The Source and Exploitation of the Program Vulnerability

Li jian[1] , Lou jing[1] , Gao Yang[2]
1.Hebei Electric Power CO., LTD. National Power Grid, 075000, China
2.ZhangJiaKou Electric Power CO.,LTD. 075000, China
electricpo@sohu.com

***Abstract*-Vulnerabilities with the C and C ++ language programming are the source of information security incidents. The language is an insecure programming language, which allows an attacker to change the behavior of the running program or completely control it through the vulnerabilities in the program. This issue has been known for more than 30 years. During this period, many vulnerability mitigation technologies have been proposed to prevent the exploitation of the vulnerabilities. However, many facts have proven that these mitigation technologies can not completely solve the problem. This paper clearly illustrates that the flaws and errors of C and C ++ language are the fundamental cause of the vulnerability generation. Then, the paper further analyzes the vulnerabilities generated by the flaws and errors as well as the implementation attacks by exploiting the vulnerabilities. Lastly, this paper also points out the key to the successful implementation attacks by exploiting these vulnerabilities.**

***Keywords*- vulnerabilities, exploit, attack**

I. INTRODUCTION

Modern society is an information society, and our daily life cannot be separated from a great variety of powerful software, from the super computer system to be used to predict the weather to the social application software such as WeChat, Twitter and Facebook. We can say that humans have been inseparable from the intelligent production and the daily life helper. However, everything has its two sides: you enjoy the convenience brought by it, at the same time you will also suffer from its troubles, sometimes even a nightmare. At present, with the rapid development of computer and Internet, endless attacks for a variety of software emerge and increase with each passing day, which cause serious damages to the society and individuals: information leak, property damage, threat to safety and so on. Among these serious losses from the attack, the clear majority are caused by exploiting software defect in the software system. Software defect is that in the process of developing software, human being integrates some errors into the software. These errors include the imperfect design and the coding. If these defects make the software at risk, the software defects are called security flaw. If these security flaws are exploited by dishonest persons to get some benefits, the security flaws become software vulnerability. Software vulnerabilities exist in three main phases of software development process-framework design, coding implementation and actual operation.

Framework design phase is the core phase of every software development phase. At this stage, it is necessary to make a series of key decisions about how to develop software, including the high level software components, functional components, installation and deployment, performance optimization, etc. And some security flaws exist at this phase, such as unreasonable framework design and imperfect demand. These security flaws cause software vulnerabilities. Normally it's difficult to find the vulnerabilities generated at framework design phase during the software running process, but the damage is great. So the reasonable and secure design of the software at the framework design phase can greatly reduce the generation of software vulnerabilities. And, the sooner the software vulnerabilities are found, the lower the vulnerability remediation is. The software running phase is the last part of the three phases. At this phase the software deployment running and software running environment configuration are main tasks, and the main insecurity flaws mainly include software configuration errors, authentication security errors and system data security errors, etc. These vulnerabilities caused by security flaws are access authentication vulnerabilities and data confidentiality vulnerabilities, etc. These vulnerabilities and the vulnerabilities generated at framework design phase are usually considered as software design vulnerabilities. Therefore, the vulnerabilities generated at two phases have little to do with the middle phase of the

software development process-the software coding implementation phase, which are mainly caused by the software development subject itself. Therefore, the vulnerabilities generated during the framework design phase and software running phase should be prevented from software design by taking measures of security design principles and security design strategies. The main design principles include the minimum permission principle, the deep defense principle, the permission separation principle and so on. The main security design strategies include integrity strategy, authentication mechanism and confidentiality strategy [1].

The academic and industrial circles focus on the vulnerabilities generated during the software coding implementation phase which equals the vulnerabilities in the traditional sense, and it's also the focus of this paper. These vulnerabilities generated at this phase have something to do with the software development subject; however the main reason is the language that has been used to develop software. At current time, the majority of software systems are written in C and C ++ language. And there are a lot of legacy code base of C language development. However, the well-known C language is not a type-safe language. Type-safe language requires a specific operation type and the result is still the same type[错误!未找到引用源。]. C language derives from two untyped languages: B language [3] and Basic Combined Programming Language(BCPL) [4]. It remains many attributes of untyped language, if there is explicit or implicit conversion, it may cause the security flaws. For example, when the pointer to one type converts to the pointer to another type, an undefined behavior occurs when the converted pointer is dereferenced [5], and then generate a vulnerability that can be exploited. Lack of type-safety has caused a lot of security flaws and vulnerabilities. In addition, C language is a flexible and easy-to-transplant high-level programming language, as well as a lightweight language with less memory consumption, and these attributes make it possible to contact tightly the underlying framework of the system and execute more efficiently, so it's especially suitable for the system software development such as operating system. However, the attributes of C language also confuse the programmers, they mistake that something like array bounds check, the pointer initialization, memory release and collection is manipulated automatically by C, but in fact C doesn't manipulate it automatically, which also leads to the generation of software vulnerabilities. In summary, the extensive use of C language in the software development process and the inherent attributes of C language make it become the hardest hit by frequent vulnerabilities. This paper gives a systematic and deeper analysis and illustration of vulnerabilities generation during software coding implementation phase by using C language, and analyzes the paper as follows: In the Second section we systematically analyze the flaws and errors of C language, which are the primary cause of the vulnerability generation. In the Third section, the paper studies the classification of the attack implementation by exploiting the vulnerabilities generated by flaws and errors, and points out the key to the successful attack implementation by exploiting these vulnerabilities. Finally, Section Four summarizes the main contents of the paper.

II. THE CAUSE OF VULNERABILITY

As described above, C language is a type-unsafe language. Due to the programmer's intentional or unintentional behaviors and some inherent attributes in C language, there are some errors in the developed code which may generate software vulnerabilities that can be exploited by attackers. Currently, the errors that exist in C language are the following: Buffer overflow, pointer subterfuge, memory management error, integer overflow and formatted output error.

Buffer overflow occurs when data is written into the memory space to certain data structure; the written data exceeds the memory space boundary assigned to the data structure and overlays the contents of the adjacent memory space to the data structure. The main cause of this error is: (1) C language defines string as a null-terminated character array. During the program execution process, the judging condition of the end of the string is whether the source string contains a null character identifier. Therefore, during a replication operation from the source string to the target string, the condition of replication end is whether the source string contains null characters, which is likely to make the number of the replication characters exceed the target string capacity and cause overflow error; (2) C language provides programmers a great number of function calls. These functions greatly

facilitate the programmers' development and greatly improve their development efficiency. But one common flaw in these functions is not to carry out mandatory bounds check for C language data type, which will cause an error of cross-border operation of data elements in the program running process[6~7] .

Pointer subterfuge means that because the pointer is not well operated and protected, resulting in the error that the pointer value is modified and used[8~9]. Normally, we can overlay and modify some pointers in program through buffer overflow, which transfer the program control to the malicious code location supplied by the attacker and complete the attack intention. To implement the attack through pointer subterfuge error, it requires to meet the following conditions: (1) There is a buffer overflow error in the program, and we can overwrite the pointer value through buffer overflow; (2) The buffer with buffer overflow error and the overwritten pointer are in the same memory segment, and the adjacent location is better. Then an attacker can implement the attack directly by overwriting the pointer adjacent to buffer overflow.

C programs often need operate the data objects with variable elements number, and the data objects should be allocated dynamically by using the dynamic memory management technology. However, incorrect usage of dynamic memory management technology will cause a lot of errors which lead to many memory-related vulnerabilities, and memory management has become the major cause of C language program vulnerabilities[10]. Common errors of memory management are: (1) Memory initialization error. When the memory allocation function is used, for example, malloc function allocates memory, allocated memory has not been initialized and the content is indefinite which is prone to cause information leak. (2) A null pointer or an invalid pointer dereference error. When the memory allocation function is used to allocate memory, if the allocation is not successful, the function returns to a null pointer. When the pointer exceeds the referred object boundary or the referred object is freed, it will become an invalid pointer. Dereference of these pointer leads to so-called temporal error or spatial error, which cause the program crash[11]. (3) Reference of freed memory. When the memory allocated to an object is freed, the pointer to the memory space should be assigned null or redirected to other memory space. Otherwise, the freed memory space still can be accessed via the pointer, resulting in information leak problem. (4) Memory leak[12]. When the memory space allocated to a object is not used without freeing, it will cause a memory leak error. Memory leak extreme case is that there is no memory space that can be allocated in the system and cause denial of service attacks.

When a signed integer operation result exceeds the maximum value of the integer type presentation, integer overflow error occurs. According to [13] the classification, integer overflow can be divided into integer overflow, integer underflow, unsigned error and truncation error. Integer overflow resulting from integers addition, subtraction, multiplication and shift operation, as well as truncation error from integers type conversation operation, which exists great difference between the integer operation result and the expected value, at the same time unsigned error from type conversion operation will confuse the correct understanding of numerical value between the negative and the maximum positive. In short, integer overflow error causes the program calculation result error, data loss and misinterpreted values. But integer overflow does not directly modify the memory content, the attackers often implement attacks by indirectly using integer overflow error in program instead of directly using integer overflow[14].

Formatted output function is defined in C language as output function of acceptance variable number of arguments, and one of the arguments is called the format string. A user can control the execution of the formatted output function by controlling the content of the format string to control the execution of the formatted output functions. Incorrect use of formatted output function causes the following errors: (1) Buffer overflow. Under normal circumstances, the formatted output function assumes that buffers of arbitrary length exist. Under this assumption, overflow error occurs without the bounds check of destination array. (2) Print the program stack and the contents of memory. When formatted output function is used, an attacker can check stack and memory of arbitrary address by using the format string which displays memory of the specified address. In this way, an attacker can obtain the memory contents of specified address in the program, which cause information leak of the program [15].

III. THE ATTACK BY EXPLOITING THE VULNERABILITIES

As we described in the first section, these errors become vulnerabilities when the errors from the coding implementation phase are used by an attacker. Therefore, vulnerability types in C language can be divided into buffer overflow vulnerability, pointer subterfuge vulnerability, memory management errors vulnerability, integer overflow vulnerability and format string vulnerability. Normally, an attacker will implement attack by exploiting comprehensive vulnerabilities rather than a specific vulnerability. Currently, according to the attack motivations and types, implementation attacks by exploiting program vulnerabilities can be divided into: information leak attacks, memory corruption attacks, control-flow hijack attacks and non-control-data attacks[16~17] [12].

Information leak attacks mean that by exploiting the vulnerabilities in program in a non-authorized way an attacker can obtain program-related information such as the program address spatial layout in memory, the entry address of the function and the register contents used in program. Even an attacker can also obtain user-related information such as the system login password and credit card account. Furthermore, information leak attacks not only directly obtain the program information, but also become the premise condition of other attacks such as control-flow hijack attacks. For control-flow hijacking attacks, firstly the attacker must obtain memory address information by exploiting the program vulnerabilities (it's information leak attacks). Then, the attacker transfers the attack payload designed ingeniously as the input argument of the function (the function is a function with security flaws, for example strcpy ()) to the program. The execution flow is hijacked during the program with vulnerabilities running and is transferred to the code location that has been set by the attacker, to complete control-flow hijack attacks. Information leak attacks are divided into two types: one is intrusive attack, which an attack obtains the program-related information and take the initiative to modify the contents of the program with vulnerabilities (for example modification of the program stack) to complete the attack. Another type is called side channel attacks (SCA) , and it is non-invasive information leak attacks. For this type an attacker does not take the initiative to modify the program, but inputs some data to the running program by exploiting the vulnerabilities in program, observes the program response to the input data and infers some "information" in program. Side channel attacks initially implement attacks against information leak of time, the power consumption or electromagnetic radiation during electronic equipment operation process. But there are signs that this type of attacks gradually infiltrates into the implementation attack of the software. For information leak attacks, vulnerability that can be exploited includes memory corruption vulnerability and format string vulnerability, and they both can be used to leak program information.

Memory corruption attacks occur after the memory in the running program is intentionally or unintentionally modified. Memory corruption violates memory safety strategy, which will cause a program crash or abnormal behavior. However, an attacker will use this corruption to implement memory corruption attacks [18-19]. Memory corruption attacks may be the goal of the attackers, for example, in order to obtain certain information of the program or a user; It may be an intermediate step of an attack implemented by an attacker, for example, the attacker should know the address spatial layout in program with vulnerabilities beforehand, in order to implement control-flow hijack attacks. When memory corruption attacks are implemented, the attacker can use the memory initialization error or release re-reference error to obtain the program information with vulnerabilities, use a null pointer or an invalid pointer dereference error to perform arbitrary written operation of memory in program with vulnerabilities, and use memory leak to deny memory requests from valid users. For memory corruption attacks, the vulnerability that can be used includes: buffer overflow vulnerability, pointer subterfuge vulnerability, memory corruption vulnerability.

Control-flow hijack attacks are a extremely harmful type of attacks at current time, which allow the attacker to get the control over the target machine and then get full control over the target machine. Under normal circumstances, program in running process should be executed according to pre-defined control flow diagram graph (CFG) to ensure that the program control flow will not be hijacked or tampered with, or deviate from control flow transfer relationship designed by program.

However, if an attacker knows the vulnerability in program that can be exploited, for example, buffer overflow vulnerability, he can exploit this vulnerability to hijack or tamper with the execution flow of program and make the program execute towards the direction set by the attacker to achieve the purpose of the attack. Early control flow hijack attacks by using the code injection are called code injection attacks (CIA). For this attack, the attacker exploits the program vulnerabilities to inject malicious code, then tamper with the program control flow to the injected code which is executed to implement attacks (system permission elevation, privacy theft etc.). In recent years, control flow hijack attacks gradually use the functions and code in the program with vulnerabilities to implement attacks. Using the function in program to attack is called return-into-libc(RILC)[20]; Using the code in the program to attack is called return-oriented programming (ROP). For RILC attack, an attacker tampers with the program control flow and makes it point to some standard library functions. An attacker can call system () function to spawn new process by attacking, or call mprotect() function to create a memory area that can be read, written and executed to bypass the traditional defense method of the system. After this attack method is improved, it can continuously call code segment in program and a series of functions in standard library, to complete more complex functions. However, RILC application range is quite limited because the function number for attacker's option is small, and the success depends on several key functions in standard library. In 2007, the RILC idea was extended and ROP technology was proposed. For ROP attack, an attacker can use return instruction ret in program instructions to organize some short attacking code snippets to implement attack, and these short snippets called gadget. For implementing ROP attacks, an attacker obtains the program information by exploiting the program vulnerabilities, and then organizes a certain code snippets into gadget chain which will be put into the program stack, and implements attacks. Compared with RILC attacks, ROP attacks does not need the standard library functions in program, because there are a lot of ret instruction-end code snippets in program, an attacker can easily construct code snippets for attacks and implement the attacks. And in 2012, Roemer also proved that these short code snippets had a turing-complete characteristic, which made it more convenient to construct the code snippets for attacks. For control-flow hijack attacks, the vulnerabilities can be exploited including buffer overflow vulnerability, pointer subterfuge vulnerability, memory corruption vulnerability, integer overflow vulnerability and format string vulnerability.

There is an attack similar to control-flow hijack attacks, for this attack, an attacker exploits buffer overflow vulnerability to modify non-control data in program such as user identity information, configuration information and password with a certain semantic variable rather than return address or function pointer (control data in program) in program. We call it non-control-data attacks (NDA). As a premise to implementing attack, an attacker should know well about the program data area and construct an ingenious attack data payload. Code 1 shows an example of non-control data attacks.

Code 1 non-control data attacks

```
void login_client(int file){
    bool admin=false;
    char client_name=[128];
/*a buffer overflow error*/
    read(file,client_name,256);
    if (admin)
```

Fig. 1. Code 1.

For above example, the attacker can design an ingenious attack payload to overlay the value of variable admin through buffer overflow errors caused by function read (), and log in the system as a non-administrator identity. This example proves that for non-control data attacks, the attacker modifies the non-control data rather than the control data in program. Furthermore, the non-control data is a semantic variable, and it's difficult to prevent the modification from binary code. Moreover, in 2016, Hu et al proposed a new attack method based on the non-control data, data-oriented programming (DOP) attacks. DOP attacks are similar with ROP attacks, by constructing many "data-oriented gadget" in the program to implement attack, and the attack method proves to be turing-complete. However, the difficulty of implementing non-control data attacks is larger than flow-control hijack

attacks, because it requires a further understanding of program with vulnerabilities, and it is better to get the source code of program with vulnerabilities and analyze it, to know the semantic information for each variable and the relative locations (convenient to exploit buffer overflow vulnerabilities to overlay information). Furthermore, the information obtained by disassembling binary program doesn't help much for attack implementation. Because many commercial binary programs add code confusion technology, normally the information obtained by disassembly has little value. However, this attack does not modify any control data in program, and it's difficult for the existing defensive measures (such as DEP and ASLR) to detect and prevent this attack, so the attack is a big threat to system safety. Similarly, for non-control-data attacks, vulnerabilities that can be exploited include buffer overflow vulnerability, pointer subterfuge vulnerability, memory errors vulnerability, integer overflow vulnerability and format string vulnerability.

IV. CONCLUSIONS

This paper analyzes the cause of the vulnerability and the attack of vulnerability - based. From the paper, we all know that the C language is a type-unsafe language. Due to the inherent nature of some C languages and the programmer's negligence, there is much vulnerability in the program. These vulnerabilities are exploited by hackers and attack the system. This is a very serious situation. We must take measures to prevent these attacks. Through this paper, we hope to draw attention to the vulnerabilities and solve the attacks caused by the vulnerabilities.

REFERENCES

[1] S. Wu, Software vulnerability analysis technology. Science Press, 2014.

[2] F. Pfenning, "Lectures notes on type safety:foundations of pro-gramming languages," Carnegig Mellon University, Pittsburgh, Lecture 6, 2004. [Online]. Available: www.cs.cmu.edu/¬~fp/¬courses/¬15312-f04/-handouts/¬06-safety.pdf

[3] S. C. Johnson and B. W. Kernighan, "The programming language b," NJ:Bell Labs, Murray Hill, Computing Science Technical 8, 1973.

[4] M. Richards and C. Whitby-Strevens, BCPL: the language and its compiler. Cambridge University Press, 1981.

[5] ISO/IEC, Programming Languages-C,Third Edition(ISO/IEC 9899:2011). Geneva,Switzerland:International Organization for Stand-ardizarion, 2011.

[6] A. One, "Smashing the stack for fun and profit," Phrack magazine, vol. 7, no. 49, pp. 14–16, 1996.

[7] J. Viega and G. McGraw, Building Secure Software: How to Avoid Security Problems the Right Way, Portable Documents. Pearson Educa-tion, 2001.

[8] J. Pincus and B. Baker, "Beyond stack smashing: Recent advances in exploiting buffer overruns," IEEE Security & Privacy, vol. 2, no. 4, pp. 20–27, 2004.

[9] T. Committee et al., "Tool interface standard (tis) executable and linking format (elf) specification version 1.2," TIS Committee, 1995.

[10] M. Graff and K. R. Van Wyk, Secure coding: principles and prac-tices. " O'Reilly Media, Inc.", 2003.

[11] B. Jack, "Vector rewrite attack: Exploitable null pointer vulnerabil-ities on arm and xscale architectures," White paper, Juniper Networks, 2007.

[12] S. Chen, J. Xu, E. C. Sezer, P. Gauriar, and R. K. Iyer, "Non-control-data attacks are realistic threats." in Usenix Security, vol. 5, 2005.

[13] D. Brumley, T.-c. Chiueh, R. Johnson, H. Lin, and D. Song, "Rich: Automatically protecting against integer-based vulnerabilities," Department of Electrical and Computing Engineering, p. 28, 2007.

[14] S. Designer, "Jpeg com marker processing vulnerability in netscape browsers," 2000.

[15] T. T. Scut, "Exploiting format string vulnerabilities," 2001.

[16] L. Szekeres, M. Payer, T. Wei, and D. Song, "Sok: Eternal war in memory," in Security and Privacy (SP), 2013 IEEE Symposium on. IEEE, 2013, pp. 48–62.

[17] P. Larsen, A. Homescu, S. Brunthaler, and M. Franz, "Sok: Automated software diversity," in 2014 IEEE Symposium on Security and Privacy. IEEE, 2014, pp. 276–291.

[18] Ú. Erlingsson, Y. Younan, and F. Piessens, "Low-level software security by example," in Handbook of Information and Communication Security. Springer, 2010, pp. 633–658.

[19] V. Van der Veen, L. Cavallaro, H. Bos et al., "Memory errors: the past, the present, and the future," in International Workshop on Recent Advances in Intrusion Detection. Springer, 2012, pp. 86–106.

[20] R. Wojtczuk, "The advanced return-into-lib (c) exploits: Pax case study," Phrack Magazine, Volume 0x0b, Issue 0x3a, Phile# 0x04 of 0x0e, 2001.