**Essay Paper: Buffer Overflows Attack**

**Introduction:**

A buffer overflow occurs when a programmer writes more data in memory that exceeds its space that is allocated in a given program. According to Nashimoto, Homma, Hayashi, Takahashi, Fuji, & Aoki (2017), a programmer is responsible for managing a given program's memory to avoid flaws that can be leveraged by the attackers. In simple terms, data overflows take place when data input overruns the existing memory capacity allocated in a given program. As a program attempts to write the data to the buffer, it overwrites to the adjacent memories. Alouneh, Bsoul, & Kharbutli (2016) discuss that data overflow is common in all programs and is triggered by problems such as allocating small space to accommodate data or input of malfunctioned data. Buffer overflow attacks are one of the most challenging attacks to execute and analyze. Programmers should check errors during the program to ensure that buffer is allocated the right type and right amount of data. Failure of counterchecking errors during program writing can attribute to program bugs, which can be leveraged by attackers to hack the system or program.

**Buffer Overflows Attack:**

A program's memory is divided into different segments, including stack, heap, and buffer. According to Nashimoto et al. (2017), the buffer is the memory storage region that holds a program's data as it is transferred to another location. The buffer overruns occur when a given program attempts to fill a buffer with more data that was not designed to hold. It occurs as a result of simple errors that are unavoidable and easily noticeable in the programming process. Therefore, attackers can exploit such coding errors to gain access to the system. The data overflow attackers can present harmful consequences to the organization as it can expose its data or customers'.

Consequently, it can create a platform where malicious hackers can steal critical information from the organization, such as credit numbers or bank accounts. Alternatively, it can attract legal suits, thus presenting financial challenges for the organization. Alouneh, Bsoul, & Kharbutli (2016) advocate that programmers can effectively mitigate buffer overflow vulnerabilities by understanding what a buffer overflow, the dangers it can present to the application or system, and attackers that hackers use to exploit these vulnerabilities.

**Search Procedures:**

Defining research procedures is critical when evaluating buffer overflow attacks. To identify what, how, why, where, and who related to buffer overflow attacks, a literature review was carried on the previously written articles that addressed the issues of buffer overflow attacks. The articles were searched from the university databases. In this case, a given criterion was used to determine the eligibility of the hacking cases published. One of the criteria used to evaluate the articles is determining the type of cyber-attacks. All articles that discussed other types of cyberattacks, such as phishing, were excluded from the research. In this regard, only ten articles were searched from the university database. After critical evaluation, only four articles met minimum requirements. All four articles addressed different buffer overflows that took place in different countries with certain time frames. In the search process, the Mesh terms were combined with words such as "and" and "or." The mesh term was filled in the search button, where the best articles were selected for the research.

**Data Analytics:**

After the data analysis, it is evident that buffer overflows more common while designing and building software. The data reveals that the attackers take opportunities for errors made during a software or system development phase. Failure of the programmers to countercheck errors when coding poses a huge opportunity for the attackers to gain access to the software, system, or device, thus stealing more sensitive information. Some of the cases of data attacks are as follows.

**The Blaster Worm:**

In 2003, approximately 100,000 Microsoft window operating systems (OS) were reported to have been infected with blaster worm damage, which cost millions of US dollars to different institutions, organizations, and people. Despite Microsoft's efforts to clean up the malware from the infected operating system, the worm persisted. On August 11, 2003, the Blaster worm was noticed while already spreading where the number peaked in august 13 in the same year (Bailey, Cooke, Jahanian, & Watson, 2005).

The blaster worm was executed after an attacker identified a flaw in Microsoft windows coding. According to the statement issued by Microsoft company, the buffer overflow in the Windows remote procedure call (RPC) created a gap that allowed attackers to put and run arbitrary code. The Last Stage of Delirium (LSD) group discovered this flaw, which affected most Windows OS, including XP and NT. 4.,2000. The attacker deliberately fed a carefully crafted data into the Microsoft Windows operating system, causing it to store codes large than the buffer allocated, overwriting the adjacent memories (Bailey, Cooke, Jahanian, & Watson, 2005). The attacker replaced the RPC code with the know executable codes, which changed the ways Microsoft operating system was expected to work. It led to most windows operating systems across the world, not responding to the users' command or eventually shutting the computers down. The attack led most companies to close down while others lost their data.

After a keen investigation, it was found that 18 years old Jeffrey Lee Parson was the person behind an attack. Lee Parson admitted to having created and executed the B variant of the Blaster worm. The attacker implemented an attack to show off his intellectual ability (Bailey et al.,2005). The attacker was only interested in challenging Microsoft company by exposing its flaws in its Windows operating system and seeking public attention. The attacker targeted Microsoft company by exposing the security gaps in its products. The worm targeted the windows operating system.

**2004 AOL Instant Messenger (AIM):**

Another buffer overflow attack was executed in 2004 AOL instant messenger (AIM). The attackers ran an attack after realizing a security gap in AOL instant messenger. The attacker ran data larger in size than the AOL AIM buffer, overriding information in the close memories (Robinson, Carey, Longe, & Parr, 2004). The attack affected all AOL users where they were unable to execute commands. The vulnerabilities existed in the AIM client software handle 'away' messages. A vulnerability in AIM allowed an attacker to execute an overly long input to the 'go away' function of the aim URL handlers crashing the system. The attack exposed the user's data.

After an analysis, it was found the attacker was interested in exposing security gaps in the AIM software. The attackers were interested in challenging AOL for leaving security gaps in their systems. Like the previous case, the attackers were seeking public attention. In this case, the people behind the AIM attack were not masked. Despite the company making investigation, it was unable to find the attackers (Robinson et al., 2004. However, the company responded to the outcry by encouraging AOL 2004 users to update the software to the latest version, which would correct such bugs.

**Adobe Flash Player:**

Another incidence of buffer overflow attack is the Adobe Flash Player. According to Song, Kim, Kwon, Jin, Kim, Kim, & Kim (2019), the adobe player attack occurred in 2017 after a remote attacker executed an arbitrary code that targeted machines. Song et al. (2019) highlight that the led computers began functioning abnormally and shutting down. After the successful exploitation of Adobe Reader, they attempted to attack vulnerable systems and execute the malware on them. In particular, Adobe Flash Player had had a stack overflow that allowed attackers to execute their arbitrary data, which changed how the entire program was working. The attackers remained unrecognized, and their intentions are not known up to date.

**Morris Worm:**

Another example of a Buffer overflow attack is the Morris worm, which was the first cyber-attack ever to occur. The attack took place in 1988, where a malicious program was unleashed from the Massachusetts Institute of Technology (MIT). The attack affected at least 60,000 computers connected to corrupted Internet (Furnell, & Spafford, 2019). Unlike viruses, worms did not need the Internet to propagate itself. The attack affected various big institutions in the US, including Princeton, John Hopkins, Harvard, NASA, Stanford, and others.

The attacker leveraged a buffer overflow vulnerability in the Unix operating system, slowing its functions. The attacker used an arbitrary code to change the functionality of the Unix operating system. Robert Tappan Morris was a 23 years old Cornel University graduate, was behind the attack. He admitted to having created a program that slowed the functionality of the Unix program. However, Morris confirmed that he was an experiment that ran out of control.

**Conclusion:**

Buffer overflow attacks are one of the most challenging attacks to execute and analyze. Programmers should check errors during the program to ensure that buffer is allocated the right type and right amount of data. Organizations or programmers can mitigate buffer overflow attacks by running canary tests, ensuring that the source code is reliable. Secondly, organizations or individuals need to implement non-executable stacks or data executable prevention (DEP). According to Nashimoto et al. (2017), DEP prohibits attackers from injecting malicious code lines on the stack program. In the end, it allows the program to run successfully.

**References:**

Bailey, M., Cooke, E., Jahanian, F., & Watson, D. (2005). The blaster worm: Then and now. *IEEE Security & Privacy*, *3*(4), 26-31.

Robinson, B. A., Carey, C. A., Longe, M., & Parr, J. (2004). *US Patent No. 6,760,580*. Washington, DC: US Patent and Trademark Office.

Song, B., Kim, S., Kwon, E., Jin, S., Kim, J., Kim, H., & Kim, M. (2019). A Study on Artifact Grouping by Analyzing Artifact Case by Vulnerability: Using Adobe Flash Player Vulnerabilities. *Convergence Security Journal*, *19*(1), 87-95.

Furnell, S., & Spafford, E. H. (2019). The Morris Worm at 30. *ITNOW*, *61*(1), 32-33.

Nashimoto, S., Homma, N., Hayashi, Y. I., Takahashi, J., Fuji, H., & Aoki, T. (2017). Buffer overflow attack with multiple fault injection and a proven countermeasure. *Journal of Cryptographic Engineering*, *7*(1), 35-46.

Alouneh, S., Bsoul, H., & Kharbutli, M. (2016). A software tool to protect executable files from buffer overflow attacks. *International Journal of Internet Technology and Secured Transactions*, *6*(2), 133-166.