Reagan Leonard

Dr. Svetlana Drachova

CPSC 3220: Operating Systems

21 Jun. 2020

Operating System Vulnerabilities

Introduction

In this paper, I will discuss the common vulnerabilities and exposures (hereafter referred to as CVEs) that pose a threat to modern-day computer operating systems. From the CVE website, the definition of the CVE project is "a list of entries—each containing an identification number, a description, and at least one public reference—for publicly known cybersecurity vulnerabilities.[1]" There is a slight but important difference between CVEs and CWEs (Common Weakness Enumeration). The CWE project is an ongoing list of security weaknesses that are commonly found in certain versions of both hardware and software that is similarly compiled by a community of software and hardware experts and users. The primary difference between the two is that CVEs are specific instances of CWEs, which represent the underlying design flaws themselves. So CVEs, or the instances of security issues, can be assigned a Common Vulnerability Scoring System (CVSS) score, which is a number on a scale of 0 to 10, where 10 represents a very severe/critical security vulnerability[2].

CWE-119 & CVE 2018-0825

One especially dangerous CWE is CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer. According to the official entry for this weakness, "Certain

[1] See cve.mitre.org in references
[2] See cvedetails.com in references

languages allow direct addressing of memory locations and do not automatically ensure that these locations are valid for the memory buffer that is being referenced. This can cause read or write operations to be performed on memory locations that may be associated with other variables, data structures, or internal program data. As a result, an attacker may be able to execute arbitrary code, alter the intended control flow, read sensitive information, or cause the system to crash.[3]" This CWE has a high likelihood of being exploited by hackers, according to the CWE website. An instance of this problem that has been reported is CVE-2018-0825. In this example, there is an integer overflow vulnerability that leads to a buffer overflow. The result of this buffer overflow (which is common in CWEs and CVEs) is that it can allow attackers to run arbitrary code on certain Windows systems. While CWE-119 was first officially reported in 2008, CVE 2018-0825 wasn't reported until late 2017. This goes to show how problems that are nearly a decade old can still resurface in bug-ridden software.

CWE-125 & CVE-2018-4160

CWE-125 is simply titled Out-of-Bounds Read. First reported in 2008, it is a vulnerability that allows attackers to access confidential information by reading data beyond the end or before the start of the intended buffer (hence, out-of-bounds read). It also has the potential to cause a crash because there may be nothing to stop the read operation in the out-of-bounds area (such as a NUL in a string). If nothing is there to stop the operation, a segmentation fault or buffer overflow is very likely to occur. So not only is this a security risk because private information could be accessed without authorization, but the operating system is also at risk of crashing at any time which could obviously lead to major problems if there was any unsaved data

[3] See cwe.mitre.org/data/definitions/119.html in references

in any programs that were running at the time of the crash. An observed example of this CWE is noted in CVE-2018-4160 which was an issue found in devices running macOS versions prior to 10.13.4. The bug allowed hackers "to execute arbitrary code in a privileged context or cause a denial of service (out-of-bounds read) via a crafted app.[4]" Similar to the previous example discussed above, CWE-125 was reported in 2008 and CVE-2018-4160 was not discovered and patched until 2018, a significant amount of time between when a vulnerability was discovered and when a major tech company was still dealing with it.

CWE-416 & CVE 2018-5344

CWE-416, called Use After Free, is a case where memory being used/referenced after it has previously been freed can cause all sorts of unexpected behaviors, such as a program crash, valid data being corrupted, or the execution of arbitrary code. A program crash can occur if "chunk consolidation occurs after the use of previously freed data, and then invalid data is used as chunk information.[5]" Data corruption may occur if the system reuses a portion of freed memory. In this scenario, the memory in question is allocated to another pointer validly at some point after it has been freed. The execution of unwanted code can be achieved by hackers if they force a function pointer somewhere inside the heap (as a result of some newly allocated data choosing to hold a class) to contain an address pointing to valid shellcode. One example of this is noted in CVE-2018-5344: "In the Linux kernel through 4.14.13, drivers/block/loop.c mishandles lo_release serialization, which allows attackers to cause a denial of service (__lock_acquire use-after-free) or possibly have unspecified other impact.[6]" On the CVE website, it is unclear who reported this problem, but on Linus Torvalds'—the creator of Linux—Github page, he has

[4] See cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-4160 in references
[5] See cwe.mitre.org/data/definitions/416.html in references
[6] See cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-5344 in references

committed the code to fix CVE-2018-5344. On this commit page, it seems that Torvalds suggests that one "范龙飞" reported the problem and then Torvalds wrote the code patch (which I think is just an extremely cool thing to have been a part of: to report a bug in Linux and have the creator himself be the one to fix it!)[7].

Definitions of Vulnerability Types

A denial of service (DoS) vulnerability's main purpose is to attempt to make a resource unable to function properly. DoS vulnerabilities are also usually triggered in order to get a single device to act as a 'bot.' These devices, which are all infected with the same virus or malware, can then be used in unison to launch a larger distributed denial of service (DDoS) attack on another separate device or system. This is often a large server or system that typically crashes during these DDoS attacks. In 2018, this vulnerability type accounted for 47.5% of Linux Kernel vulnerabilities, 37.27% of Mac OS X vulnerabilities, and 8.17% of Windows 10 vulnerabilities[8].

A code execution vulnerability is an OS weakness that can allow for a malicious party to remotely execute arbitrary (but sometimes harmful) code on a device. These hackers typically run code that triggers a command to be executed on the remote device. The command has the potential to cause a data leak to a private/third party server or even force the download of a dangerous application that could contain more malware. Over the entire course of time that CVEs have been tracked in detail (since 1999), code execution vulnerabilities account for the highest percentage of all CVEs (23.6% as of the end of 2019). It is also one of the most dangerous types of CVE because they can sometimes result in the targeted device completely shutting down and being permanently broken. In 2018, this vulnerability type accounted for

[7] See github.com/torvalds/linux/commit/ae6650163c66a7eff1acd6eb8b0f752dcfa8eba5 in references
[8] See cvedetails.com in references for these and all the following vulnerability type distribution percentages

1.7% of Linux Kernel vulnerabilities, 44.54% of Mac OS X vulnerabilities, and 17.51% of Windows 10 vulnerabilities.

An overflow vulnerability is a bug in OS code that may allow a hacker to access and potentially overwrite sensitive data and executable code on the target device. This type of OS vulnerability can usually be tracked down to the stack/heap buffers. Essentially, some flaw in the device's mechanism to prevent overflow allows the malicious party to remotely overwrite code in a certain memory location of the device. When this happens, the buffer loses the ability to limit how much code can be generated. This can result in anything from unexpected or unpredictable device behavior, system crashes, and/or potential data loss. In 2018, this vulnerability type accounted for 15.82% of Linux Kernel vulnerabilities, 31.81% of Mac OS X vulnerabilities, and 7.39% of Windows 10 vulnerabilities.

A memory corruption vulnerability is one that basically leaves a device's memory exposed to a hacker. The result of one of these attacks can range from unexpected or unpredictable device behavior to the crashing of the device. The source of this type of weakness lies in the memory component of a device and the main way a hacker goes about performing one of these attacks is by modifying the code associated with the device's memory. This clearly puts private information at risk of being corrupted. Memory corruption attacks are usually a 'foot in the door' for hackers, so to speak, and are often combined with a code execution and/or DoS attack. In 2018, this vulnerability type accounted for 5.1% of Linux Kernel vulnerabilities, 27.27% of Mac OS X vulnerabilities, and 0.38% of Windows 10 vulnerabilities.

A 'bypass something' vulnerability, sometimes referred to as a 'back door' in an OS, is a flaw that allows someone to circumvent at least one layer of protection on the device set up by

the user, an administrator, or by the OS, itself. This 'layer of protection' is typically whatever security authentication procedure a device has, hence the name of this vulnerability. If a password (set up by the user) is required to gain access to the device, a bypass attack could allow a hacker to exploit a workaround to gain access to the device. In 2018, this vulnerability type accounted for 2.26% of Linux Kernel vulnerabilities, 22.72% of Mac OS X vulnerabilities, and 15.17% of Windows 10 vulnerabilities.

A gain information vulnerability is exactly what it sounds like: a flaw in the operating system that allows a hacker to gain access to private information on the device. These types of attacks often result in the hacker accessing and exporting the victim's personal data or information that can be used to identify that user. Gain information attacks can be carried out using a multitude of malicious mediums, including but not limited to applications, webpages, or other programs. In 2018, this vulnerability type accounted for 11.3% of Linux Kernel vulnerabilities, 16.36% of Mac OS X vulnerabilities, and 28.02% of Windows 10 vulnerabilities.

A gain privilege vulnerability is also fairly self-explanatory: a flaw in the OS that allows attackers to achieve a certain permission level (or privilege) on the device. Similar to gain information attacks, gain privilege attacks typically result in the hacker accessing the target user's personal information (as this information may be used to actually gain the desired privileges). Again, this can be done via a malicious application, webpage, or other program. In 2018, this vulnerability type accounted for 1.7% of Linux Kernel vulnerabilities, 1.81% of Mac OS X vulnerabilities, and 0.38% of Windows 10 vulnerabilities.

Concentration & Lifetime of OS Bugs

From the paper *An Empirical Study of Operating Systems Errors*, researchers found that, "device drivers have error rates up to three to seven times higher than the rest of the kernel[9]." They also reported that the largest 25% of functions in an OS---that is, the 25% of functions that are longest in length---possess, "error rates two to six times higher" than the smallest 25%. In addition, the newest 25% of OS files recorded "error rates up to twice that of the oldest quartile," which they claimed, "provides evidence that code 'hardens' over time." As for the lifespan of a bug, the authors of the report also discovered that, "bugs remain in the Linux kernel an average of 1.8 years before being fixed." That is a significant length of time, to say the least; especially considering how widely-used and thoroughly tested the Linux kernel is as a piece of software! Lastly, the researchers note that bugs are often found in clusters within OS code. The paper offers a few reasons why bugs may cluster together. Here they are in order from most to least common: 1) ignorance - programmers unknowingly create clusters of bugs because they are unaware of the system rules; 2) incompetence - simply put, the sections of code that bad programmers work on are more likely to be bug-ridden; 3) cut-and-paste errors - oftentimes, programmers believe that code that runs and performs the desired function is correct. However, if a programmer copies and pastes code that is not error-proof or pastes it in a location that it was not intended for, it can be a source of a cluster of errors; 4) lack of testing - some code is not tested nearly as often as other sections of code, therefore making it much more likely to contain clusters of bugs.

[9] See dl.acm.org/doi/pdf/10.1145/502034.502042 in references for all quotes in this paragraph

References

"Common Vulnerabilities and Exposures (CVE)." *CVE*, 2020; cve.mitre.org/.

"Current CVSS Score Distribution For All Vulnerabilities." *CVE Security Vulnerability Database. Security Vulnerabilities, Exploits, References and More*, 2020; www.cvedetails.com/.

"Common Weakness Enumeration." *CWE*, 19 Feb. 2020; cwe.mitre.org/data/definitions/119.html.

"Common Weakness Enumeration." *CWE*, 20 Feb. 2020; cwe.mitre.org/data/definitions/416.html.

"CVE-2018-4160." *CVE*, 2 Jan. 2018; cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-4160.

"CVE-2018-5344." *CVE*, 11 Jan. 2018; cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-5344.

Torvalds, Linus. "Loop: Fix Concurrent lo_open/lo_release · Torvalds/Linux@ae66501." *GitHub*, 5 Jan. 2018; github.com/torvalds/linux/commit/ae6650163c66a7eff1acd6eb8b0f752dcfa8eba5.

Chou, Andy, et al. "An Empirical Study of Operating Systems Errors." *ACM Digital Library*, 2001; dl.acm.org/doi/pdf/10.1145/502034.502042.