An Analysis of the Deployment of Exploit Mitigation Techniques in Operating Systems

Vinod-Kumar Mechiri and Dhruv Madappa

Abstract

The following paper will serve as a comprehensive analysis on the deployment of exploit mitigation techniques on different compilers and various binaries in operating systems. The paper first provides an overview of exploits and explains the exploit mitigation techniques chosen for our analysis. The paper then focuses on mitigations used in different versions of compilers and provides recommendations on which versions are ideal to use based on the versions that are more secure. Following this, we move on to the core focus of this paper, which is analyzing the mitigations implemented in the binaries of the Operating Systems, namely Ubuntu 18.04.3 and Windows 10, which are the most recent versions of these operating systems. We first explain the data collection methodology used to extract the binaries of the operating systems and highlight the total number of binaries that do not have the appropriate mitigations. This is followed by an in-depth evaluation of specific important binaries, programs and applications that we have identified as critical that have not enabled mitigations. We then compare the availability of mitigations present in Ubuntu 18.04.3 and Windows 10. Following this we then conclude by discussing the results obtained from our analysis.

Introduction

Exploits are pieces of software or a series of commands that take advantage of any vulnerabilities or bugs found in computer software or hardware in order to perform an attack[1]. Research done on the statistics of reported memory corruption shows that there is an upward trend in the number of reported vulnerabilities in computer systems in recent years. In order to combat this explosive growth in exploits, Exploit Mitigations techniques have been developed to strengthen computer systems and make the exploitation of these vulnerabilities more difficult to achieve [2].

The project focuses on the following six exploit mitigation techniques [3].

1) **Position Independent Executable (PIE):** PIE is a piece of machine code that executes properly regardless of its absolute address when placed somewhere in the primary memory and is commonly used for shared libraries.
2) **Address Space Layout Randomization (ASLR):** ASLR is a memory-protection process for operating systems that guards against buffer-overflow attacks by randomizing the location where system executables are loaded into memory. ASLR involves randomly positioning the base address of an executable and the position of libraries, heap, and stack, in a process's address space.
3) **Non-Executable Bit (NX):** A piece of technology that is used in CPUs to segregate areas of memory for use. This is done either by storage of processor instructions or for the storage of data. The NX bit marks certain areas of memory as non-executable.
4) **Stack Canaries / Stack-Smashing Protection (SSP):** Stack canaries work by placing a randomly chosen integer at the start of the program in memory before the stack return pointer. This helps to detect a stack buffer overflow prior to the execution of any malicious code since the canary value would be checked to see if it has been changed, which would be the case in most buffer overflow attacks.
5) **Fortify_Source:** The FORTIFY_SOURCE macro serves to detect buffer overflows in various functions that perform operations on memory and strings. It provides an extra level of validation for some functions that are potentially a source of buffer overflow flaws. It protects both C and C++ code. FORTIFY_SOURCE works by computing the number of bytes that are going to be copied from a source to the destination.
6) **RELocation Read-Only (Relro):** RELRO is a generic mitigation technique that hardens the data sections of an ELF binary/process. It does this by making the linker resolve all dynamically-linked functions at the beginning of an execution and make the GOT read-only.
7) **Control Flow Guard (CFG):** CFG is a security feature that was created to combat memory corruption vulnerabilities. It places tight restrictions on where an application can execute from, which makes it harder for exploits to execute arbitrary code through vulnerabilities. CFG is an extension from other exploit mitigation techniques like ASLR and DEP [4].

In order to understand exploit mitigation techniques, we first conducted research into the availability of current mitigation techniques, how they prevent exploits and how they are implemented on compilers. We decided to choose the six exploit mitigation techniques above as

they are currently the security techniques most commonly available on most recent distributions in the Linux environment. Research has shown that while these mitigations help to combat exploit attacks, they may interact at multiple levels and finding a perfect interaction between different mitigations on a system is still an unsolved issue. The use of one mitigation that protects against one attack vector may make another attack vector easier [5]. Additionally, the use of all or some of any of these mitigations may affect on the performance of a system, which would make it undesirable to implement in many cases (such as in a developer environment). Our initial focus was to incorporate the performance overhead of each mitigation technique analyzed into this project. However, this could not be done as the spec benchmark tool needed for this purpose was too expensive to purchase and hence will be part of future work on this topic our team aims to work on following the completion of this paper [6].
While much work on exploit mitigation has been done by academia, another problem is the reproducibility of this work in industry systems. Research into mitigation techniques that have been published in paper, have not been transferred onto the industry mainly due to it being difficult to reproduce [5]. Even basic hardening schemes (such as stack canaries and position independent code) that are readily available in compilers are not adopted and most shipped binaries only have few of the basic mitigations in place. Besides this, not much work has been done to understand the use of exploit mitigation techniques in commodity operating systems. This paper seeks to shed light on this issue by highlighting the security vulnerabilities present in the available binaries of popular Operating Systems.

Project Goals

Based on our research and for the purpose of this paper, we aim to analyze the prevalence of exploit mitigation techniques in the binaries of the most recent versions of Ubuntu (version 18.04.3) and Windows 10. While some work on exploit mitigation techniques has been done on Linux only systems and some embedded systems, there has been no body of work done on analyzing the availability of these techniques on other systems such as on Windows Operating Systems. In addition to this, our paper will be the first to analyze the prevalence of mitigations in individual binaries of these operating systems. This is incredibly important especially since many of these binaries include important and popular programs, applications and files. By analyzing the security in the binaries of each OS, we aim to identify specific important binaries that have significant security vulnerabilities. By shedding light on these vulnerabilities, we hope that this information will be used in future patches and updates in order to make these operating systems more secure. Following this, we also provide a comparison of the implementation of mitigation techniques between the Ubuntu and Windows Operating Systems, which has not been done before. The paper will also be the first comprehensive analysis that will be performed on the deployment of exploit mitigation techniques on different versions of GCC.
Based on our analysis and and at the conclusion of our findings, our main goal for the project is that it can serve as a resource or guide for security professionals, developers, researchers or industry professionals in their work and spread awareness on the security vulnerabilities present in popular operating systems.

Related Work

While not much work has been done on this topic, these are the following works that helped inspire this paper.

1) There has been some work done on the challenges in designing exploit mitigations for deeply embedded systems: https://www.syssec.ruhr-uni-bochum.de/media/emma/veroeffentlichungen/2019/04/18/uArmor-EuroSP19.pdf
2) There has also been work done on Linux only systems: https://capsule8.com/blog/millions-of-binaries-later-a-look-into-linux-hardening-in-the-wild/
3) Exploit Mitigation Techniques in Linux Systems: https://7h3ram.github.io/2012/exploit-mitigation-techniques-on-linux.html

Design and Implementation

1. Availability of mitigation techniques in Compilers

At the conclusion of our research, the next step was to find the prevalence of each of these techniques in compilers, namely GCC and Clang. When compiling code, developers may have a number of reasons to pick a certain compiler or different version of a compiler. However, the main reason is usually due to convenience without much regard for security. Though some versions of compilers provide more security through the availability of mitigation techniques, other versions our used for greater convenience but at the expense of security. By detailing the mitigations available in each version of these compilers, we aim to provide a guide to developers and security professionals in order to promote more awareness on versions that have greater security and to make it easier for them to base their decision to compile secure code. Based on the results obtained from our analysis, we then provide recommendations on what versions developers should use when compiling code.

1.1 GCC

The following section provides an overview of the availability of mitigation techniques in different versions of GCC.The GNU compiler collection (GCC) is an incredibly popular compiler system used to compile source code. In order to find the security techniques currently available in each particular version of GCC, we first created a simple test program and ran it through that compiler (For example, in order to check if PIE was available in GCC 7.4 we executed the following command: *gcc -no-pie test1.c -o test1*). We followed this process for each version of GCC and for each of the available mitigation techniques.

Exhibit 1 illustrates the availability of each mitigation technique in various versions of GCC. At the conclusion of our analysis and as seen in *Exhibit 1*, we noticed that every version after GCC 3.4 had all five mitigations available while versions prior to 3.4 were missing some or all of the mitigations.

Therefore, we conclude that developers who use GCC versions 3.4 and prior for project development do so at the expense of security. Based on our analysis, developers who use GCC

to compile code should use GCC versions 4.1 and after for projects with increased security and fewer vulnerabilities.

GCC VERSION	MITIGATION TECHNIQUE				
	PIE	NX	SSP	Fortify_Source	Relro
3.0	X	X	X	X	X
3.2	X	X	X	X	X
3.4	✓	✓	X	✓	✓
4.1	✓	✓	✓	✓	✓
7.4	✓	✓	✓	✓	✓
8.3	✓	✓	✓	✓	✓
9.2	✓	✓	✓	✓	✓

Exhibit 1: Availability of Mitigation Techniques by GCC Version

1.2 Clang

Clang is a compiler front end for the C, C++, Objective-C and Objective-C++ programming languages, as well as the OpenMP,OpenCL, RenderScript and CUDA frameworks. It is an open-source software and uses the LLVM compiler infrastructure as its back end and has been part of the LLVM release cycle since LLVM 2.6 [8].

It is designed to act as a drop-in replacement for the GNU Compiler Collection (GCC), supporting most of its compilation flags and unofficial language extensions. Its contributors include major companies such as Apple, Microsoft, Google, ARM, Sony, Intel and Advanced Micro Devices (AMD).

Released in 2009, Clang incorporates all the mitigation techniques implemented by recent versions of GCC in addition to Control Flow Integrity (CFI) and ASLR [8].

The decision to choose which compiler to use is based on developer preference, however Clang is known to be faster and uses less memory when compared to GCC as well as being as secure when compared to most recent versions of GCC.

2. Availability of mitigation techniques in Operating Systems

The following section explains our data collection methodology used to identify the availability of mitigations in Ubuntu and Windows operating systems. For each of the operating systems, we first provide an overview of the availability of mitigation techniques in the total number of binaries available in each Operating System, following which we showcase the results received from our data collection methodology. Following this, **Section 3 and 4** in the evaluation section of this paper provides an in-depth analysis into the availability of mitigations in individual binaries of each OS.

2.1 Ubuntu

Ubuntu is a free and open-source software Linux distribution that is based on Debian [9]. While Ubuntu aims to be secure by default, there continues to be a number of security issues identified that have been patched by each subsequent version of the software over the years. This section focuses on Ubuntu 18.04.3, which is the most recently released version of the software, and focuses on our data collection methodology used to extract the available binaries in this OS as well as the process used to identify the number of binaries that did not contain all or some mitigations.

2.1.1 Data Collection Methodology

For extracting the availability of mitigations in Ubuntu binaries, we first developed a bash script that checks the availability of mitigation techniques in all the binaries of Ubuntu 18.04.3 with the help of the checksec tool. The output of the script provides the total number of binaries present in each folder, the availability of mitigations in each binary, the type of mitigations enabled in each binary (For example, for RELRO - Full RELRO, Partial RELRO or No RELRO) and splits up the binaries based on the type of mitigation available. *Exhibit 2* illustrates the results of the availability of mitigation techniques in the total number of binaries in Ubuntu 18.04.3. For each mitigation technique, the table provides information about the total number of binaries that have that mitigation enabled, the total number of binaries that do not have that mitigation enabled and the total percentage of binaries that have that mitigation enabled.

Section 3 in the Evaluation section of this paper provides an in-depth analysis into the security of individual binaries of Ubuntu 18.04.3.

Mitigation Technique	Enabled in Binaries	Not Enabled in Binaries	Percentage with Mitigations Enabled
NX BIT	96206	11553	89.27%
FORTIFY SOURCE	83784	23975	77.75%
STACK CANARIES	84514	23245	78.42%
RELRO (FULL/PARTIAL)	69080 (Full Relro) 26867 (Partial Relro)	11812	64.10%(Full Relro) 24.93%(Partial Relro)
PIE	62292*	10343*	85.76%
TOTAL NUMBER OF BINARIES	107759		100 %

*Tool error received for 33421 binaries.

Exhibit 2

2.2 Windows 10

The Windows Operating System produced by Microsoft is used by millions of people across the world. This section focuses on Windows 10, which was released in 2015 as a successor to Windows 8.1 and is currently the most recent version of the operating system [10]. Windows receives new builds and updates on an ongoing basis like Ubuntu, but differs in that it is not an open-source software. Similar to Ubuntu, there continues to be security issues identified with Windows that Microsoft combats by providing regular patches and updates to its users.
This section explains our data collection methodology used to extract the available binaries in Windows 10 as well as the process used to identify the number binaries that did not contain all or some mitigations.

2.2.1 Data Collection Methodology

To analyze the mitigation exploit techniques of Windows 10 binaries, we used the *wincheckseс* tool, which provides information on the availability of mitigations techniques of a binary. We then developed a script using the scripting language 'vbscript' to execute on all the binaries of the windows operating system. The script provides the output of each existing binary of the windows operating system and the availability of mitigation techniques on each of these binaries. We have included ASLR and Control Flow Guard in our analysis, which are the mitigation techniques (among others) that are most commonly used in Windows 10.
Exhibit 3 illustrates the results of the availability of mitigation techniques in the total number of binaries in Windows 10. For each mitigation technique, the table provides information about the total number of binaries that have that mitigation enabled, the total number of binaries that do not have that mitigation enabled and the total percentage of binaries that have that mitigation enabled.
Section 4 in the Evaluation section of this paper provides an in-depth analysis into the security of individual binaries in Windows 10.

Mitigation Technique	Enabled in Binaries	Not Enabled in Binaries	Percentage with Mitigations Enabled
ASLR	4899	1323	78.74%
NX BIT	4959	1263	79.70%
STACK CANARIES	3794	2428	60.98%
CONTROL FLOW GUARD	3487	2735	56.04%
TOTAL NUMBER OF BINARIES	6222		100 %

Exhibit 3

Evaluation

The following section evaluates the results received for specific binaries in each of the operating systems in order to provide a more detailed look into how secure they are. This section identifies several core binaries, popular applications and programs that do not have the appropriate mitigations in place. Following the results obtained from our data collection methodology, we then analyze the security of each directory and highlight major security vulnerabilities associated with those directories.

3. Availability of Mitigation Techniques for specific binaries in Ubuntu 18.04.3

The following section provides a detailed overview of the results received by the script for specific binaries of critical importance and the availability of mitigations. As seen in the following sections, we have identified a significant number of binaries that do not have even basic mitigations in place which leads to a number of security vulnerabilities that can be taken advantage of by an attacker. For each directory, we have provided the results of our data collection methodology that is split between each mitigation. The results also showcase the total number of binaries available in each directory and the total percentage of each mitigation enabled in the binaries for that directory. Finally, we also provide a composite total security score for each directory based on the results, which is computed by taking the aggregate of the percentages of mitigations enabled. Based on this score, we analyse how secure each directory is.

3.2.1 /bin

The bin directory is an important directory that contains certain executable programs that are necessary in single user mode to bring the system up or repair it. For example, Busybox is a program in the bin directory that can perform the actions of many common unix programs, such as ls, chmod, wget, cat, etc. It is most commonly used in Linux on embedded systems due to its small executable size. We identified Busybox as a program without any mitigations in place, which can be a huge security issue. Based on our results and with a security score of 98.97%, the bin directory is more secure especially when compared to other directories. However, a vulnerability in a single program or process can be exploited to cause significantly adverse consequences in a system. Take for example the results identified for Busybox, (*//bin/busybox* = *Partial RELRO* | *No Stack Canary* | *NX enabled* | *No PIE* | *FORTIFY SOURCE: No* |). As noted earlier, Busybox is an important program that interacts with multiple other programs and processes and has virtually no mitigations in place, which is a significant vulnerability. Similarly, we have identified six binaries without some or all mitigations in place that can be exploited by an attacker. *Exhibit 4* illustrates the results of our findings.

Mitigations	FS	NFS	SC	NSC	FREL	PREL	NO_REL	NX	NO_NX	PIE	NO_PIE	SECURITY SCORE
No. of Binaries	114	2	113	3	115	1	0	116	0	115	1	98.97%
% Enabled	98%		97%		100%			100%		99%		

Exhibit 4

3.2.2 /lib

The Linux kernel is a monolithic kernel with various loadable modules. These modules contain parts of the kernel used typically for device drivers, file systems and network protocols. In most cases, necessary kernel modules are loaded automatically and dynamically without administrator interaction. *Exhibit 5* illustrates the results of our findings.

Based on these results and with a security score of 20%, we conclude that the lib directory has a significant number of security vulnerabilities without any mitigations in place that can be easily exploited by an attacker. Based on our results, we have identified that most of the firmware, build and kernel lib modules in this directory have a majority of mitigations not enabled. The results of our findings further showcases that for all the binaries in the lib directory, only 3% of the total number of directories have Fortify, Relro and NX mitigations enabled which means that there is a significant number of individual binaries in this directory that can be easily exploited by an attacker.

Mitigations	FS	NFS	SC	NSC	FREL	PREL	NO_REL	NX	NO_NX	PIE	NO_PIE	SECURITY SCORE
No. of Binaries	289	10823	8008	3104	146	153	10813	298	10814	102	7	20%
% Enabled	3%		72%		3%			3%		94%*		

*Tool error received for remaining files

Exhibit 5

3.2.3 /boot

Boot is an important folder in Linux that contains all the boot related information files and folders such as grub.conf, vmlinuz image aka kernel etc. Based on our findings and with a security score of 0.003%, we conclude that the boot directory is an extremely serious security risk with a significantly large number of binaries identified without even basic mitigations in place. Even more significant is the fact that out of the total number of directories available in the boot directory, less than 1% of total binaries have either FORTIFY or stack canaries mitigations enabled (although without RELRO, NX BIT or PIE enabled), while an astounding 99.997% (267 total binaries) have virtually no mitigations in place. *Exhibit 6* illustrates the results of our findings.

To illustrate the significance of this vulnerability take for example the GRUB bootloader in the boot directory, which contains a large number of modules that provide additional capabilities (similar to the modules used by Linux). The GRUB bootloader is used to activate a GRUB module and the modules are typically stored under the */boot/grub/i386-pc/* directory. Within this directory, there are a large number of binaries with important functions that do not have some or all mitigations enabled. For example, the '*//boot/grub/i386-pc/file.mod'* stored in this directory is a module that provides basic file I/O functions. For this specific module, we have identified that this file does not have FORTIFY, stack canaries, NX BIT and RELRO mitigations enabled, which makes it vulnerable to multiple forms of attack.

Mitigations	FS	NFS	SC	NSC	FREL	PREL	NO_REL	NX	NO_NX	PIE	NO_PIE	SECURITY SCORE
No. of Binaries	1	268	2	267	0	0	269	0	269	0	1	0.003%
% Enabled	0.004%		0.007%		0%			0%		0%*		

*Tool error received for remaining files

Exhibit 6

3.2.4 /sbin

Similar to the bin directory, the sbin directory holds commands that are needed to boot the system. However, these commands are usually not executed by normal users and instead requires that the superuser (root) privileges are required to execute the binaries. With a security score of 98.2%, the sbin directory is highly secure with a majority of binaries containing either all or multiple mitigations enabled. However, we have identified certain important commands that do not have some mitigations enabled. While a significant number of binaries in the sbin directory have all mitigations enabled, a single vulnerability can be exploited to successfully perform an attack. For example, we have identified the following commands that do not have stack canaries enabled: *capsh, findfs, fstab-decode, ldconfig.real, partprobe, pivot_root, regdbdump and Setcap*. *Exhibit 7* illustrates the results of our findings.

Mitigations	FS	NFS	SC	NSC	FREL	PREL	NO_REL	NX	NO_NX	PIE	NO_PIE	SECURITY SCORE
No. of Binaries	112	1	105	8	112	1	0	113	0	113	0	98.2%
% Enabled	99%		93%		99%			100%		100%		

Exhibit 7

3.2.5 /snap

The snap directory is where the files and folders from installed snap packages appear on a computer system by default. Some of the popular Snap packages are htop, VLC, Minecraft server, WebDM, Freecad and many others. As seen in *Exhibit 8* and with a security score of 91.1%, the snap directory contains a significant number of security vulnerabilities, further showcased by the fact that a large number of binaries have many or no mitigations enabled. This is a serious security risk, considering that this directory contains programs and applications that are incredibly popular and used by millions of people. For example, we have identified that many libraries of Atom IDE as well as the VLC application do not contain FORTIFY, stack canaries and PIE mitigations.

Mitigations	FS	NFS	SC	NSC	FREL	PREL	NO_REL	NX	NO_NX	PIE	NO_PIE	SECURITY SCORE
No. of Binaries	40617	6181	36647	10151	23035	23601	162	46684	114	17812	9571	91.1%
% Enabled	87%		78%		99.65%			99.76%		65%*		

*Tool error received for remaining files

Exhibit 8

3.2.6 /usr

The ‘usr’ directory is a critical directory that contains the largest amount of data in a system that contains all the user binaries and their documentation, libraries and supporting directories. It consists of several subdirectories that contain additional UNIX commands and data files and is also the default location of user home directories.

With a security score of 91.3%, this directory has a large number of security vulnerabilities which is especially problematic considering the importance of this directory. Based on our results, around 13% (6610 binaries) of total binaries are not protected by FORTIFY and 20% (9622 binaries) do not have stack canaries enabled. There are several commands that do not contain FORTIFY and stack canary protections such as the commands ‘asciitopgm’ (when

executed, converts ASCII graphics into a PGM image) and 'col' (used to filter out reverse line feeds and replace whitespace characters and tabs).
In addition to those, we have identified several grub modules that do not have the NX BIT mitigation enabled (such as *'//usr/lib/grub/i386-pc/file.mod' and '//usr/lib/grub/i386-pc/font.mod'*), several commands that do not have PIE enabled (such as *'//usr/bin/emacsclient26', '//usr/bin/python3.6' and '//usr/bin/etags26')* and several grub system modules that do not contain RELRO (such as *'//usr/lib/grub/i386-pc/terminal.mod', '//usr/lib/syslinux/modules/bios/hexdump.c32* and *'//usr/lib/syslinux/modules/bios/host.c32'*). *Exhibit 9* illustrates the results of our findings.

Mitigations	FS	NFS	SC	NSC	FREL	PREL	NO_REL	NX	NO_NX	PIE	NO_PIE	SECURITY SCORE
No. of Binaries	42642	6610	39630	9622	45665	3024	563	48899	353	44147	756	91.3%
% Enabled	87%		80%		98.86%			99.28%		98%*		

*Tool error received for remaining files

Exhibit 9

4. Availability of Mitigation Techniques for specific binaries in Windows 10

Similar to what has been done with Ubuntu 18.04.3, the following section provides a detailed overview of the results received from our data collection methodology. The following sections showcase that we have identified a significant number of Windows executables that do not have the necessary mitigations in place which indicates a number of security vulnerabilities. For each directory, we have provided the results of our findings for each mitigation technique. The results also showcase the total number of binaries available in each directory and the total percentage of each mitigation enabled in the binaries for that directory. As done for Ubuntu 18.04.3, we also provide a composite total security score for each directory based on the results, which is computed by taking the aggregate of the percentages of mitigations enabled. Based on this score, we analyse how secure each directory is. In the following sections, we have identified three directories of critical importance for our analysis.

4.2.1 C:/WINDOWS

The C:\WINDOWS directory (appears as C:\Windows in Windows 10), is the folder that contains the Windows operating system. Although every file in the C drive is technically used by the OS, the C:\WINDOWS folder contains the files that, in turn, contain the code to run the OS. This is an important directory as the majority of Windows system files are stored in C:\Windows, especially in subfolders like /System32 and /SysWOW64. However, system files can also be found scattered throughout user folders (like the appdata folder) and app folders (like ProgramData or the Program Files folders).
With a security score of 90.7% and based on the results of our findings, there are a number of vulnerabilities identified in this directory. While most of the operating system files contain the four mitigations, many of .NET Framework, system32 and SysWOW64 files do not have NX, Stack canaries, ASLR and CFG protections enabled. To provide a better understanding of this security risk, we have provided the following examples of important executables and tools that do not have any of the four mitigations enabled, which makes them incredibly vulnerable to an attack.

1) memtest.exe: An important diagnostic tool in Windows Memory that is used to troubleshoot a system's installed memory and is installed in Windows 7 by default. It can be run from the Boot Manager, or scheduled to run on startup.
2) winload.exe: An important tool whose main function is to load essential device drivers, as well as ntoskrnl.exe, which is a core part of Windows.
3) setup.exe: A core executable and the standard installation file used by Windows for InstallShield Windows applications, its main function is to install a variety of software packages and is run on command.

These are just a few examples of important tools identified that do not have any of 4 important mitigations in place. *Exhibit 10* illustrates the results of our findings.

Mitigations	NX	NNX	SC	NSC	ASLR	NOASLR	CFG	NOCFG	SECURITY SCORE
No. of Binaries	3539	116	3117	538	3540	115	3065	590	90.7%
% Enabled	97%		85%		97%		84%		

Exhibit 10

4.2.2 Program Files

Windows was initially only available as a 32-bit operating system and on these versions of Windows (even 32-bit versions of Windows 10, which are still available today), only a "C:\Program Files" folder can be seen. The main function of this core folder is that it is the recommended location where installed programs store their executable, data, and other files. With a security score of 31.6%, we have identified a large number of binaries in this directory that has serious security vulnerabilities with only a few or no mitigations enabled. *Exhibit 11* illustrates the results of our findings. As seen in the results, only 17% of identified binaries have Stack Canaries protections enabled while only 12% have CFG enabled. We have identified the following examples of key files and process of significant importance that are vulnerable to an attack.

1) CMake is an open-source, cross-platform family of tools designed to build, test and package software. CMake is used to control the software compilation process using simple platform and compiler independent configuration files, and generate native makefiles and workspaces that can be used in the compiler environment of your choice. This is an important building tool and it doesn't contain CFG mitigation technique.
2) The mozilla Maintenance Service does not contain SC, ASLR and CFG. Firefox and Thunderbird install an optional service called the Mozilla Maintenance Service which allows application updates to occur in the background, without requiring you to click Yes in the Windows User Account Control (UAC) dialog.
3) Beyond Compare is a time-saving tool in Windows used by a large number of companies to manage source code, sync folders and validate copies of data. This tool does not have stack canaries or CFG protections enabled.
4) The Git application commands (*"C:\Program Files\Git\usr\bin\awk.exe"*) do not contain any of the four mitigations.

5) A large majority of the Android jdk files (such as *"C:\Program Files\Android\jdk\microsoft_dist_openjdk_1.8.0.25\bin\java.exe"*) do not have stack canaries enabled.

Mitigations	NX	NNX	SC	NSC	ASLR	NOASLR	CFG	NOCFG	SECURITY SCORE
No. of Binaries	191	336	95	454	183	115	61	466	31.6%
% Enabled	36%		17%		61%		12%		

Exhibit 11

4.2.3 Program Files (x86)

On 64-bit versions of Windows, 64-bit applications are installed to the Program Files folder. However, 64-bit versions of Windows also support 32-bit programs. In order to make sure that 32-bit and 64-bit software do not get mixed in the same locations, Microsoft has implemented that installed 32-bit programs are stored in the "C:\Program Files (x86)" folder [7]. Similar to *Program Files*, this is a core folder in Windows with many important programs and applications. *Exhibit 12* illustrates the results of our findings.

With a security score of 61.2%, this folder is more secure than the Program Files on a 32-bit operating system. However, we were still able to identify a large number of security vulnerabilities and considering the important files present in this folder, can be a huge security risk. As seen in *Exhibit 12,* 55% (682 binaries) of the total binaries in this folder do not have stack canaries enabled while an astounding 73% (907 binaries) do not have CFG protections enabled. We have identified the following important programs with a large number of vulnerabilities and are huge security risks.

1) LinuxLive USB Creator: A free Microsoft Windows program that creates Live USB systems from installed images of supported Linux distributions. This tool does not have any (NX, SC, ASLR, or CFG) of the mitigations enabled, making it a huge security risk.
2) The Windows Media Player: An application used by thousands of people for viewing videos, we have identified that this application does not contain NX bit.
3) Most of the Microsoft Visual Studio IDE tools (Ex: *"C:\Program Files (x86)\Microsoft Visual Studio\2017\Community\Common7\IDE\vshost.exe"*) do not have stack canaries enabled.
4) Some of the adobe reader files (Ex: *"C:\Program Files (x86)\Adobe\Acrobat Reader DC\Reader\LogTransport2.exe"*) do not have CFG mitigation technique enabled.

Mitigations	NX	NNX	SC	NSC	ASLR	NOASLR	CFG	NOCFG	SECURITY SCORE
No. of Binaries	1094	141	553	682	1050	185	328	907	61.2%
% Enabled	89%		45%		85%		27%		

Exhibit 12

5. Comparing Windows 10 and Ubuntu 18.04.3

The following section will serve as a discussion based on the results received from our data collection methodology. *Exhibit 13* illustrates the comparison of the percentage of total number of binaries that have mitigations enabled for each of the Operating Systems.

For Ubuntu, a large majority of binaries have the NX bit, PIE and Stack Canaries enabled. However, the operating system is still devoid of Relro and Fortify protections. While there could be a number of reasons for not enabling these protections (either for convenience or performance), it is done so at the expense of security. By identifying the specific binaries that do not have mitigations in place, we hope this paper sheds light on those binaries and that developers of code take advantage of this information.
For Windows 10, we see a large majority of binaries are missing Stack Canaries and CFG protections. When comparing the common mitigations enabled in binaries of Windows with that of Ubuntu, we can see that Windows 10 is lacking in NX bit and Stack Canaries protections. Granted, these Operating Systems are considerably different with a complex structure of interrelated processes and functions which undoubtedly leads to different decisions being made on enabling mitigations. However, as stated with Ubuntu, any decision made on disabling these mitigations is done so at the expense of more secure code. By identifying the important individual binaries that lack mitigations in this paper, we hope this aids developers in finding any security vulnerabilities that may have previously gone undetected as well.

MITIGATION	UBUNTU 18.04	WINDOWS 10
NX BIT	89.27%	79.70%
STACK CANARIES	78.42%	60.98%
ASLR	-	78.74%
RELRO	64.10% (Full) 24.93% (Partial)	-
PIE	85.76%	-
CFG	-	56.04%
FORTIFY SOURCE	77.75%	-

Exhibit 13

Limitations

This paper was initially going to incorporate multiple other Operating Systems, such as macOS Catalina, Android N to Android P and iOS (9.0 - 13). However, as Masters students with no prior background or experience in this topic, a significant amount of time went into researching into exploit mitigation, extracting binaries of the operating systems as well as identifying which individual binaries had mitigations enabled/disabled as well as understanding the process to do so. Based on the success of this project and the results received however, we aim to incorporate other Operating Systems in future work that we will be working on following the submission of this paper.
We also aimed to incorporate the performance toll each mitigation had on a system in order to include a recommendation on an appropriate combination of performance and security, but as stated earlier the necessary spec tool required for this purpose was too expensive to purchase.

Conclusion

The paper was successful in identifying the important individual binaries of operating systems that contain vulnerabilities in order to shed light on the security risks associated with not enabling the appropriate mitigations and protections. We were also successful in identifying several core files, programs and applications that do not have these mitigations in place, which could constitute a serious security risk. We hope that this paper will be used as a guide for developers when compiling code or for vendors of products that don't necessarily have security as a priority when releasing new products.

As seen in our analysis, even extremely popular Operating Systems like Windows and Ubuntu have a large number of security vulnerabilities in their systems. While there may have been many reasons for not enabling these mitigations, we have reached out to the appropriate developers to find out regarding how these decisions are made when choosing to enable/disable the appropriate mitigations.

The paper was also successful in comparing the mitigations enabled between Windows 10 and Ubuntu 18.04.3 with some very interesting findings. We hope to compare these with other Operating Systems in the future which will undoubtedly provide some interesting findings as well.

References

1. *Rouse, Margaret, and Brien Posey. "What Is Computer Exploit? - Definition from WhatIs.com." SearchSecurity, searchsecurity.techtarget.com/definition/exploit.*

2. *Comprehensive Exploit Prevention. Sophos, www.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/Sophos-Comprehensive-Exploit-Prevention-wpna.pdf.*

3. *7h3rAm. "Exploit Mitigation Techniques on Linux Systems." Grey Matter, 10 July 2012, 7h3ram.github.io/2012/exploit-mitigation-techniques-on-linux.html.'*

4. *Lastnameholiu. "Control Flow Guard - Win32 Apps." Win32 Apps | Microsoft Docs, 2018, docs.microsoft.com/en-us/windows/win32/secbp/control-flow-guard.*

5. *Payer, Mathias. "Mitigations: Completeness/Effectiveness vs Performance." Random Ramblings of a Security Nerd, Nebelwelt, 7 July 2017, nebelwelt.net/blog/20170707-mitigation-panel.html.*

6. *"Standard Performance Evaluation Corporation." Order SPEC Benchmarks, www.spec.org/order.html.*

7. *Hoffman, Chris. "What's the Difference Between the 'Program Files (x86)" and 'Program Files' Folders in Windows?" How, How-To Geek, 2 Oct. 2017, www.howtogeek.com/129178/why-does-64-bit-windows-need-a-separate-program-files-x86-folder/.*

8. *Raynal, Fred. "Clang Hardening Cheat Sheet." Quarkslabs Blog ATOM, blog.quarkslab.com/clang-hardening-cheat-sheet.html.*

9. *"About the Ubuntu Project." Ubuntu, ubuntu.com/about.*

10. *"Windows 10 Features: What Is in Windows 10." Microsoft, www.microsoft.com/en-us/windows/features.*