Research on the Protection Mechanism of Cisco IOS Exploit

Jin Zhu Wang, Ruijie Cai and Shengli Liu*
State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450001, China
*Corresponding author: wangjinzhugood@126.com

Abstract. Cisco IOS is an operating system that runs on Cisco routing and switching equipment. Its security is important to the entire network. Since vulnerability attacks have become one of the main sources of threat to Cisco IOS security, it is necessary to analyze the defense mechanism of IOS against vulnerability attacks. The purpose of this paper is to find out the deficiencies of IOS vulnerability defense mechanism in time and put forward improvement methods and preventive measures. Firstly, this paper introduces the protection mechanism of Cisco IOS vulnerability, including CheckHeaps, DEP, ASLR. The principle of these mechanisms and the idea of vulnerability protection are analyzed. And then, after analyzing the shortcomings of these protection mechanisms, the basic idea of bypassing these protection mechanisms is proposed. Finally, in view of the shortcomings of Cisco IOS vulnerability defense mechanism and the proposed bypass methods, some advices about how to better protect Cisco IOS from vulnerability exploitation are given.

1. Introduction
With the continuous development of science and technology, the scale of the network has expanded rapidly. As the core switching equipment of the network, routers play an increasingly important role. Currently, the most commonly used routers on the market are Cisco routers. The operating system of Cisco routers is Cisco IOS (Internetwork Operating System). The vulnerabilities of Cisco routers which can be exploited by hackers and criminals threaten the whole Internet.

The exploitation of IOS vulnerabilities can be traced back to around 2002. Since then, researchers have continuously proposed various methods to try to bypass the vulnerability protection mechanism of IOS. Cisco IOS, which is generally based on PowerPC or MIPS architecture, does not support third-party software operation. Very little research of IOS vulnerabilities is published because of the IOS doesn’t open source. Among them, Lynn introduced a method of bypassing the protection mechanism of IOS CheckHeaps at the BlackHat conference [1] and the exploit method introduced by Felix in the article Cisco IOS Router Exploitation [2]. These researches are attacks against the defects of the IOS vulnerability protection mechanism. Therefore, in order to defend against Cisco IOS vulnerability attacks, we must first understand the deficiencies of IOS vulnerability utilization protection mechanism, and then put forward relevant defense strategies in time.

This paper analyzes the methods of bypassing the protection mechanisms of Cisco IOS for vulnerability exploits and proposes the corresponding protection and improvement measures for these vulnerability exploit methods. The reminder of this paper is organized as follow: Part 2 analyzes the protection mechanism for Cisco IOS exploits; Part 3 briefly introduces the ideas of bypassing Cisco
IOS exploit protection techniques, and Part 4 proposes improvement plans and Defensive measures; the last part is a summary and outlook of future work.

2. Cisco IOS protection mechanisms

Please Facing the threat of vulnerability attack, some characteristics of IOS make it more difficult to exploit the vulnerability of IOS. IOS is a non-open source operating system, and its vulnerability exploitation relies on reverse engineering analysis technology. Because IOS image file is a huge static link binary file, the reverse analysis of IOS is very difficult. IOS does not provide the API used by the third-party developers, so developers must search the address of the required function by themselves to increase the vulnerability exploitation Difficulty. In addition, IOS does not have an exception handling mechanism. Once an error occurs, IOS will restart immediately.

IOS also has some exploit protection technologies to combat vulnerability attacks, such as Data Execution Prevention (DEP), Address Space Layout Randomization (ASLR), CheckHeaps, etc. we will briefly introduce the main vulnerability protection technologies of Cisco IOS.

2.1. CheckHeaps

The CheckHeaps process is unique to Cisco IOS. The process checks the heap block regularly to protect the IOS memory blocks and ensure the integrity of the heap blocks. The block header structure of the IOS memory block is as follows [3]:

```
struct HeapBlock {                     // 0xAB1234CD
    DWORD Magic;                     // 0xAB1234CD
    DWORD PID;
    DWORD AllocCheck;
    DWORD AllocName;
    DWORD AllocPC;
    void *Nextptr;                           // Pointer to the following block
    void *Prevptr;                           // Pointer to the previous block’s NextBlock
    DWORD BlockSize;
    DWORD RefCnt;
    DWORD LastDealloc;
};
```

The memory block exists in the form of a doubly linked list. The CheckHeaps process adds a REDZONE value of 0xFD0110DF after the data area of the memory block to verify whether an overflow occurs.

The memory block structure is shown in Figure 1.

![Figure 1. Structure of a block header.](image)

Because multiple linked list structure is easy to break, in order to prevent Cisco router from crashing due to heap memory block damage, CheckHeaps process will check the integrity of memory block on a regular basis, and the contents of CheckHeaps process check are as follows [4]:

1. Verify that the block header Magic value is 0xAB1234CD;
2. If the memory block is a non-free block, verify that REDZONE contains 0xFD0110DF;
3. Verify that the Prevptr pointer is not empty;
4. Verify that the Nextptr pointer of the Prevptr pointer points to this block;
5. If the Nextptr pointer is not empty, verify that it is just behind the REDZONE field of this block;
6. If the Nextptr pointer of the Prevptr pointer points to the block;
6. If the Nextptr pointer is empty (the last block in the chain), verify that it does end on a memory boundary.

In addition to regular heap block integrity checks, these checks are performed when a process requests or releases a heap block. If an error occurs after integrity verification, the CheckHeaps process will perform corresponding logging and recovery operations, and then force Cisco IOS to restart.

2.2. Data Execution Prevention (DEP)

IOS divides the virtual address space into different areas according to the characteristics of the corresponding physical memory. The memory area can be divided into eight categories, as shown in Figure 2[5].

| Memory Region Class | Characteristics |
|---------------------|-----------------|
| Local               | Normal run-time data structures and local heaps; often DRAM. |
| Ionmem              | Shared memory that is visible to both the CPU and the network media controllers over a data bus; often is SRAM. |
| Fast                | Fast memory, such as SRAM, used for special-purpose and speed-critical tasks. |
| iData               | Executable IOS code. |
| iBus                | Uninitialized variables. |
| PO                  | PO bus memory, visible to all devices on the PCI buses. |
| Flash               | Flash memory. This region class can be used to store run-from-Flash or run-from-RAM IOS images. It often also can be used to store backups of the router configuration and other data, such as crash data. Typically, a file system is built in the Flash memory region. |

**Figure 2.** Memory Region Classes.

The characteristics of each memory area of IOS are shown in Figure 2, and their distribution in memory is shown in Figure 3:

![Figure 3. Memory Regions.](image)

In the process of vulnerability exploitation, the vulnerability attack code makes the program execution flow jump to the shellcode written by the attacker to implement specific functions through careful construction. The shellcode is mostly stored as data on the stack or in the IO cache. It is not difficult to see that the root of the overflow vulnerability attack is that data and code are not clearly distinguished. If the shellcode as data is not run as code, the exploit attack cannot be performed. The introduction of Data Execution Prevention (DEP) is to make up for the shortcoming of data and code confusion in IOS.

According to the memory distribution of IOS, when the system is loaded, IOS places data and code in different memory areas according to the memory block category. DEP sets access permissions for each memory block based on the memory block category and attribute. The basic principle of DEP is to identify the memory area where the data is located as unexecutable. When the program overflows and is successfully transferred to shellcode, the program will try to execute instructions in the memory area where the data is located. At this time, an IOS exception will be triggered and the router will be restarted.

The role of DEP is to prevent the IOS data memory block from executing code. The IOS stack and IO-memory are used as memory for storing data, so the IOS stack and IO-memory are not executable.
2.3. Address Space Layout Randomization (ASLR)
There is a common feature throughout all vulnerability exploitation methods: a clear jump address is needed to enable the attacker to enter the expected attack process. ASLR technology is a protection mechanism to interfere with the code location of vulnerability attack by loading IOS image without fixed base address. After the address space is randomized, it is difficult for the attacker to determine the relative offset of the attack code first. Therefore, it is not possible to implement a jump by overwriting the memory address. As shown in Figure 4, there are two types of ASLR on the IOS (and they can coexist):
- RTO: Move the IText section by a random offset when loading IOS (usually the last 4 digits of the IText starting address)
- RDO: move the IData and IBSS areas where data is stored to a random offset when IOS is loaded. This means that the pointing addresses are changed within the IText section, so the content of the IText section changes between two system or on the same system when reloaded.

![Figure 4. ASLR Implementation Types.](image)

The principle of ASLR is that after the router restarts, when the router loads the IOS image, the data and code loading address in IOS will randomly change, which makes the jump instruction used in the previous exploit is not stable. The ASLR mechanism does not prevent the exploitation, but makes it impossible for the exploitation to be performed stably.

3. How to bypass the Cisco IOS protection mechanisms
IOS uses a series of exploit protection mechanisms such as CheckHeaps, DEP, and ASLR, which alleviate the harm of exploit attacks. In order to find the flaws of these exploit protection mechanisms in time and propose corresponding improvements, we will analyze the bypass methods of these exploit protection mechanisms.

3.1. CheckHeaps
The memory protection mechanism of IOS is mainly completed by the CheckHeaps process, which not only checks the integrity of the entire IOS heap block regularly, but also checks the integrity of the free heap memory block before the heap block is applied or released. Through the analysis, it can be found that the integrity check does not check the value of Nextptr pointer although it checks the correctness of the bidirectional linked list of heap memory block and the related fields of its header structure [6]. Next, we take free block merging as an example to introduce CheckHeaps bypass method.

Before the heap memory block is released, the system will check whether the adjacent memory block is free block. If the adjacent memory block is free block, it will perform the free block merging operation, and then add the combined free block to the free block list. Assuming that the free block block1 is adjacent to the memory block waiting to be released, the list of free blocks before and after merging is shown in Figure 5.
**Figure 5.** Free block merge.

In the process of free block merging, the system just takes the adjacent free block from the free block list and merges it with the memory block to be released. It does not check the heap memory block and the data above the Nextptr pointer of the memory block to be released, but does the following to the relevant pointer on the heap memory block:

\[
\text{(block -&gt; Prevptr -&gt; Nextptr} = \text{block -&gt; Nextptr} \quad (1)
\]

\[
\text{(block -&gt; Nextptr) -&gt; Prevptr} = \text{block -&gt; Prevptr} \quad (2)
\]

\[
*\text{(block -&gt; Prevptr)} = \text{block -&gt; Nextptr} \quad (3)
\]

\[
\text{delete(block)} \quad (4)
\]

The memory block to be released and the free block block1 are merged into a new block. Since the memory is not deleted or modified when IOS merges the free block, the data in block1 has not changed.

According to (3), the attacker can forge a free block and fill in the field values of the block header reasonably to prevent the CheckHeaps process from checking. At the same time, the pointer of the free block Prevptr points to the return address of the system process, and Nextptr points to the address where the shellcode is located. When the exploit code triggers the merging of free blocks, the return address pointer of the system process can point to the shellcode address to run the shellcode, thereby achieving the purpose of bypassing CheckHeaps.

### 3.2. Data Execution Prevention (DEP)

The root cause of the exploit failure under DEP protection is that DEP detects that the program goes to the non-executable memory area to execute instructions, which causes the restart of Cisco IOS. In order to bypass DEP, we can use code reuse attacks, especially ROP attacks[7].

ROP is a code reuse attack method that uses small sections of code instructions in the program. These instructions are called gadgets, and the choice of gadgets must meet two conditions: the instruction segment can execute the payload (that is, write or read Some memory) and contains instructions (blr, blrl, bctr or bctrl in PowerPC) that can transfer execution flow to the next gadget. The ROP attack can theoretically implement arbitrary function code execution by linking multiple gadgets through instructions such as jmp / vall / ret. The process is shown in Figure 6.

**Figure 6.** Return oriented programming.
If the conditions of the vulnerability are appropriate, the attacker can use gadget to construct a ROP chain to directly implement the shellcode function, thereby bypassing DEP protection. For example, use ROP technology to write shellcode to a memory area with RW (writeable executable) permissions and then jump to shellcode execution.

There are many code areas that have RW permissions, but in most cases, it is not possible to write shellcode to the IText area because this is prohibited. At this time, if you want to bypass DEP, you can only use gadget to construct a ROP chain to turn off the DEP mechanism of IOS. The following uses the PowerPC405 family as an example to briefly explain how to bypass DEP.

TLB (Translation Lookaside Buffers) stores the recently used page translation address mapping table, and is generally implemented by hardware. TLB entries are used to describe memory area information including access rights. Management of TLB entries is achieved with the help of special instructions. In the PowerPC405 family, the register ZPR can rewrite flags EX (execute enable, 1 bit) and WR (write-enable, 1 bit) of code execution and code write control in the TLB structure. As long as we find a gadget that can modify the register ZPR and use ROP technology to set EX and WR to 0 through register ZPR, then we can write and execute operations in the corresponding memory to achieve DEP bypass.

Although there are many types of Cisco IOS, and the instruction sets used are also different, each instruction set has a corresponding area to manage access rights. An attacker can use ROP technology to modify the value of the corresponding area, change its access rights, and finally realize DEP bypass.

3.3. Address Space Layout Randomization (ASLR)

ASLR has been widely used in windows, Linux and other systems before it was introduced into Cisco IOS system. Attackers analyzed the vulnerability of ASLR according to different situations and summarized the corresponding bypass methods. For example, for the situation where the position of some component changes is small and within a certain range, the attacker may find ways to guess these addresses and then use them. Heap spraying is based on this idea. If there is some code or data in a fixed position in memory, then the attack can use these fixed addresses to bypass ASLR.

Because the stack of any IOS process is an arbitrarily allocated memory block on the heap, its position is random enough to be unpredictable, so it is impossible to use a technique similar to heap spray to implement a vulnerability attack on IOS. In order to overcome the problem of uncertain IOS memory distribution caused by the ASLR mechanism, attackers often implement vulnerability attacks by attacking code segments that do not have ASLR enabled. That is, a piece of executable code with a fixed location is loaded into the memory after the IOS image is loaded, and then this piece of code is used to locate the attacker's attack code and perform subsequent related operations.

ROMMON is a small piece of IOS boot code. It is just like EFI BIOS to modern desktops for Cisco routers. It mainly completes Cisco IOS decompression and loading before startup. In order to match the virtual addressing and address translation of the CPU with the memory map of the IOS image after initialization, ROMMON is placed in the highest area of memory. Therefore, the position of ROMMON in IOS memory is fixed.

Because ROMMON's location in memory is fixed and its contents can be executed, attackers can use ROMMON to bypass the ASLR mechanism. The attack program first jumps to ROMMON with a fixed offset, and uses the return-oriented programming technology (ROP) to find the appropriate gadget to write the shellcode to any writable executable memory area. Because ROMMON's memory is fixed, the location where the shellcode is written can also be determined by the offset. In this way, the attacker can accurately find the jump address of the shellcode required for the exploit, thereby achieving the purpose of the vulnerability attack.

It should be pointed out that different versions of ROMMON and the specific model of IOS will lead to differences between the fixed offset of ROP and the ROP attack chain. Therefore, when bypassing ASLR through ROMMON, you need to know the IOS version number and ROMMON version number.
4. Improvements
With the increasing threat of vulnerability attack, Cisco IOS has gradually introduced a series of vulnerability protection mechanisms, but the threat of vulnerability attack still exists. In order to alleviate the threat of vulnerability attack, IOS needs to add more measures to improve the vulnerability protection mechanism. For example, CheckHeaps verifies the value of Nextptr pointer during integrity verification to prevent its content from being maliciously modified. Integrity check is added when memory blocks are released and merged to prevent attackers from maliciously attacking IOS by constructing fake blocks. Strictly distinguish the data and code areas, and improve the DEP defense mechanism in combination with hardware, so that even if the corresponding access control flag is modified, an attacker cannot change the access permissions of the corresponding memory. It is also possible to verify the modified flag bit, which can only be modified if the operation is legal. Adopt a more thorough ASLR technology for IOS, so that the address loaded by ROMMON is not the same; At the same time, access control is performed on ROMMON. If no IOS restart occurs after IOS operation, the program execution flow shall not jump to ROMMON for execution.

In addition, we should update the IOS version in time, introduce a better detection system, and maximize the security of Cisco routers.

5. Conclusions
With the continuous expansion of the network scale, the security of routers as the core equipment of the network has gradually become the focus of the security industry. Through the introduction of this article, we can find that there are still many deficiencies in Cisco IOS vulnerability protection, which need to be paid enough attention to and timely measures are taken. In the next step, we will deeply study the operating mechanism of Cisco IOS and the vulnerability attack methods for Cisco IOS, and propose better protection measures to protect Cisco routers from vulnerability attacks to the greatest extent.

Acknowledgment
The authors thank to the support of the Science and Technology Commission Foundation Strengthening Project, number 2019-JCJQ-ZD-113.

References
[1] Michael Lynn 2005 Cisco IOS Shellcode BlackHat USA
[2] Felix Linder 2009 Cisco IOS Router Exploitation BlackHat USA
[3] Chris Anley, John Heasman, Michael Lynn, Gerardo Richarte 2007 The Shellcoders Handbook 2nd Edition 339-374
[4] Xiaoyan Su, Dongying Wu, Da Xiao, Yuxiang Han 2011 Research on Cisco IOS Security Mechanisms
[5] Bollapragada V, White R, Murphy C. 2000 Inside Cisco IOS Software Architecture 1-37
[6] Michael Lynn 2010 The Holy Grail: Cisco IOS Shellcode and Exploitation Techniques
[7] George Nosenko 2015 CISCO IOS SHELLCODE: ALL-IN-ONE