Security

Daniel R. Cender

Grand Canyon University: CST-221-O500

Feb 20, 2020

This assignment will explore a variety of angles around the topic of security in operating systems.

**Buffer Overflow**

**Explanation**

A buffer overflow is a condition in a program where the attempted memory access exceeds the scope of memory allotted for the program. It’s often exploited as a security vulnerability in large programs based in languages like C/C++ that have little or no automatic memory garbage collection or array bounds checks.

**Diagram**

Figure 1 shows the process of how a buffer overflow can occur.

A close up of a map

Description automatically generated

*Figure 1.* Diagram of how buffer overflow occurs and the result.

**Issues and Harm**

Buffer overflow is, at the very least, a great annoyance to software developers. Programs may crash at the worst moments or behave erratically if buffer overflow cases are not sniffed out in the code and fixed promptly. Concerning security, however, buffer overflow is a massive spot of potential vulnerable which operating systems must vigilantly prevent from occurring. If an attacker can override a function pointer in the memory heap, they could make the program crash, call an unwanted function, or a malicious piece of injected code (Arora, 2013).

**Prevention**

Modern CPUs have a few methods for preventing buffer overflow. One of the most common methods is using pieces of data on the stack, called “stack canaries,” which are checked for changes which will indicate an overflow of allocated address space (Tanenbaum, 2015). This may work for most cases, but highly intelligent attackers can still circumvent stack canaries. Modern operating systems support an “NX” (No-eXecute) bit, dictates that data bits may be written to, and text pieces may be executed (Tanenbaum, 2015). More broadly, this is known as DEP (Data Execution Prevention), which can be enforced in hardware or software (Tanenbaum, 2015). As a developer, it’s imperative to not implicitly trust users to input the information you expect of them. User input should always be checked and cleaned of improper statements.

**Zero Day Exploit**

**Description**

A zero-day exploit is a security vulnerability that originates in a bug produced by a certain software or hardware vendor, which has yet to be patched (Porup, 2019). It’s called a “zero day” vulnerability because that’s exactly how many days a fix has been available for it. These exploits are commonly discovered and then traded around on both “black markets” and “grey markets,” where there is a fuzzy dividing line between which state agencies are allowed to buy or not (Porup, 2019). There are many cases where organizations offer “bug bounties” to those who discover zero day exploits and turn them into the vendor promptly. Unfortunately, a whole other host of organizations drag their feet when alerted to these exploits by private researchers, resulting in greater damage if the exploit ends up discovered and used by a malicious actor first.

**Ethical Issues**

Today, work and vocation are primarily seen as a means towards pure self-fulfillment, self-actuation (Kim, Fisher, & McCalman, 2009). This view prizes material gain and contrasts the values of the more traditional, Christian perspective that work exemplifies a person’s God-given purpose to steward resources in diligent service. Zero day exploits certainly seem like a handy way for someone to make quick money, and potentially from multiple parties. A person can go ahead and sell the exploit on a grey or black market (whichever pays highest) and then sell it for a “bug bounty” to the vendor, profiting from multiple angles without consideration to how the black market may use the knowledge. This would be upholding a post-modern view of vocation and use of one’s skillset. But, someone working from a Biblical perspective would likely be compelled to reveal their findings to the vendor so any bugs can be fixed before the masses are negatively impacted. Although they might stand to gain materially from their technological prowess, it might as well be them one day that will be hurt by a zero-day exploit attack. It’s best for them to treat their fellow computer users as they would like to be treated, by protecting vulnerable devices from exploits they have ready knowledge of.

**Kali Linux**

**Description**

Kali is a Debian-based Linux distribution put out and maintained by Offensive Security today (Everything You Need To Know About Kali Linux, 2019). It is a free operating system packed full of tools useful for network analysts, penetration testers, and ethical hackers. Since it is an open-source project, all of its source code is publicly available, along with the six hundred or so tools it comes installed with. This operating system focuses in on that purpose of security testing, so it’s not a system anyone might expect to be advertised for consumer models of computers, as macOS or Windows are.

**Tools**

Some popular tools that “ship” with Kali are:

* Aircrack-ing: Suite of tools that capture WiFi network packets, export data for further processing, simulate network attacks, test network card capabilities, and test for broken encryption usage (WEP or WPA PSK).
* Nmap: This tool discovers the active hosts on a network by intercepting and evaluating IP packets en-route. It can determine what operating systems are running, what firewalls may be operational, and what applications may be conducting activity on the network.
* THC Hydra: This tool conducts brute-force attacks against various network protocols (HTTP, HTTPS, Telnet, FTP, etc.).

There are many more tools that ship with Kali, too many to list out for this assignment.

**Ethical Issues**

There are some large ethical dilemmas that arise when someone becomes proficient in using the tools that come with Kali. It grants users the ability to find many kinds of network security vulnerabilities, with minimal intimate knowledge of cryptography really. The pervasiveness of moral relativism begs whether a community of Kali users could even decide on what is ethical/unethical use of its tool set. Every person could potentially define morality by practicality, personal fulfillment, societal betterment, Biblical standards, or a multitude of other means and outcomes (Kim, Fisher, & McCalman, 2009). One could learn one or two of these tools, then take them to a local café and sniff out a dozen credit cards, or passwords being used openly on an unsecured network connection. From a Biblical perspective, this would be contrary to how God would have humans treat each other and their belongings. Yet, the fragmentation of society around relativism shows that few people may hold this perspective of using these powerful tools. As a software developer, knowledge of Kali and its tool chest may be useful in granting awareness of glaring vulnerabilities in code we write, of which we’d otherwise be ignorant. This could serve to better stewarding our use of coding abilities and how we approach security for the sake of our users.

**Password Strength Bash Script**

**Description**

This bash script takes a single parameter as input from a user, and then evaluates the password for its secure status based on a set of checks.

* Checks the length of the password, and alerts the user if their password is less than 8 characters.
* Checks the password for a numeric character (digit). The test alerts the user and exits the script if there isn’t at least 1 digit.
* Checks the password for a non-alphanumeric character of the kind in (@, #, $, %, &, \*, +, -, =). The script exists early if none are found.

**Results**

**A screenshot of a cell phone

Description automatically generated**

*Figure 2.* Test output from a script that checks a password for security flaws.

**Source**

The source code for this script can be found on my GitHub account at: <https://github.com/DanielCender/CST-221/tree/master/Wk7/password.sh>

If the code cannot be found at that location, it is below in full:

#!*/bin/bash*

#

# *Desc: A script that checks the password inputted for security vulnerabilities.*

# *Author: Daniel Cender*

# *Date: Feb 21, 2020*

#

rxContainsDigit="*(\d).*"

# *Even though \W means Non-alphanumeric,*

# *our quest is to check for one of these specific non-alphanumeric chars*

rxContainsNonAlphaNumeric="*[@#\$%|&\\*\+-=].*"

pass=$1

# *Measures nbr of arguments in commands passed into script*

# *If commnads are*

*if* [ $# -ne 1 ]*;* *then*

echo "*Your command line contains* $*# arguments.*"

echo "*Please pass in single password argument as a string!*"

*else*

# *Evaluate password*

echo "*Passed in:* $*pass*"

size=${*#*pass}

*if* [ $size -lt 8 ]*;* *then*

echo "*Your password is not long enough! It's only* $*size characters long!*"

exit

*fi*

# *Check for digits in password*

*if* *!* [[ $pass =~ $rxContainsDigit ]]*;* *then*

echo "*Your password needs to contain at least 1 digit (0-9) character!*"

exit

*fi*

# *Check for Non-alphanumeric chars*

*if* *!* [[ $pass =~ $rxContainsNonAlphaNumeric ]]*;* *then*

echo "*Your password needs to contain at least 1 Non-alphanumeric character!*"

echo "*Acceptable characters are any of: (@, #, $, %, &, \*, +, -, =)*"

exit

*fi*

echo "*\*\*\* Congratulations! Your password is secure! \*\*\**"

*fi*

**User Management Bash Script**

**Description**

This script takes a few arguments (exactly 3) and runs a process to manage users on a Linux operating system. By setting the 3rd argument to either ‘-a’ (add users) or ‘-r’ (remove users), the script runner can add users from a text file, formatted as lines of usernames and hashed passwords, separated by a space character.

**Results**

A screenshot of a cell phone

Description automatically generated

*Figure 3.* Output from running script to create five Linux user accounts and home directories.

Figure 3 shows the logged output from running the script with a list of users and a group to create. The contents of the users.txt file is below:

newUser1 $1$secret$O.fwKYPvUCNtpXo/GBVDY.

newUser2 $1$secret$tSLuH6ddwI7tOYBI77u9k.

newUser3 $1$secret$pUUdb.pg.15bctSGIyI1X.

newUser4 $1$secret$0rG7qgFnuQdosisx9K3D01

newUser5 $1$secret$iKgbR90H4/nRoQRhb0Lgz1

When the /etc/group file is opened in Ubuntu Linux’s default text editor, the new group and users can be seen (shown in Figure 4).

A screenshot of a cell phone

Description automatically generated

*Figure 4.* The/etc/group file after new group has been created.

Once these users and their umbrella group was created, I executed the script again to remove them all.

A screenshot of a cell phone

Description automatically generated

*Figure 5.* The same /etc/group file after removing new group and users.

As of right now, this script fails to explicitly check the format of the text file’s contents; the integrity of the input is dependent on the script runner’s inspection. Ideally, the script would check the input for errors, blank lines, or inappropriate characters.

**Source**

The source code for this script can be found on my GitHub account at: <https://github.com/DanielCender/CST-221/tree/master/Wk7/management.sh>

The raw code for this script is also printed below:

#!*/bin/bash*

#

# *Desc: A script that adds users to a Linux operating system and user group*

# *using a file as input. Please execute with 'sudo' prepended.*

# *Author: Daniel Cender*

# *Date: Feb 22, 2020*

#

#

# *Some Resources:*

# *- Parsing text files: https://stackoverflow.com/questions/37474694/parsing-lines-in-text-in-bash-script*

# *- Encrypting passcode with openssl: https://stackoverflow.com/questions/10106771/encrypt-a-string-using-openssl-command-line/38581705*

# *- Creating users with hashed password: https://www.techrepublic.com/article/how-to-create-users-and-groups-in-linux-from-the-command-line/*

# *- Passwords hashed with: openssl passwd -1 -salt secret password1*

# *Parse out three command-line arguments*

userFile=$1

groupName=$2

opFlag=$3

echo "*Passed in* $*opFlag*"

*if* [[ $opFlag != "*-a*" && $opFlag != "*-r*" ]]*;* *then*

# *This is an invalid 3rd argument, exit script*

echo "*Third argument must be either an -a (add users) or -r (remove users) flag*"

exit

*fi*

# *Measures nbr of arguments in commands passed into script*

*if* [[ $# -ne 3 ]]*;* *then*

echo "*Your command line contains* $*# arguments.*"

echo "*Please pass in the 3 required args: (user filename, group name, operation flag)!*"

*else*

# *Operate on users as specified*

echo "*Checking for valid group*"

# *Check if user group already exists*

*if* echo "$*OUTPUT*" *|* grep -q $groupName /etc/group*;* *then*

echo "*User group already exists*"

*if* [[ $opFlag == "*-r*" ]]*;* *then*

echo "*Deleting user group now*"

# *Force the group's deletion*

groupdel $groupName

*fi*

*else*

*if* [[ $opFlag == "*-a*" ]]*;* *then*

echo "*Creating user group*"

groupadd $groupName

*fi*

*fi*

# *Parse out input file*

cat "${*userFile*}" *|* *while* IFS="" read username password

*do*

[[ $opFlag == "*-a*" ]] *&&* echo "*user '*$*username' will now be created if they don't exist already...*"

[[ $opFlag == "*-r*" ]] *&&* echo "*user '*$*username' will now be deleted if they exist already*"

cat /etc/passwd *|* grep ${username} *>*/dev/null *2>&1*

*if* [[ $? -eq 0 ]] *;* *then*

echo "$*username Already Exists!*"

[[ $opFlag == "*-a*" ]] *&&* usermod -a -G $groupName $username

# *The -r option on userdel removes mail spools*

# *and user directories for us*

[[ $opFlag == "*-r*" ]] *&&* userdel -r $username

*else*

echo "*User Not Found*"

# *create user with home directory and group assignment*

[[ $opFlag == "*-a*" ]] *&&* useradd -m -G $groupName -p $password $username

*fi*

*done*

echo "*\*\*\* User management completed! \*\*\**"

*fi*

References

Arora, H. (2013, June 4). Buffer Overflow Attack Explained with a C Program Example. Retrieved February 21, 2020, from <https://www.thegeekstuff.com/2013/06/buffer-overflow/>

Everything You Need To Know About Kali Linux. (2019, December 20). Retrieved February 21, 2020, from https://www.edureka.co/blog/ethical-hacking-using-kali-linux/

Kim, D., Fisher, D., & McCalman, D. (2009). Modernism, Christianity, and Business Ethics: A Worldview Perspective. *Journal of Business Ethics*, *90*(1), 115–121. https://doi-org.lopes.idm.oclc.org/10.1007/s10551-009-0031-2

Porup, J. M. (2019, July 30). What is a zero day? A powerful but fragile weapon. Retrieved February 21, 2020, from <https://www.csoonline.com/article/3284084/what-is-a-zero-day-a-powerful-but-fragile-weapon.html>

Tanenbaum, A.S. & Bos, H. (2015).*Modern Operating Systems.*Chapter 9.