Academic Journal of Research and Scientific Publishing | Vol 2 | Issue 19
Publication Date: 5-11-2020 ISSN: 2706-6495

The Buffer Overflow Attack and How to Solve Buffer Overflow in Recent Research

Samah Muhammed S. ALHusayn
Student in cyber security master, Taif Universty, Taif 21944, Saudi Arabia
Email: sama7muhammed@gmail.com
Dr. Emad Alsuwat
Assistant Professor, Taif Universty, Saudi Arabia
Email: Alsuwat@tu.edu.sa

Abstract

Buffer overflow attacks, whether by software error or an attack, is one of the most important security problems that represent a common vulnerability of software security and cyber risks. In this paper, we will present simple concepts of the buffer overflow attack, its types, vulnerabilities, and a protection mechanism from exploiting vulnerabilities. In the last, it will be presented More than one way to detect and solve buffer overflow.

Keywords

Buffer Overflow, stack , heap , vulnerable , Recent studies, detect and solve , from templates and k-induction, SD2-C2M2, Fuzzing , code-level model, BOVP model ,Adaptable and Automated Static Analysis Framework.

1. Introduction

Buffer overflow attacks became known as one of the Morris Worm Web attack in 1988[3].Buffer overflow are the most popular type of vulnerability over the last decade This fact written in 2000[5] . and today buffer overflow threats is the most growing and severe form of vulnerability in software , it has become increasingly critical issues as network security[18].The buffer overflow problem is a root in software security as it pertains to the way the low-level computer works.

More of the buffer overflow issues are possibly the product of poor programming, and could be detected and resolved by the suppliers before the software was released if the suppliers had carried out basic checking or code corrections along the way.Buffer overflows[4].Buffer overflow is an phenomenon that happens when program passes data to a buffer overruns the buffer's space, causes in the overwriting of neighboring memory locations[2].When the program accesses memory addresses beyond specified memory regions, are considered to reason for a large of software vulnerabilities[6].The buffer issue classification has the largest percentage of vulnerabilities, nearly 14.08 per cent based on distribution of bugs in the June 2017 Document of (CNNVD) , owing to the different types of buffer overflow drawbacks, it is difficult to logically deduce buffer overflows and owing to the vast range of application root programming and complicated logical invock rates, several weaknesses in a time and typically detection software weakness involves could not be easily detection of source code and Boolean analysis [18].

The remainder of this article is structured accordingly. The definition of buffer is illustrated in section 2. In section 3 I discusses buffer overflow attack concept , types, vulnerabilities, and a protection mechanism from exploiting vulnerabilities. Section 4 view recent research to detect and solve the buffer overflow. The paper will be closed in Section 5.

2. Background

A buffer is a compact memory space within a device that can contain several cases of a certain data form[18].The buffer is often in RAM, which is an area to temporarily store physical memory of data when moved from one place to another and help improve performance, as most modern hard disks benefit from in accessing data efficiently[2]. Some internet services are based on buffers such as their use to prevent internet outages and changing the connection speed in the video stream so that part of the video is stored and is streamed from the buffer that is designed to

accommodate a specific amount of the program, otherwise when sending more than the buffer capacity will overflow to overwrite the neighboring memory data[2]this exceed called buffer overflow.

3. Buffer Overflow attack (BOF):

The ultimate purpose of a BOF attack been to thwart a feature of an illegal application so that the intruder can gain dominate of the system and controlling host if the application is adequately privileged[5].The intruder prepares to see the right code in the memory of the program and to transfer the program to it, by the necessary requirements installed into the registries & memory to locate the root program and run a code called "exec(sh)" to reach the root buffer[5].In view of the limitations which could be applied to a particular exe file, the efficient use of a stack-based buffer overflow involves a possible data address spill. In many forms, among the most common techniques is to find and exploit a string bug in a format, which effectively enables the stack to steal keys and is a weakness that enables an attacker to exploit the display output format [5].Buffer overflow attacks constitute a significant one of all security attacks where that buffers are so popular[5] and simple to exploitation[9].Today, the buffer overflow is increasingly difficult to exploit because of the many security mechanisms implemented by the operating system and at the runtime level[17]. There are three aspects of the buffer overflow attacks: code caching, overflow breach and service shooting down[17].Four kinds of overflow attacks are present, namely destruction of operation logs, destruction of heap data, alter method parameters and fixed buffer overflows[17]. Buffer overflow is a popular and very risky vulnerability, prevalent in different operating systems and programs[17].

3.1 The common type of Buffer overflow attack:

Buffer overflows categorized into stack based buffer overflows, integer overflows , heap-based buffer overflows and Unicode overflow.

3.1.1 stack based buffer overflows:

The stack is an arbitrary form of data that acts as the item array, of two key processes: push that inserts an item to the array and pop delete an item that inserted at the finally[18] based on LIFO method which means last in of data add to stack first out from it.

The stack frame is the data set for a single call for a subprogram in the stack[18]. Typically, a stack frame contains the return address, the stacked arguments, the local variables, and the stored copies of the sub-software updated registers[18].A modern structure for a stack is formed and placed Under a particular stack frame every time the function is called (shown in Firure1) [18].The EBP is the basic indicator for the recent stack frame, which marks to a high address at the top of the stack and doesn't alter while the ESP is the recent stack indicator which marks to the next offered byte of that stack and differs with operate implementation[18].A buffer overflow on the base of the stack happens when a program writes data that increases of the capacity that the stack assigns to the buffer at the stack memory[18].A buffer overflow on the base of the stack happens when a program writes data that increases of the capacity that the stack assigns to the buffer at the stack memory[18].It is the most popular kind of buffer overflow attack which means a buffer on the request stack is overflowed[2].

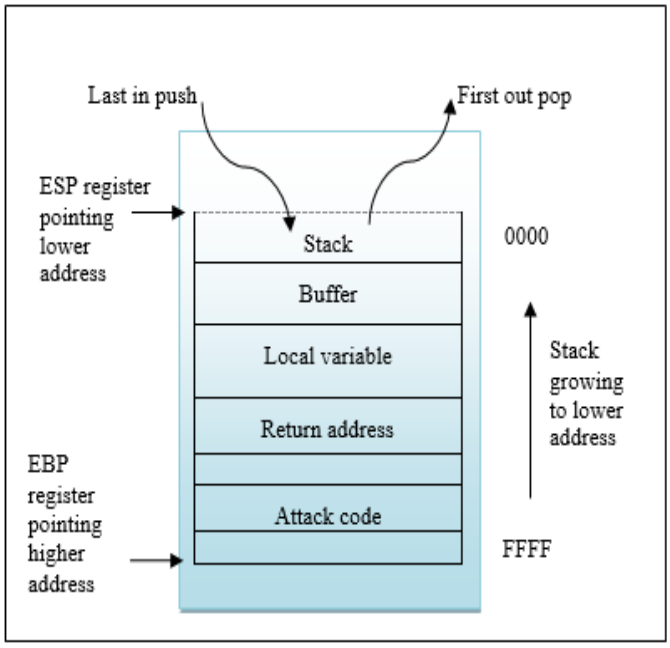

Figure 1. Buffer Overflow Attack

3.1.2 Integer Buffer Overflow :

Integer Buffer Overflow is the mathematical process of an integer overflow leads to an integer which is bigger for the integer that can hold[2].divided into three categories based on undersigned structural, symbolic and truncation issues underflow and overflow[18].The underflow of unmarked integer is due to the fact that unmarked entities are unable to accept negatives[18].

The problem with the code is induced by the comparison of marked integer, the process of signed integer, and the comparison of unmarked integer and marked integer and the shortness problem arises primarily when a large-order integer such as 64 bits is copied to a minimal-order integer such as 32 bits[18].

3.1.3 Heap buffer overflows:

The heap is a space which is assigned as the code executes [18]. The user usually demands space via malloc, fresh, etc. and manages the released space via the retrieved beginning address pointer[18]. It must be considered as part of the memory after every use, by free, delete, etc. ; otherwise it would produce a memory leak[18]. The heap blocks in the same heap are normally memorized parallel And when code is inserted into a heap stack, code overflowing exceeds the capacity of the heap stack, allowing the data overflow behind the heap block to protect the peers[18].This kind of target enemies data in the distributed memory called the heap[2].

3.1.4 Unicode overflow:

Unicode Buffer Overflow appears by adding Unicode characters into an entry expecting ASCII characters[20]. When ASCII code just includes Western language characters, Unicode can initial a character for all languages. Setup the software in Unicode phase to delete the Unicode buffer overflow and substitute cout, printf with wcout, wprintf alike[20].

3.1.5 Format-String Buffer Overflow:

Overflow of the format-string buffer happens with records and console. Often text strings are compiled into wider formats directly[20]. The formatted string is entered by sprintf in a letter array in the same way that 'printf' is used to insert the string in to the control information[10, 12].Many methods were suggested including runtime, debugging, and static analysis, but none can differentiate between all kinds of format string vulnerabilities. The analysis method defines the string features in the format. String processes typically include printf, fprintf, sprintf, etc. This strategy measures only the flaw, but decreases the false positive rate[20].

3.2 The Vulnerable Of Buffer Overflow attack:

Vulnerabilities are flaws and shortcomings when designing or executing an operating system or programs[21, 15].These vulnerabilities can make the program to crash and also compromise an intruder, which could bring system security concerns and harm and there are several causes why device insecurity evolves[11].

From the reports of international authoritative emergency organizations, the insecure nature of copy data over the scale of the target buffer resulted for approximately 1/2 of the overall vulnerability in 2018[14]. The build-up of a buffer overflow vulnerability contains three requirements: a memory allocation operation, the main requirement to buffer overflowing ; a buffer copying process with bigger space of the stack space capacity so the buffer allocation length the main reason of the issue[11].The vulnerability of the buffer overflow will be observed, if the program meets all the requirements specified previously[11].

3.3 Mechanisms protection for exploitation buffer overflow:

Buffer overflow was an important subject of study in computing background and software security , some causes have been discussed to avoid various exploitations[17]. We can observe several protection mechanisms that prevent overflow from occurring or respond when overflow occurs[13].

3.3.1 Data implementation preclusion (DEP) is used at Boolean basis and defines the right to implement at the location of the memory with blocks suspicious programming instruction from the buffer meaning implementation by requiring the implementation rights of just a particular memory spot[17].

3.3.2 Address Space Layout Randomization (ASLR) is defense mechanism done by an operating system and automatically resizes the binary and existing library memory address each time it is executed as just that, any attack during fixed defined values would failure [17].

3.3.3 Stack cookies and canaries : This approach applied by inserting a tiny chosen haphazardly binary within the system stack, in memory, directly earlier than the stack revisit indicator thus guarantees that the stack content is not distorted or overwritten from potentially malicious data supplied by the user The return in detector is altered when the buffer overflow writes stack space from the bottom to the upper end so updated stack [17].

3.3.4 Part or complete RELRO eliminates all dynamic related process and guarantee that is read-only Chart of Worldwide Offset (GOT). The attacker can no longer change outside request addresses to the protected storage register by having GOT inputs read-only of the stack[17].

3.3.5 Place Isolated Executable (PIE)

Is an alternative function can use at run time, rendering the executable behavior a dynamic external library at the time of connecting and installing. This function brings more pseudo random to the process of connecting and installing. Remember that ASLR predates PIE and ASLR does not allow PIE to be allowed[17].

4. The recent research to detect and solve the buffer overflow:

There are many studies and the inventions in a big and important subject in the field of software engineering and security related to the characteristics of the devices running it due to the gaps and weaknesses that cause cyber security violations, for example, the buffer overflow capacity in many way, I chose the following from them:

1- Inventor Murthy, Praveen research [16] explained a method to reduce buffer overflow problems as illustrated by the method summarized in the table.

2-Researcher display how to detecting buffer overflow via a synthesis of assertions from templates and k-induction[6]

3-Researcher enhancing the software development cycle to reduce buffer by using maturity model (SD2-C2M2)[8].

4-Researcher proposed static analysis Firm Scanner to study the implement level security issues in software components of IoT [19]

5- Researcher fixed problem in Amazon Web Services (AWS) like overflow and more by the result of the model which improve the software developer practice[1].

6-Researcher proposed BOVP model to predict vulnerability in buffer over flow[18].

7-In the aim of discovering security vulnerabilities such as buffer overflows or crashes, Fuzzing continuously implement with all kind of input varieties[7].

Although numerous methods of buffer overflow detection be present, many of these solutions hurt from great difficulty and/or poor efficiency[6].so we can't pass judgment on the effectiveness of any method if there are experimental opportunities available for a reasonable period of time and be supported on the ground.

Research title	Aim	Research approach
Reducing buffer overflow	**Minimize and detect occur of overflow**	The approach takes place in more than one way by running a program that determines the maximum length that causes a buffer overflow and the minimum length that causes the buffer overflow through an input that causes the buffer overflow as such The approach may require the calculation of whether a number of calls surpass the threshold. In reaction to the amount of calls that reach the threshold, the approach may require adding a patch designed to avoid buffer overflow in the request function.
This research is about Buffer overflow disclosure depend of a models and k-extrapolation composition	**Detection of buffer overflow**	The buffer overflow detection technique includes gaining A software script formed by the buffer indicator parameter to obtain storage spots in a cycle, gaining a statement framework planned to produce dependence in the program code between the buffer index parameter and the loop index parameter, Create an statement through the statement framework and test that the claim is maintained by means of a k-inducement and determine if a buffer overflows occur.
This paper talks about an advanced model (SD2-C2M2): in its design flexibility and security .	**enhancing the program development cycle to reduce buffer overflow**	Maturity Model (SD2-C2M2) offers a browser-based method that offers a conceptual case study on the device by enhancing the software development cycle to reduce the potential for program buffer overflow attacks.In this case study rsearchers Prove the effectiveness of a SD2-C2M2 technique as well program execution via reviewing that one to show how best practices can be implemented to protect against buffer overflow attacks
OVER: Overhauling	**A need to study the implement**	Internet of Things (IoT) presents numerous software-level weaknesses, so researcher proposes a static IoT analysis

Vulnerability Detection for IoT through an Adaptable and Automated Static Analysis Framework	**level security issues in software components of IoT through static analysis**	framework called Firm Scanner to identify vulnerabilities like Buffer Overflow, Resource Leaks, Malware Injection, TOCTOU, Forbidden Operations, and other weaknesses linked to code .The study reached many important results, including that They find obsolete copies of BusyBox rife with vulnerabilities to buffer overflow, at least 1800 of the firmware running v0.60.
Code-Level Model Checking in the software Development Workflow	**Fixed problem in Amazon Web Services (AWS) like overflow and more by the result of the model which improve the software developer practice.**	Research explains the model of application of symbolic design testing built across the course of four years in Amazon Web Services (AWS) The knowledge gained are taken from proof the features of various C-, such as custom hypervisors, encryption code, boot loaders and an IoT operating system. Applying them methods to prove the correctness of industrial low-C-based system with equal effort and predictability. Traditional source code testing can have a substantial positive effect on the consistency of the manufacturing code that the cause of a high level of security issues, regardless of the number, was the buffer overflow compare with memory safety, functional and Null-pointer
This topic is about A forecasting method for buffer overflow of program criteria and deep learn	**Predict vulnerability for buffer overflow**	The BOVP model introduced in a study first selects the functional software origin blade afterwards structures dynamic data stream evaluation framework at program operation stage by advance the decision tree algorithm through choice with Gini coefficient. Lastly, a weakness detection approach centered off program measures is introduced. An ability in template BOVP is checked through tests employing system origin code coded in various program idiom, and the foretelling reports are better exact

		than previous techniques, and the duration needed to the collection of data obtained through C++&C were lower and the precision performance existed 82.53 percent.
Fuzzing :Hack‘ Art, and Science	**Important Fuzzing to detect buffer overflow**	FUZZ Checking, is the method of discovering security issues in input parsing code by continuously checking the converter with changed, or fudged, inputs. Thousands of security weakness were discovered when fuzzing programs of all kinds. If the program is coded in low-level language such as C or C++, the risk becomes even greater because security vulnerabilities are generally simpler to exploit then. If the program is written in high-level handled code such as Java or C #, Fuzzing can expose unmanaged exclusions that may or may not be critical to security based on the content.

6. Conclusion & Future Work:

In this paper, we outlined buffer overflow attack concept which means the code exceed the capacity of the buffer. Types are stack ,heap , integer, string-formats and Unicode offer flow .Vulnerable that’s make the program to crash and also compromise an intruder, protection mechanism from exploiting vulnerabilities which are data execution prevention, address space Layout randomization, stack cookies and canaries, read-only the global offset table and place isolated executable . The last view of recent research to detect and solve the buffer overflow in many way. Buffer overflow attack is a common vulnerability of software security, cyber risks, and an inspiring field for many researchers and those interested. There must be support for modern methods, comparison among them, and the conclusion of judgments that are useful in solving a large part of this threat.

7. Reference

[1] N. CHONG, B. COOK, K. KHAZEM, K. KALLAS, F. MONTEIRO, D. SCHWARTZ-NARBONNE, S. TASIRAN, M. TAUTSCHNIG and M. TUTTLE, *Code Level Model-Checking in the Software Development Workflow*, International Conference on Software Engineering, 2020.

[2] CLOUDFLARE, *What is a buffer over flow*, (visit in 2020).

[3] C. COURCOUBETIS and R. WEBER, *Buffer overflow asymptotics for a buffer handling many traffic sources*, Journal of Applied Probability, 33 (1996), pp. 886-903.

[4] C. COWAN, C. PU, D. MAIER, J. WALPOLE, P. BAKKE, S. BEATTIE, A. GRIER, P. WAGLE, Q. ZHANG and H. HINTON, *Stackguard: Automatic adaptive detection and prevention of buffer-overflow attacks*, *USENIX security symposium*, San Antonio, TX, 1998, pp. 63-78.

[5] C. COWAN, F. WAGLE, C. PU, S. BEATTIE and J. WALPOLE, *Buffer overflows: Attacks and defenses for the vulnerability of the decade*, *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*, IEEE, 2000, pp. 119-129.

[6] F. GAUTHIER, N. KEYNES, P. KRISHNAN, C. CIFUENTES and T. Q. TA, *Buffer overflow detection based on a synthesis of assertions from templates and k-induction*, Google Patents, 2020.

[7] P. GODEFROID, *Fuzzing: hack, art, and science*, Communications of the ACM, 63 (2020), pp. 70-76.

[8] S. N. G. GOURISETTI, S. MIX, M. MYLREA, C. BONEBRAKE and M. TOUHIDUZZAMAN, *Secure Design and Development Cybersecurity Capability Maturity Model (SD2-C2M2) Next-Generation Cyber Resilience by Design*, *Proceedings of the Northwest Cybersecurity Symposium*, 2019, pp. 1-9.

[9] S. GUPTA, *Buffer overflow attack*, IOSR Journal of Computer Engineering, 1 (2012), pp. 10-23.

[10] W. HAN, M. REN, S. TIAN, L. DING and Y. HE, *Static analysis of format string vulnerabilities*, *2011 First ACIS International Symposium on Software and Network Engineering*, IEEE, 2011, pp. 122-127.

[11] L. JIE, H. DA and R. ZHIHONG, *An Analysis Model of Buffer Overflow Vulnerability Based on FSM*, *Proceedings of the 2019 2nd International Conference on Geoinformatics and Data Analysis*, 2019, pp. 47-51.

[12] M. J. KHALSAN and M. O. AGYEMAN, *An overview of prevention/mitigation against memory corruption attack*, *Proceedings of the 2nd International Symposium on Computer Science and Intelligent Control*, 2018, pp. 1-6.

[13] E. LEON and S. D. BRUDA, *Counter-measures against stack buffer overflows in GNU/Linux operating systems*, Procedia Computer Science, 83 (2016), pp. 1301-1306.

[14] Y. LI, S. JI, C. LV, Y. CHEN, J. CHEN, Q. GU and C. WU, *V-fuzz: Vulnerability-oriented evolutionary fuzzing*, arXiv preprint arXiv:1901.01142 (2019).

[15] B. LIU, L. SHI, Z. CAI and M. LI, *Software vulnerability discovery techniques: A survey*, *2012 fourth international conference on multimedia information networking and security*, IEEE, 2012, pp. 152-156.

[16] P. MURTHY, *Reducing buffer overflow*, Google Patents, 2020.

[17] Ș. NICULA and R. D. ZOTA, *Exploiting stack-based buffer overflow using modern day techniques*, Procedia Computer Science, 160 (2019), pp. 9-14.

[18] J. REN, Z. ZHENG, Q. LIU, Z. WEI and H. YAN, *A Buffer Overflow Prediction Approach Based on Software Metrics and Machine Learning*, Security and Communication Networks, 2019 (2019).

[19] V. SACHIDANANDA, S. BHAIRAV and Y. ELOVICI, *OVER: overhauling vulnerability detection for iot through an adaptable and automated static analysis framework*, *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, 2020, pp. 729-738.

[20] A. SHAHAB, M. NADEEM, M. ALENEZI and R. ASIF, *An automated approach to fix buffer overflows*, International Journal of Electrical and Computer Engineering, 10 (2020), pp. 3777.

[21] L. K. SHAR, L. C. BRIAND and H. B. K. TAN, *Web application vulnerability prediction using hybrid program analysis and machine learning*, IEEE Transactions on Dependable and Secure Computing, 12 (2014), pp. 688-707.