Linux and also because Linux is open source as opposed to either MAC or windows which are proprietary. This, however, does not mean that these systems are secure and that the discussed approaches would not apply to them. On the contrary, they too have vulnerabilities that are specific to the platform and thus during exploit development some attempt has to be made in order to port the shell code to test the distribution it is being run on in order to determine the code to execute.

Also, the assumption is that when the paper talks about earlier versions of the Linux operating system, the same concepts can be applied to the earlier versions of other operating systems such as Windows as well.

The attacks demonstrated in the project here will mainly be used to perform privilege escalation as a proof of concept and assume that generally with root access, an attacker can basically perform any kind action on the target system.

The methods demonstrated in this project are intended only for educational purposes.

# CHAPTER 2

## Literature review

….talks about the geometry of assembly instructions

# CHAPTER 3

## 3.1 Methodology

The project aims at first introducing the vulnerable programs compiled for and running earlier in operating systems and then explore the complexity of exploiting them. The same is then done for a modern system. The programs compiled for older systems and then run on the modern systems and attempts at exploiting them are done. At each stage, a patch will be demonstrated as to how the said vulnerability can be mitigated.

Observations are then made and recorded and the findings analysed.

Self-modifying programs will also be explored briefly

## 3.2 Procedure

Initially, we shall consider a normal buffer overflow that can be used to redirect execution to another point in the program. This is useful, for instance if an individual was interested in bypassing the security features of a program that is proprietary. We shall then devise a much secure program and show a way that can make a program more secure then crack that too using a keygen program. The example code is as shown below;

Parser differential technique will is a technique that can be used to protect the program from being disassembled using a debugger. This has the advantage that an attacker cannot disassemble the program to look for bugs. The downside is that the program becomes a challenge to debug in case of problems and the same method can be used by an attacker to prevent his malware from being disassembled. The example of code below shows a rudimentary example of this;

Data Execution Prevention or W^X is a security policy that ensures that the areas in memory that can be written to cannot be executed while areas that can be executed cannot be written to. This is a feature that is implemented by the operating system. This defence in essence eliminates attacks such as shell code injection. Address Space Layout Randomization is a technique used to randomize the areas where the code is loaded every time the program is executed. A stack canary is a value placed before the return addressed on the stack to prevent the exploitation of buffer overflows. It works by comparing the value of the canary on the stack with a value loaded into a particular register and if they are not the same, the program will never return.

Position Independent Executables

A way to defeat ASLR and guess the stack canary in a 32bit binary in Linux box is to write a program that spawns a child process then runs the target program in the child process. The parent would then send the exploit to the child and wait for the child to respond. Is the child terminates without having redirected code execution then the child is re-spawned and run in a loop until the target has been hit. This is a rather crude method that involves brute force and thus requires a lot of CPU time. However, the randomness of the canary in a 32bit machine is semi-random i.e. one only needs to guess only four bytes as the others remain constant. The code snippet below shows the implementation of this;

An alternative would be to reverse engineer the binary and compile with an older compiler to remove the security features.

## 3.3 Observations and result

When you run code that is compiled on an older system in a new system, the application is not protected by the security features of the modern system. This is because the security features are not enforced on the application by the modern system. The modern simply supports the features and depends on the program to opt in. An application compiled in a modern system however has these settings set by default because the modern compiler is optimised by default to opt in to them.

The programs

# CHAPTER 4

## 4.1 Data analysis

Various tools have been employed to assist in the analysis of the programs, these include; gdb, redare2, hopper, objdump, strace, ltrace among others. They have been used to provide data such as disassembly information, show flow of execution, read values of registers and memory at various instances of time while stepping through the code, show the address of functions during code execution.

# CHAPTER 5

## 5.1 Conclusions and recommendations

People should aim at acquiring verified software solutions from trusted companies in case there data is to protect especially if it is confidential or its integrity is imperative.

Run security auditing for your software regularly to ensure that is up to date with the state the art techniques.

Make sure that the software is up to date.

Make sure that the system on which the software is run is also up to date.

Care should be taken when installing old applications in new or otherwise,

At the end of the day, the issue of security has been a cat and mouse game between the hacker and the security personnel. We must accept the fact that there is no system that is immune to penetration and exploitation. It is just a matter of the complexity of the task which can be decomposed to the amount of resources required, the skills and the determination of the hacker(s). This is because these systems are built and maintained by man and since man is prone to error, these systems cannot be perfect. Thus the attempts made are aimed at frustrating the attacker and making the hack impractical.

## 5.2 References