Jeremy Holloway
CPSC-3220-001
5/19/2019
Dr. Smotherman

Project #1

1. A Common Vulnerability and Exposure (CVE) is a reference to a either a vulnerability or an exposure, which if exploited can have negative repercussions for the system. A vulnerability is a logical error in the computations performed in hardware or in software, but can usually be resolved with changing code or components. An exposure is a configuration issue or some mistake in software, however an exposure is merely a stepping stone to causing a vulnerability so they are dealt with in different ways. A vulnerability is far more serious than an exposure, but doesn't make an exposure not an important issue to track as exposures often lean to vulnerabilities. A CVE is identified by a unique number assigned by the CVE Numbering Authorities (CNA) from across the globe. A Common Vulnerability Scoring System is used to give the CVE a score that provides a qualitative standard that most users would be familiar with. Most people associate red as a severe rating for a risk assessment and yellow for a moderate rating, and a higher score being more severe than a lower score. The CVSS score combines these to provide a quick glance of what CVE is more noteworthy than another and it's importance among others [1].

2. Definitions for vulnerability types
   - Code Execution: A code execution is when a software system allows uploading of file extensions and allows for execution if the file is executable [2].
   - Overflow: An overflow occurs when a function allows data to be copied without performing any boundary checking [2].
   - Memory Corruption: A memory corruption is an error in code allowing an attacker to cause corruption to various memory locations in order to crash the system [2].
   - Bypass Something: Bypass something allows an attacker to bypass some sort of security check in order to gain the same level of privileges as an authenticated user [2].
   - Gain Information: Gain information is a software vulnerability that is exploited to gain some information that may not cause a system crash but could be useful in doing so later [2].
   - Gain Privileges: Gain privileges is when an attacker gains access to high-level privileges while skipping through the user authentication process and executing instructions [2].

3. I would follow the layout of Rex Plantado's article because it has a nice general flow with simplistic concepts to follow. Being a novice to operating systems but familiar with the terminology and some conceptual practices made it very easy to follow. I would try to emulate this type of linear format with a few stops along the way to explain concepts. While someone with no experience in software and coding might find this article hard to read, but I don't think the author wrote it in a style conducive to that audience. I would also make the assumption that anyone who would take the time to read this material would be knowledgeable enough that I could write with some common knowledge expected. The diagrams were a very nice feature to help follow the structure of the attack chain since macros are a subject that I don't

fully understand. The visual aids he used were very simple and made the read more engaging and informative at the same time. While he did have some complex code samples they were usually prefaced with some explanation that they were intentionally made that complex to obfuscate their true purpose. Overall, I found the article interesting and I felt like I learned something new afterwards [3].

Kai Lui did have one really good idea that I might incorporate of what he called a "quick look," which served as a pseudo abstract. I thought it was good way to quickly give someone an idea of what they were about to read without being too dense. I think his overall approach was too dense for someone unfamiliar with the intricate aspects of operating systems. It was helpful to try to understand his process and the responses, but the jargon and code snippets were a little much for a novice audience. I would try to make a simpler explanation rather than a full detailed analysis as it makes for a hard and uninteresting read. The shear amount of examples and samples made navigating the article very cumbersome, and overly complex. I could imagine that a very knowledgeable person in this subject might enjoy the peer audience level that the author chose, but it was lost on me. I did like a few of his examples but overall I feel more confused about what I just read than when I started [4].

4. Chou makes an argument that most bugs are located in the drivers since the majority of drivers are written by people whose expertise lies in the device rather than operating systems. This means the programmer wouldn't be familiar with the systems calls of the operating system and would more prone to causing errors. The author describes the distribution of the bugs as monotonic, so the more files the less bugs and vice versa. This makes sense to anyone who writes code as writing helper functions and splitting code into multiple files makes bugs and logic errors easier to spot and correct on the spot. Chou averages the lifespan of a bug to be 1.8 years with a median of 1.25 years. Bugs tend to cluster around each other because the person who made the error would be more likely to make other errors so it makes sense that there would be others. Chou backs this up with various statistical models, but does make clear that bugs are not always dependent. He also concludes that programmers often fall into a common pitfall of working code is good code so if something works as intended it is rarely checked again for errors and merely copied until a problem arises [5].

Palix followed up with the study by Chou by rechecking if his results still held,, and many still do. Drivers remained the main category of faults, but seeing as how drivers make up over half of the Linux kernel source code it would make sense that it would half the largest percentage of errors. The life span remained about the same as well with some fluctuations a new developer tools allowed for greater error checking. The Hardware Application Layer (HAL) managed to achieve the highest rate of faults, which was previously held by drivers as well. All in all Palix managed to show that operating systems have a lot of issues that need to be fixed, but most of all good programming practices

Bibliography

[1] Anat Richter, "*What Is a CVE Vulnerability And How To Understand Its Details*," blog, 7 Jan. 2019; https://resources.whitesourcesoftware.com/blog-whitesource/what-is-cve-vulnerability

[2] Alan Litchfield and Abid Shahzad, "*A Systematic Review of Vulnerabilities in Hypervisors and Their Detection*," Proc. 2017 Americas Conference on Information Systems (AMCIS 2017)

[3] Rex Plantado, "*Analysis of a targeted attack exploiting the WinRAR CVE-2018-20250 vulnerability*," blog, 10 Apr. 2019; https://www.microsoft.com/security/blog/2019/04/10/analysis-of-a-targeted-attack-exploiting-the-winrar-cve-2018-20250-vulnerability/

[4] Kai Lu, "*Detailed Analysis of macOS/iOS Vulnerability CVE-2019-6231*," blog, 24 Jan. 2019; https://www.fortinet.com/blog/threat-research/detailed-analysis-of-macos-ios- vulnerability-cve-2019-6231.html

[5] Chou, et al., "*An Empirical Study of Operating Systems Errors*," Proc. 2001 ACM Symp. Operating Systems Principles (SOSP 01), pp. 73-88.

[6] Nicolas Palix, Gaël Thomas, Suman Saha, Christophe Calvès, Julia Lawall, and Gilles Muller. 2011. "*Faults in linux: ten years later*." In Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems (ASPLOS XVI). ACM, New York, NY, USA, 305-318. DOI: https://doi.org/10.1145/1950365.1950401