Liu *et al. Cybersecurity* (2023) 6:45
https://doi.org/10.1186/s42400-023-00164-x

Cybersecurity

RESEARCH

Open Access

A buffer overflow detection and defense method based on RISC-V instruction set extension

Chang Liu[1,2], Yan-Jun Wu[1,3*], Jing-Zheng Wu[1,3] and Chen Zhao[1,3]

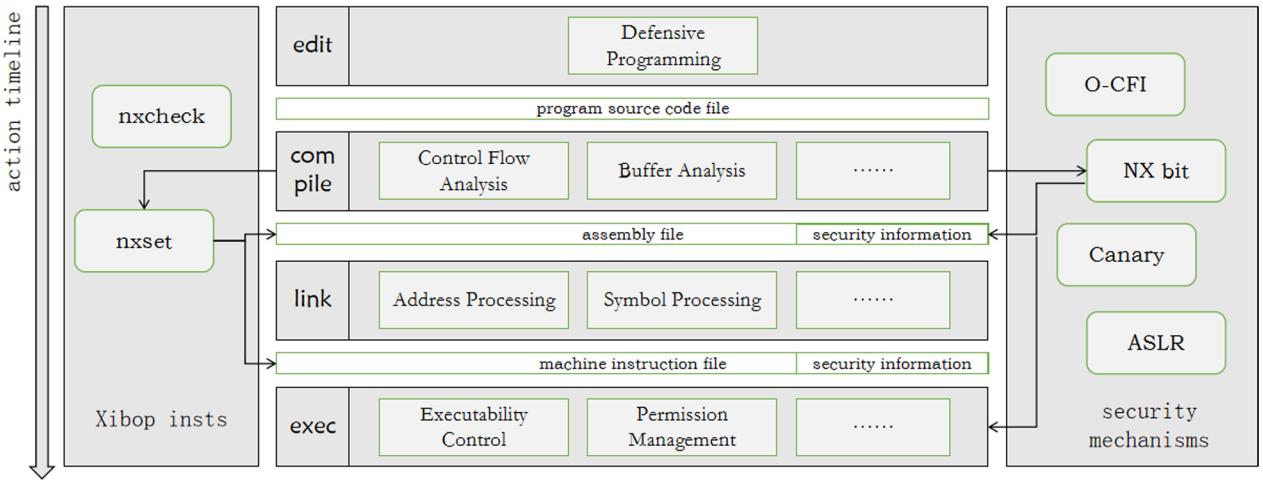
Abstract

Buffer overflow poses a serious threat to the memory security of modern operating systems. It overwrites the contents of other memory areas by breaking through the buffer capacity limit, destroys the system execution environment, and provides implementation space for various system attacks such as program control flow hijacking. That makes it a wide range of harms. A variety of security technologies have been proposed to deal with system security problems including buffer overflow. For example, No eXecute (NX for short) is a memory management technology commonly used in Harvard architecture. It can refuse the execution of code which residing in a specific memory, and can effectively suppress the abnormal impact of buffer overflow on control flow. Therefore, in recent years, it has also been used in the field of system security, deriving a series of solutions based on NX technology, such as ExecShield, DEP, StackGuard, etc. However, these security solutions often rely too much on the processor architecture so that the protection coverage is insufficient and the accuracy is limited. Especially in the emerging system architecture field represented by RISC-V, there is still a lack of effective solutions for buffer overflow vulnerabilities. With the continuous rapid development of the system architecture, it is urgent to develop defense methods that are applicable to different system application environments and oriented to all executable memory spaces to meet the needs of system security development. Therefore, we propose BOP, A new system memory security design method based on RISC-V extended instructions, to build a RISC-V buffer overflow detection and defense system and deal with the buffer overflow threat in RISC-V. According to this method, NX technology can be combined with program control flow analysis, and NX bit mechanism can be used to manage the executability of memory space, so as to achieve a more granular detection and defense of buffer overflow attacks that may occur in RISC-V system environment. In addition, The memory management and control function of BOP is not only very suitable for solving the security problems in the existing single architecture system, but also widely applicable to the combination of multiple heterogeneous systems.

Keywords RISC-V, Operating system security, Buffer overflow, Control flow hijacking, NX bit, Xibop

*Correspondence:
Yan-Jun Wu
yanjun@iscas.ac.cn
Full list of author information is available at the end of the article

Graphical abstract



Introduction

Buffer overflow attack (One 1996) is a typical attack method to threaten the memory security of operating system. By making use of the loopholes such as boundary check lacking in the source program written by the memory insecure language, it can break through the buffer capacity limit, cover the data content in other memory regions, and then destroy the integrity and correctness of a program. Due to low implementation difficulty and convenient operation, buffer overflow attacks have developed rapidly, and their influence and harmfulness have become increasingly prominent. According to the statistics of China National Vulnerability Database of Information Security (CNNVD for short) (CNNVD 2022), in all the 12 months of 2022, the total number of security vulnerabilities has exceeded 24,800, including more than 2600 buffer errors, accounting for more than 10% (Table 1). The buffer overflow attack has the characteristics of rewriting the memory content, which makes it widely used in attacks such as program control flow hijacking, and further increases the security threat to the system. For example, the return oriented programming (ROP) attack uses the buffer overflow to cover the correct function return address information, hijacks the control flow and jumps to the code fragment designated by the attacker when the program executes function return codes, thus completing the malicious operation(Wang et al. 2019). Therefore, academia and industry have been actively looking for effective methods to deal with buffer overflow attacks, so as to curb and mitigate the adverse impact of buffer overflow on system security as much as possible.

In attack scenarios such as program control flow hijacking, the main goal of buffer overflow attack is to guide the control flow to the incorrect code for execution. These codes are located outside the Control Flow Graph (CFG for short). Some are external malicious codes prepared by attackers, which will break the system from normal work flow after execution and act directly according to the attacker's intentions, such as code injection attacks (Pan et al. 2021). Some codes that exist in the system but should not be executed, they will cause the system to process program data incorrectly after execution, such

Table 1 CNNVD vulnerability statistics (2022)

No	Month	Collection	Buffer Errors	Percentage (%)	Rank in all Error Types
1	2022.12	2173	248	11.41	2
2	2022.11	1922	167	8.69	2
3	2022.10	2001	265	13.24	1
4	2022.9	2133	325	15.24	1
5	2022.8	2240	291	12.99	1
6	2022.7	1924	215	11.17	1
7	2022.6	2346	197	8.40	3
8	2022.5	2204	195	9.54	2
9	2022.4	2098	190	9.06	3
10	2022.3	2065	208	10.07	2
11	2022.2	1726	105	6.08	[a]
12	2022.1	2054	210	10.22	3
Total		24,886	2616	10.51	

[a] The monthly report of this month has not been published. The data of this month is the approximate value accumulated from all weekly reports of this month, and the cumulative range is from January 31, 2022 to February 27, 2022

as code reuse attack (Zhao et al. 2021). In this regard, some defense methods based on non executable memory have been proposed. For example, OpenBSD implemented the W ^ X mechanism (Wikipedia 2022) in 2003, so that every page in the address space of a process or kernel space cannot be writable and executable at the same time. Red Hat put forward the defense mechanism Exec Shield (Molnar 2003) for Linux system in 2004 and it has been used ever since. The Exec Shield divides the memory space of a process into executable and non executable segments. It uses the Segment Limit Approach to approximately separate the read and execution permissions to ensure that the return address only points to trusted code. Microsoft also designed a "Data Execution Prevention" (DEP for short) security mechanism for Windows system (Gao et al. 2013). DEP tracks and monitors the portion of system memory which used to store instructions, and when an application program attempts to execute in a memory area which has been designated as non-executable, closes it and issues a notification.

However, the existing solutions rely heavily on the hardware environment and system characteristics, which limits their protection scope. For example, Exec Shield is designed for×86 architecture, and the segment restriction method it depends is also a fuzzy feature of Intel processor; DEP is only applicable to Windows platforms, and Intel has also developed Execute Disable Bit (EDB for short) memory protection technology for their processors to coordinate and support the implementation of DEP functions. Without×86 architecture and Intel processors, these security solutions will be powerless. Therefore, for many non-×86 operating systems, we still need to continue to explore new ways to eliminate threats. In order to achieve this goal, researchers have started some new attempts, combining other technologies (such as StackGuard and SSP (Cowan, et al. 1998)) or adopting new technology application methods (such as CHERI (Davis et al. 2019)), but there are also problems such as limited application scenarios or ease of use, and there is still room for improvement.

RISC-V is a new open source RISC architecture that conforms to the design requirements and architecture development trend of modern information systems. Because of its characteristics of openness, freedom, light weight, convenience, low cost of learning and promotion, strong portability and customization, it is highly suitable for emerging intelligent devices and a variety of intelligent scenarios. In recent years, it has gradually become a major research hotspot in the field of system architecture, and is expected to become the mainstream choice for customizing the next generation operating system. However, the operating systems based on RISC-V architecture also face security challenges such as buffer overflow and control flow hijacking. Developing the RISC-V architecture oriented system memory security defense technology is the inevitable course to promote the further development of RISC-V architecture.

To this end, we have done the followings:

1. We analyzed the characteristics of existing security schemes, and summarizes the relevant key points and basic design ideas of system memory security defense in RISC-V architecture based on the requirements of RISC-V architecture itself.
2. We designed a set of RISC-V memory security related instructions, which is called the Xibop extension. Based on this extension, a buffer overflow detection and defense system for RISC-V architecture is proposed, which is called BOP method.
3. Around BOP method, we also discussed the possibility of supporting multiple security mechanisms and establishing RISC-V memory security defense system. By using the BOP method, we can fully leverage the advantages of RISC-V architecture and achieve better software hardware collaboration when deal with security problems.

The content of this paper is arranged as follows: Sect. "Introduction" introduces the buffer overflow security challenges faced by RISC-V architecture; "Research background" analyzes some existing buffer overflow prevention methods; Sect. "Memory security in RISC-V architecture" briefly summarizes the basic idea of RISC-V architecture for memory security defense; Sect. "Xibop instruction set extension for RISC-V memory security" focuses on the Xibop instruction set extension; Sect. "BOP Method and its Implementation" discusses the specific implementation of the BOP method in the system; Sect. "Discussion" discusses the further work required for BOP method to support multiple security mechanisms and establish RISC-V memory security system; Sect. "Conclusion" summarizes the full text.

Research background

In order to suppress the buffer overflow attacks that system may encounters, the academic and industrial circles have put forward a variety of coping technologies and measures, including the use of memory non-executable (NX for short) attribute, Address Space Layout Randomization (ASLR for short), Canary, and Control Flow Integrity (CFI for short). Among them, using memory non-executable attribute is a typical technology that appeared earlier and has been developed and practiced in a variety of specific systems. The memory security schemes, including Exec Shield, DEP, etc., all use NX as their technical basis, but all depend on the specific

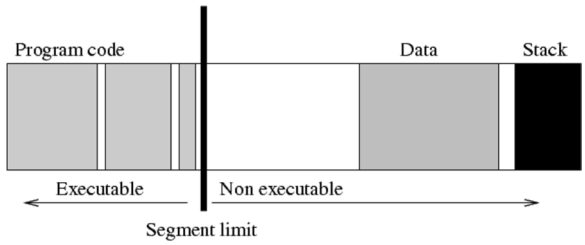


Fig. 1 Memory layout of exec shield process

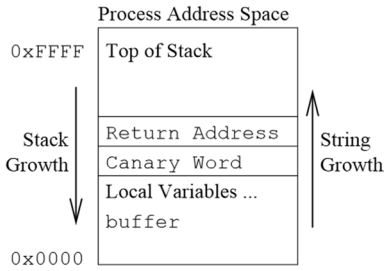


Fig. 2 Principle of Canary technology (Cowan et al. 1998)

hardware environment or system features.In addition, there are also some efforts (StackGuard, SSP, CHERI, etc.) to adopt other technologies or technology application methods. Although they need to be improved, they have also played a certain role in the follow-up exploration.

Exec shield

Exec Shield (Molnar 2003) is a security technology for Linux x86 kernel, which was disclosed by Red Hat in 2004 to prevent stack, buffer or function pointer overflow. In the x86 architecture, there is no difference in the permissions to read or execute code from a part of memory, as a result, Exec Shield needs to distinguish between the two. So Exec Shield tracks the executable mapping specified by the application, and maintains a "maximum executable address" value based on this, forming a "segment limit", thus dividing the memory address into an executable and a non-executable parts, approximately separating the read and execute permissions (Fig. 1). By this setting, ensure that all program codes are on the low address side of the limit, and all data are on the high address side; When the program violates the execution authority and crosses the segment limit, a segmentation error will be triggered to terminate.

The "one size fits all" approach of Exec Shield makes its defense effect completely depend on whether the "segment limit" is properly selected. The memory area near its boundary may lose protection due to inaccurate estimation of "segment limit". In addition, this practice requires that the memory layout must be arranged in strict accordance with the rule of "executable code at low address, data and other content at high address", which has a serious dependency on the system itself, even hardware, and limits the application scope of Exec Shield.

DEP

DEP (Gao et al. 2013) is a data execution protection service developed by Microsoft for the Windows platform to prevent the execution of malicious inserted code. In the "Help and Support" function of Windows system, we can find Microsoft's official description of DEP:

Data Execution Protection (DEP) is a security feature that helps to protect your computer from viruses and other security threats. Harmful programs may attack Windows by attempting to run (also known as "execute") code reserved in the computer's memory for Windows and other authorized programs. These types of attacks can damage your programs and files.

DEP can help to protect your computer by monitoring programs to ensure that they use computer memory safely. If DEP notices that a program on your computer is using incorrect memory, it will close the program and notify you.

In actual operation, DEP marks the memory location which only contains data as NX (non-executable); If an application attempts to execute code from a memory location marked NX, it will be blocked by DEP to achieve protection. In terms of implementation, the Windows system automatically adds a group of special pointers to the data objects stored in the memory on the software level, and the EDB memory protection technology developed by Intel processor is used to cooperate on the hardware level.

DEP has a good defense effect when against code injection type attacks; However, because only data is protected, it is easy to be bypassed by code reuse type attacks. Also, the buffer overflow protection provided by DEP has some side effects, that is, the affected applications are often suspended. Frequent triggering may lead to other types of attacks, such as denial of service attacks.

StackGuard and SSP

StackGuard (Cowan et al. 1998) is a specific memory protection mechanism provided by GCC compiler, which uses Canary technology to protect stack security. Canary is a small special character set between the buffer and control information such as EBP. When a buffer overflow occurs, this character will be overwritten and destroyed first, as shown in Fig. 2. Therefore, it is only necessary to detect whether the value of Canary has changed before the jump execution to determine whether an overflow has occurred.

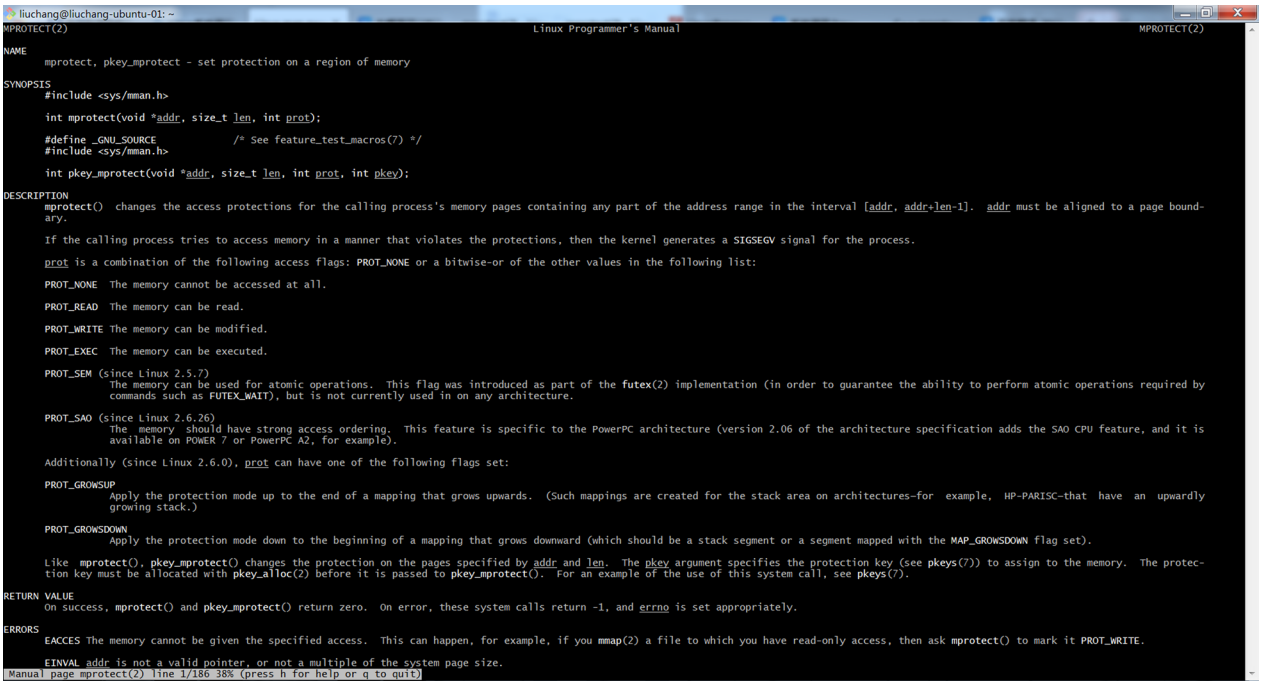


Fig. 3 Extract from the official document of system call mprotect

SSP (Stack Smashing Protection) is a further development of StackGuard. It strengthens the storage security of Canary words and provides a variety of methods to generate Canary words. But in principle, it is similar to StackGuard.

As its name describes, StackGuard and SSP only protect against stack overflow and do not pay special attention to heap space and other overflow problems. In addition, if the area where the Canary word is located is managed to bypass (such as writing out of bounds), or the Canary value is disclosed, the defense effect of the application cannot be achieved.

mprotect

mprotect is a system call provided by the Linux operating system to set the protection properties of a memory region. When a user wants to modify a protection property (readability, writability, executability, etc.) of a memory region, he can specify the starting address addr of the corresponding memory page for the region, the total length len from addr to region ending, and the property value prot. And then, call mprotect(addr, len, prot) to complete the modification. The description of the system call can be found in the official Linux documentation through the Linux system command man mprotect (Fig. 3).

The document states that the starting address of the region protected by mprotect must be aligned to a certain page boundary, so it cannot accurately specify a memory region at any location for protection. In addition, as a system call, mprotect can only function on Linux systems, and cannot be directly used in other non-Linux system environments.

CHERI

CHERI (Davis et al. 2019) is a system security research jointly completed by Cambridge University, Stanford Research Institute and other institutions. It aims to achieve fine-grained memory protection and highly scalable software partitioning, and significantly improve system security. CHERI proposes a hardware supported data format "architecture capability" for representing integers and pointers in memory insecure languages, and protecting underlying data addresses. "Capability" consists of an integer address and a same size metadata, and is associated with a 1-bit validity tag in a register or memory (Fig. 4). Base on the content of the metadata and the validity tag, hardware can then control the operations to the address so that the security can be guaranteed.

However, in order to support its new concept of abstract capabilities, it involves the adjustment and modification of a variety of software and hardware, and there is still much room for improvement in ease of use and other aspects.

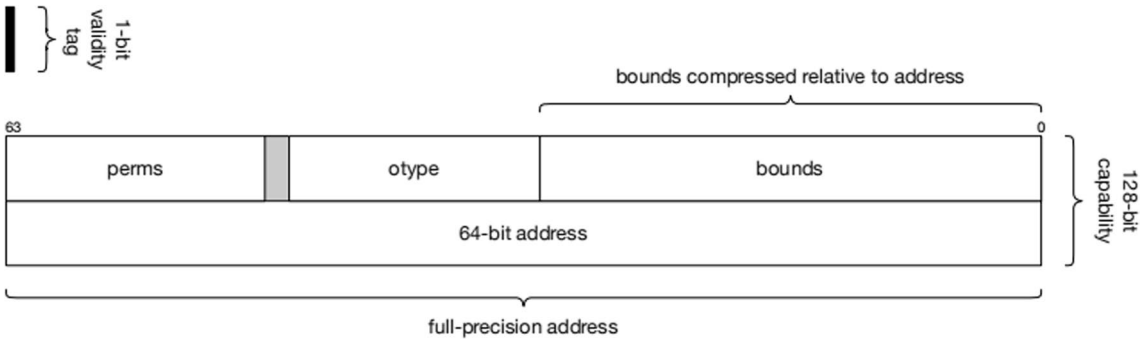


Fig. 4 128-bit capability for 64-bit address in CHERI (Watson et al. 2019). The permissions(perms), object type(otype), and authorization bounds(bounds) are jointly formed the metadata part of this capability

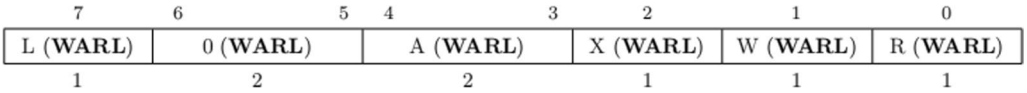


Fig. 5 The structure of RISC-V PMP configuration register (Waterman et al. 2023). The R, W, and X fields indicate the readability, writability, and executability settings of the target address, respectively. The A field indicates the address matching mode, and the L field indicates whether the PMP entry is locked and becomes not writable

PMP

Physical Memory Protection (PMP) is a fundamental security mechanism introduced by the RISC-V Privileged Instruction Set Specification (Waterman et al. 2023). It is used to limits the physical addresses which are accessible by software running in hardware threads (harts). PMP combines an 8-bit configuration register (pmpcfg) and an MXLEN bit[1] address register (pmpaddr) into a PMP entry to record and manage the protection properties of a physical memory region. The A field of the configuration register (Fig. 5) specifies a method that determines the physical memory address range to be protected through the corresponding address register.

PMP is a relatively basic protection measure applied to physical addresses, so it often requires preparation work such as address translation and entering the M privilege level in advance. Moreover, PMP requires that the protected memory addresses must be physically continuous. These all imply the feature that PMP may need to be used in conjunction with other processing logic.

Memory security in RISC-V architecture

Main features

RISC-V is a new RISC architecture. Compared with other existing architectures, RISC-V has its unique features in memory security problems.

First of all, RISC-V, as an emerging architecture, lacks enough mature hardware and software security solutions to match it. This shows, on the one hand, that solving the memory security problem in RISC-V often requires starting from scratch, with considerable space for technology implementation; On the other hand, it also means that to solve the RISC-V memory security problem, it is usually necessary to comprehensively consider the requirements and capabilities of both software and hardware, and realize the defense scheme with software-hardware collaboration.

Secondly, RISC-V, as a reduced instruction set architecture, has the advantages of controllable cost, portability and customization, and has good applicability in heterogeneous, modular, functional specialization and other application scenarios. Therefore, systems based on RISC-V architecture may have multiple memory environments, that are significantly different in address space size, growth direction, available addressing methods, segment distribution, address read/write/executable characteristics, and supported permission modes, and are prone to expose a larger attack surface. Due to this reason, the memory security defense scheme of RISC-V is required to be either universal to a certain extent, or comprehensive with multiple specific mechanisms for various memory environments, with sufficient hierarchy or multi-dimensional property.

In addition, RISC-V, as a reduced instruction set architecture, has stronger regularity in its instructions. It has strict specifications in instruction format, instruction alignment, etc., which greatly reduces the design complexity of hardware support, improves the decoding efficiency, and reduces the time, space, and energy

[1] A generic representation of the register bit width under M privilege level. This can be 32 or 64 bits depending on the actual situation.

costs when completing instruction functions. On the one hand, it simplifies the specific implementation of RISC-V defense scheme to a certain extent, while it also implies a layer of constraint: the implementation of RISC-V defense scheme should be as consistent as possible with the original style of the system; On the other hand, when it is really necessary, RISC-V's energy efficiency features can be used to balance the negative impact of inefficient solutions that other architectures usually can't bear, making the design and implementation of RISC-V defense solutions more optional.

At the same time, RISC-V will also show some commonalities with other architectures in some aspects when dealing with memory security threats. For example, the basic principle of buffer overflow attack is to illegally overwrite other data by breaking the buffer capacity limit. Therefore, whether in RISC-V or other architectures, buffer boundary must be one of the key points of defense. This commonality makes the design of RISC-V memory security defense scheme still able to draw inspiration from similar schemes of existing architectures and absorb some of their advantages for own use.

Basic ideas

According to the main characteristics of the memory security issues in RISC-V architecture above, the RISC-V memory security defense scheme can have the following basic design ideas:

1. Based on the characteristics of RISC-V architecture itself, and with RISC-V instruction set as the link, to design the defense scheme of software-hardware collaboration from both the software and the hardware sides together.
2. The method of combining multiple security mechanisms is adopted to design and implement a multi-dimensional defense scheme from several different levels for various memory environments that may exist in the target system.
3. Prefer the design based on RISC-V's native characteristics and keeping the original style of the system, and use the existing resources and mechanisms of the system effectively to solve specific security threats. However, when necessary, different external software and hardware modules can also be introduced to support it.
4. On some common problems of memory security, we can absorb the advantages of similar schemes of existing architectures and make them fit for RISC-V.

Specifically, the following ideas can be followed when solving the buffer overflow problem in RISC-V architecture:

1. According to each time that the threats occur and develop, deploy targeted defense mechanisms respectively to form a multi-level comprehensive software-hardware coordinated defense system. For example, in the process of program development, use defensive programming and other means, such as manually checking the array bounds, to avoid obvious security risks from the code text level; In the process of program compilation and testing, the relevant tool chain can effectively supervise and control, to extract the possible overflow threat information and deal with it; In the program execution phase, RISC-V security hardware uses a variety of security mechanisms to protect the buffer that may be threatened, and so on.
2. Based on the RISC-V instruction set specification, use special custom instructions to constrain the specific implementation of RISC-V security hardware. On the one hand, it helps the security defense scheme to make full use of the native features of RISC-V and maintain the RISC-V system style. On the other hand, it provides an unified adaptation method for different memory environments, which improves the universality of the security defense scheme. In addition, the RISC-V tool chain can also be targeted and optimized accordingly to better achieve software-hardware collaboration.

Xibop instruction set extension for RISC-V memory security

Around the basic ideas proposed in Sect. "Basic ideas", we hope to build a memory security defense system based on the RISC-V instruction set, which combines hardware and software in the RISC-V architecture, to mitigate the threat of buffer overflow attacks on the RISC-V architecture. To this end, we designed a set of RISC-V instructions related to memory security, called Xibop instruction set extension. The Xibop extension includes some custom instructions to support various memory defense methods, such as ALSR (Marco-Gisbert et al. 2019), Canary (Krerk et al. 2008), O-CFI (Mohan et al. 2015), etc. In this article, we focus on using the Xibop extension to implement the defense mechanism of non-executable memory.

Design overview

Non-executability is a feature of storage media, including memory, which means that the corresponding content can only be used for data access, not for code execution. When existing systems implement this feature, they often need to maintain special tag data to control the non-executability of certain memory areas. When a

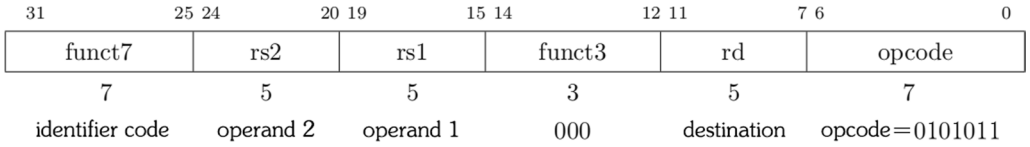


Fig. 6 Encoding format of Xibop extended instructions

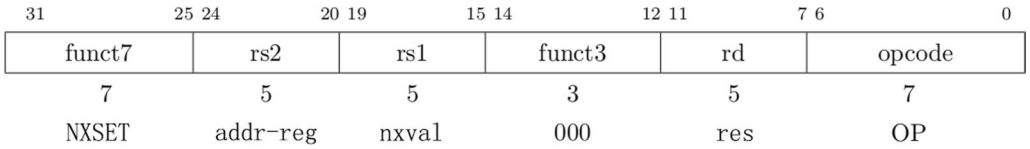


Fig. 7 Format of nxset instruction, where opcode is 0101011, indicates that the instruction belongs to Xibop; instruction identify code funct7 is 0000100, means this is a nxset instruction; field funct3 is 000

piece of memory is marked as non-executable, accessing to any location within the memory range will throw an exception.

Therefore, the core behavior of the defense mechanism based on non-executable memory is to mark the non executable of a specified memory range. To this end, the Xibop extension introduces the nxset instruction, which is used to set the non-executability of the target memory. Depending on the settings, the target memory will become non-executable or revert to an executable state.

The Xibop extension also introduces the nxcheck instruction, which is used to detect the executability of the target memory. This instruction is not necessary, but it can provide a certain degree of convenience in some scenarios (such as testing). In addition, in order to facilitate application, Xibop also introduces some pseudo instructions: setnx and clrnx, which are used to turn on and off the non-executability of the target memory, respectively, and they will eventually be interpreted as nxset instructions; setnxr and clrnxr, are respectively used to enable and clear the non-executability of a continuous target memory range, and they will eventually be interpreted as a microprogram containing nxset instructions.

In the aspect of instruction coding, Xibop extension uses the R-type encoding format to divide the instructions into six fields: opcode, rd, funct3, rs1, rs2, and funct7. The Xibop extension uniformly uses 0101011 as the opcode (opcode field) of instructions to occupy the minimum space for custom opcodes; The funct7 field is used as the instruction identifier to distinguish different instruction functions. The rd field, rs1 field and rs2 field represent the parameters of the instruction, which are respectively the destination register, the first operand register and the second operand register. The funct3 field is used to indicate the type distribution of operands. However, in the current design, only 32-bit register type operands are used, so the funct3 field is fixed to 000. As shown in Fig. 6.

Xibop extension is designed as an XLEN independent extension, that is, it can work in both RV32 and RV64, or even RV128 in the future. When it comes to 64-bit or wider architecture, funct3 field should play a role in explaining how to use the operands.

For the privilege mode of instructions, Xibop extension is designed for the non-privilege level use and works under U mode in principle to achieve the similar effect as S privilege level. However, when implementation, the M privilege level can also be used according to actual needs (for example, in some simple system environments that do not support the U mode).

Instruction nxset

The nxset instruction of the Xibop extension is used to set the non-executability of the memory at the specified address. Its syntax is: nxset rd, rs1, rs2. Among them, rd is the destination register for storing instruction execution results, and you can judge whether the non-executable is correctly set according to whether the value of rd is zero. rs1 is an operand register that stores NX values. A value of 1 indicates that the target memory is set to be non-executable, and a value of 0 indicates that the target memory is set to restore the executability. rs2 is the operand register that stores the target memory address, indicating that the corresponding operation will be performed on the address. The encoding format of the nxset instruction is shown in Fig. 7.

In terms of hardware, the nxset instruction can be implemented as shown in Fig. 8.

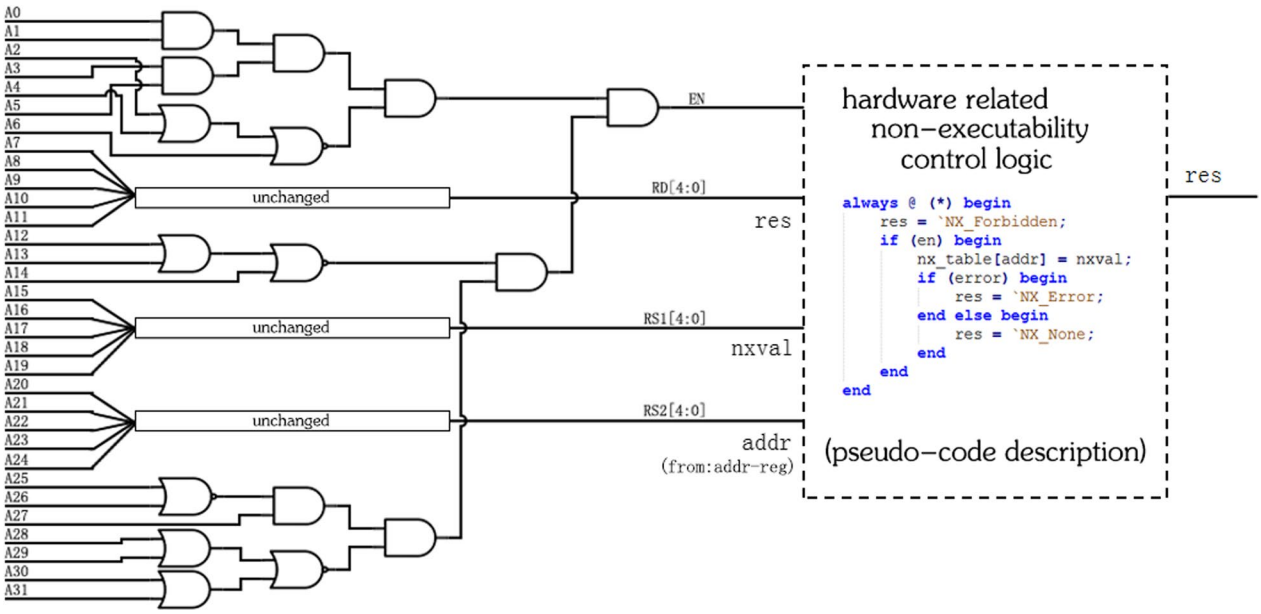


Fig. 8 An implement method of nxset instruction in RISC-V security hardware. Left part describes the instruction decoding process. opcode, funct3 and funct7 fields are combined to set the enable signal(en). When enabled, nxval from rs1 will be stored to a place corresponding to the value of addr-reg from rs2. res is set according to the execute result and is returned

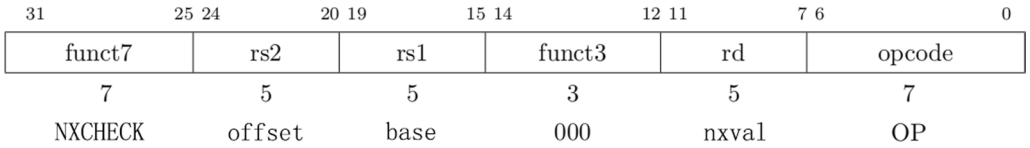


Fig. 9 Format of nxcheck instruction. similiar with nxset, the opcode is 0101011, which means the instruction belongs to Xibop extended instruction set; instruction identify code funct7 is 0000101, to tell this is a nxcheck instruction; field funct3 is 000

Instruction nxcheck

The nxcheck instruction provided by Xibop is used to detect the executability of the target memory. Its syntax is: nxcheck rd, rs1, rs2. Among them, rd is the target register for storing the target memory's executability. We can judge whether the target memory is not executable according to whether the value of rd is 1. rs1 and rs2 are operand registers used to form the target memory address. Generally, rs1 stores the base address of the target memory address, while rs2 stores the corresponding address offset. The encoding format of the nxcheck instruction is shown in Fig. 9.

In terms of hardware, the nxcheck instruction can be implemented as shown in Fig. 10.

Pseudo instruction setnx and clrnx

The setnx pseudo instruction of the Xibop extension is used to set the target memory to be non-executable. Its syntax is: setnx addr. Where addr represents the address of the target memory. This instruction will be interpreted as an instruction: nxset rd, 1, addr.

Similarly, the clrnx pseudo instruction of the Xibop extension is used to set the target memory to be executable. Its syntax is: clrnx addr, and it will be interpreted as an instruction: nxset rd, 0, addr.

Pseudo instruction setnxr and clrnxr

The setnxr pseudo instruction extended by Xibop is used to set a continuous target memory range as non-executable. Its syntax is: setnxr from, to. Where, from represents the low address boundary of the target memory range, and to represents the high address boundary of it. This instruction will be interpreted as a microprogram in Fig. 11a.

Similarly, the Xibop extended clrnxr pseudo instruction is used to set a continuous target memory range as executable. Its syntax is: clrnxr from, to, and will be interpreted as the microprogram in Fig. 11b.

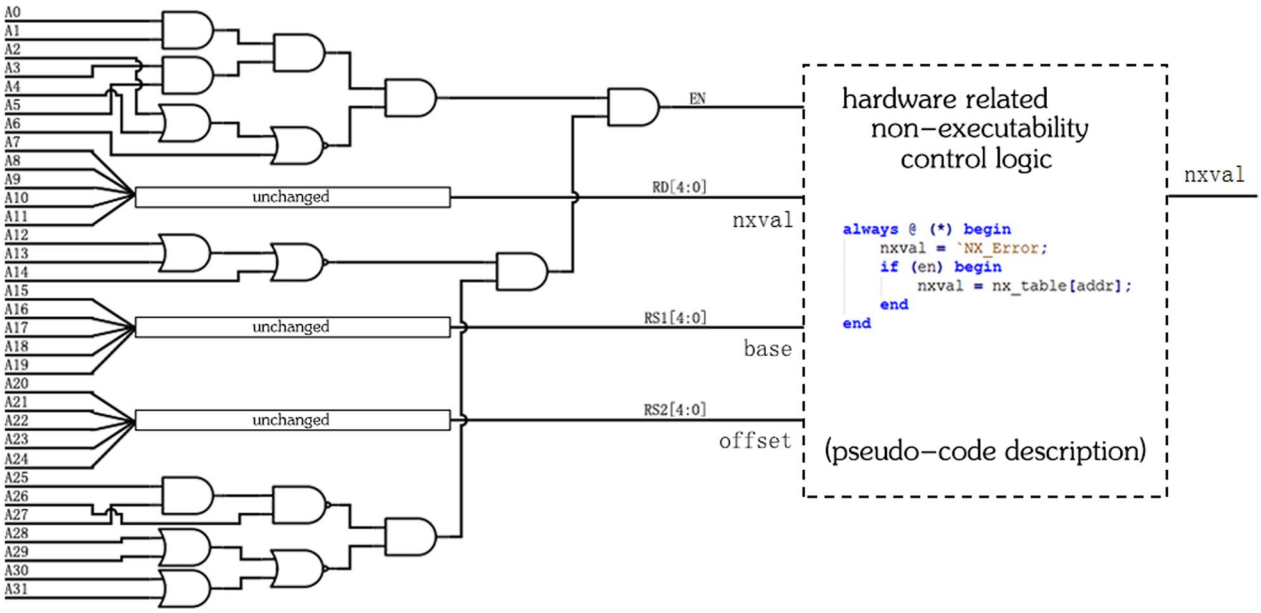


Fig. 10 An implement method of nxcheck instruction in RISC-V security hardware. Left part describes the instruction decoding process. opcode, funct3 and funct7 fields are combined to set the enable signal(en). When enabled, read the nxval as result from the store place corresponding to addr-reg. If error occurs, nxval will be the error code

```
.L2:
    nxset rd, 1, from
    addiw from, from, 16
    bge from, to, .L2
```

(a)

```
.L2:
    nxset rd, 0, from
    addiw from, from, 16
    bge from, to, .L2
```

(b)

Fig. 11 Explanation of setnxr and clrnxr pseudo instructions. Here ". L2" is a jump tag, which may have different names in specific implementations

BOP method and its implementation

Based on the Xibop extension, we propose a buffer overflow detection and defense system for RISC-V architecture, called BOP method. This method can be used to implement specific memory security defense mechanisms such as those based on No eXecute Bit (NX bit).

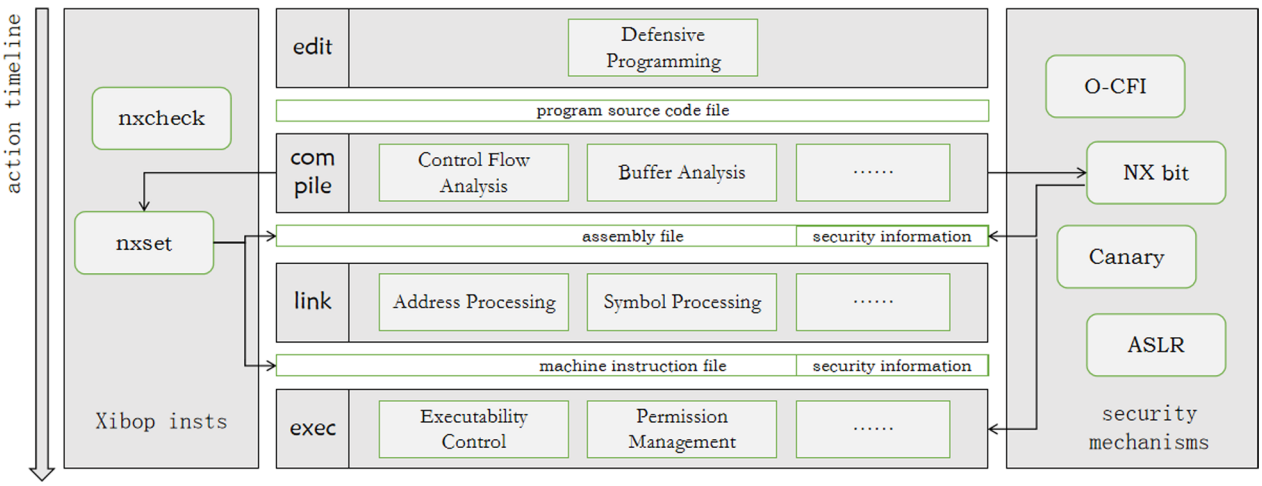


Fig. 12 Overall architecture of BOP design

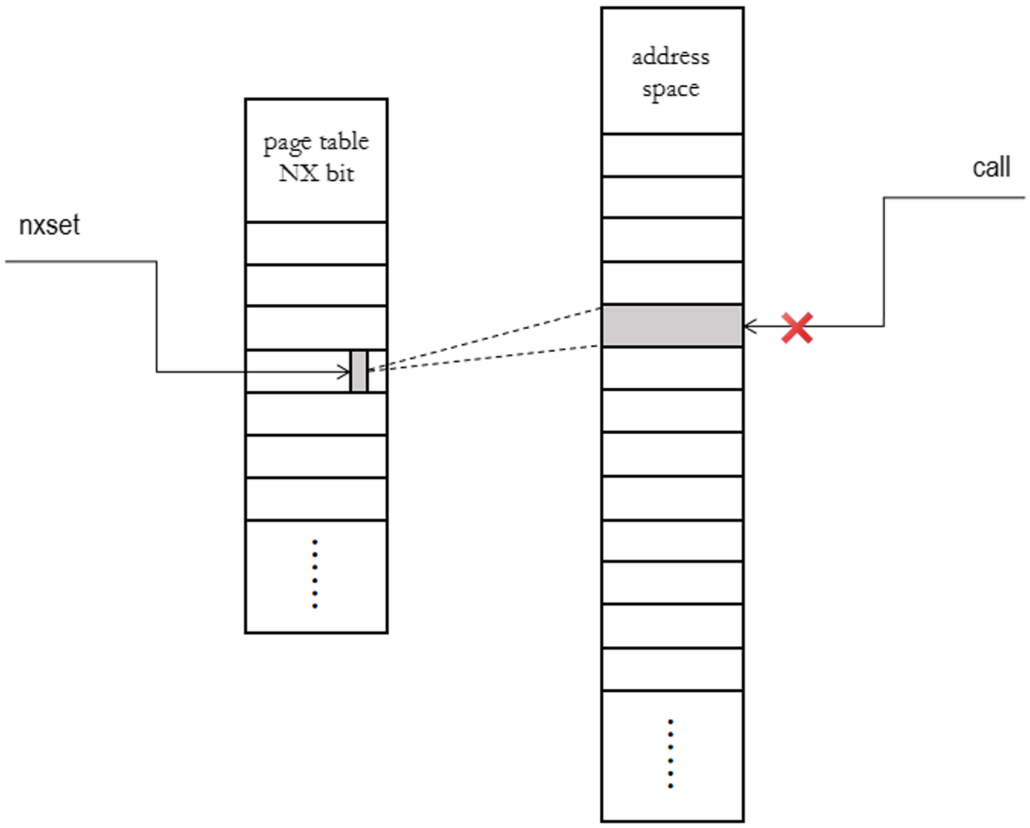


Fig. 13 Main workflow of NX bit based non-executable memory defense mechanism

Overall architecture

The BOP method is the synergy of several hardware and software memory defense mechanisms. Its main idea is to take targeted defense in turn on multiple links that may cause memory security threats, and combine all the memory defense mechanisms to cooperate with each other. The defense mechanism in each link can not only complete its own functions, but also provide information for subsequent links as much as possible to improve the overall defense capability and efficiency. Figure 12 outlines the overall design architecture of the BOP method.

According to actual needs, resource constraints and other factors, the BOP method can load any security defense mechanism it supports. As for the defense mechanism of non-executable memory, BOP uses NX bit to control the executability of system memory. NX bit is a flag bit used to identify the executable permission of memory space, which is mostly in page table entries. By manipulating NX bits, program code can be prohibited or allowed to execute from a specific memory range, thus constraining the direction of control flow. Figure 13 shows the main workflow of this mechanism.

Support from software side

BOP is a security method based on RISC-V Xibop extension instruction set. Therefore, compiler, assembler and other tools need to be modified accordingly so that the method can be correctly recognized and accepted by the system. A feasible method is to modify the official tool chain RISCV-GNU-TOOLCHAIN provided by RISC-V, add the description information of Xibop extension instructions, and insert BOP related instructions according to the code characteristics in the compilation phase, so as to set or clear specific NX bits as required. Algorithm 1 describes the insertion process of the instruction nxset.

Algorithm 1: insert instruction NXSET

```
1  cfg = build_cfg()
2  foreach (bb, cfg) {
3      foreach (stmt, bb) {
4          if (is_call(stmt)) {
5              target = get_call_target(stmt)
6              range = update_nx_range(target)
7              nxset(stmt, range, nx_val=1)
8          } else if (is_jr(stmt)) {
9              target = get_jump_target(stmt)
10             no_error = check_nx(target)
11             if (no_error) {
12                 range = update_nx_range(stmt)
13                 nxset(stmt, range, nx_val=0)
14             }
15         }
16     }
17 }
18
```

Algorithm 1: The process of nxset instruction participating in program compilation

In this process, the BOP first analyzes the control flow graph of the program in the compile period, finds out the code fragments to be protected, such as function jump (involving the stack space), malloc (involving the heap space), and determines the memory space associated with this fragment that should not be executed. BOP will record the symbols which can describe the boundary of that memory space first, and update these symbols with their address value later in the link period. Then, before entering this segment, insert the nxset instruction to set the NX bit so that the relevant memory is made not executable; After exiting the fragment, insert the nxset instruction to clear the NX bit and restore the executability of the relevant memory.

Implement at hardware side

In terms of implementation, the BOP method relies on an independent security component which contains BOP modules. The BOP security component is a physical component used to implement the module, including related circuit logic such as processing NX mechanisms, and related parts such as additional registers. This component is connected with both the processor and memory device to detect the memory status and control the memory access request of the processor at any time. At the same time, the decoding logic of the processor is modified to enable it to recognize and execute various functions of Xibop extended instructions. The specific transformation method is consistent with the design idea (Fig. 5, Fig. 7, etc.) mentioned in Chapter 4.

The BOP module is used to implement various BOP memory security defense mechanisms at the hardware side. The non-executable memory defense mechanism mainly involves memory access request filtering, NX bit management, exception alarm and other functions, as well as some preprocessing operations. The BOP security component on which it relies provides the basic isolation function, and also can be equipped with more self-protection schemes, like anti-electromagnetic interference devices, as needed.

During an instruction cycle, the fetching module will perform a memory access, take the XLEN bit[2] length data from the address pointed to by the PC register, and give it to the decoding module as an instruction; The decoding module will decode this instruction, analyze its operation code, operand, operand type and other information, and send it to the execution module; The execution module

[2] A general representation of the register bit width, which may be 32 or 64 bits depending on the actual situation; This is used to represent fetching data of the same length as the register bit width at one time.

executes the corresponding operation according to the operation code, and may access the memory again at this time. After the introduction of the BOP method, the memory access requests of the fetching module and the execution module will first be filtered by the BOP module. After confirming that there is no exception in the target memory address, they will be passed to the memory access module to complete the normal memory access process. If the BOP module, by analyzing the memory access type, querying NX bit and other processing logic, judges that the program has initiated an incorrect memory access request and is about to execute the data content in the non-executable memory, it will block this memory access request and raise an instruction exception. This process is shown in Algorithm 2.

Algorithm 2: non−executable memory defense

```
1  inst = fetch()
2  decode(inst)
3  if (inst[opcode] == OP_NXSET) {
4      nxval = inst[rs1]
5      addr = inst[rs2]
6      set_mem_nx_attr(addr, nxval)
7  } else if (will_access_memory(inst)) {
8      addr = get_access_addr(inst)
9      nxval = get_mem_nx_attr(addr)
10     if (no_execute(nxval)) {
11         stop_execute(inst)
12         throw_exception()
13     } else {
14         continue_execute(inst)
15     }
16 } else {
17     continue_execute(inst)
18 }
```

Algorithm 2: The process of BOP security component to execute non-executable memory defense.

In addition, for systems that do not yet support the NX bit management mechanism, the BOP security component will maintain a set of NX registers, simulate the mapping relationship between memory addresses and NX bits, and save the values of each NX bit. In this group of registers, a NX bit can be uniquely identified by its bit number b and the register number ln where the register is located; While, there is a specific mapping relationship between (b, ln) and the memory address addr corresponding to the NX bit: addr = f (b, ln), so that addr and (b, ln) pair can convert each other. In the specific implementation, the conversion process can be quickly completed by using a simple preprocessing logic. Figure 14 describes the design structure of the NX register group organized by several 32-bit registers.

For systems that have supported NX bit management mechanism, BOP can directly connect with the existing management module in the system, and complete the setting of NX bit through the interface provided by the existing module, instead of being responsible for the specific logic implementation of related management work. For example, driving PMP entries by doing preparation works described in Sect. "PMP" may be a lazy approach.

Key to safety assurance

According to the BOP design, the BOP module that manipulates NX bits and filters memory access requests should be located in an independent security component, with a set of working logic and register groups that are not subject to external interference. Even if the

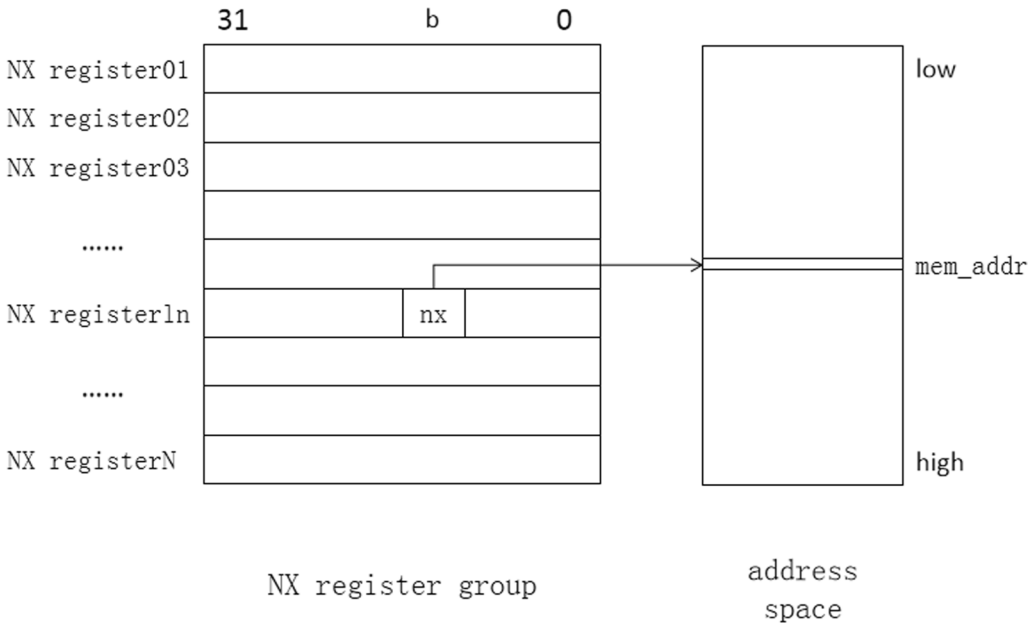


Fig. 14 A design structure of NX register group

instructions or data in the memory have been attacked and tampered with in advance, the effectiveness and accuracy of the BOP module itself will not be affected. At this time, the BOP module will normally detect the exception in the memory, and when the instruction pipeline enters the memory access link, it will stop and cause the instruction exception in time, and turn to the exception handling process.

Discussion

Availability experiment

The experiment mainly consists of two parts: software or toolchain part, and hardware part.

1. Experiments in riscv-gcc v11.1.0, riscv-gas v2.37, riscv-ld v2.37 and riscv-gdb v10.1 show that the modified RISC-V toolchain can recognize the category and boundary of target memory address range correctly, and generate security information to assembly file and machine instruction file as expected.
2. Experiments in Xilinx ARTY A7 FPGA show that the new circuit logic written by verilog can recognize the BOP instructions and update the NX value successfully. As a result, abnormal memory access is stopped and the illegal control flow transfer is terminated.

Performance evaluation

Basically, each instruction that needs to be protected will bring at least two Xibop instructions: usually one setnx instruction, and its corresponding clrnx instruction. Therefore, the specific impact on performance is related to the program structure, especially the distribution of the instructions to be protected. Some of same-type Xibop instructions which work at a same address can be merged by toolchain to simplify work flow and improve performance.

Preliminary measurement and performance experiments show that the proposed method can averagely cause around 7% additional assembly file size growth and has a slight impact on the program's running time. However, such a result is still unstable according to the above analysis.

Future works

The BOP method is still being expanded and improved. This article only focuses on the detection and defense scheme of buffer overflow in RISC-V system by managing memory executability. Further work can be done in the following aspects in the future:

1. Add more sufficient support for RV64. Though our instruction extension Xibop is designed for all width, current work focus more on RV32. The instruction formats and the registers will be changed when XLEN is extended. More comparative experiments will be also implemented.
2. Add support for more defense mechanisms. The goal of BOP is to form a complete set of detection and defense system against buffer overflow and other memory security problems. The defense mechanism based on NX bit management is only one of the func-

tions provided by BOP, with limited scope of application and single defense effect. In the future, multiple defense mechanisms based on Canary, O-CFI and other memory security technologies will be added to form a complete system as shown in Fig. 1 to jointly maintain system memory security.

3. Combined with more processor platforms. BOP is a general security solution for RISC-V architecture, not just designed around a single processor type. For example, BOP can first implement its prototype in simple environments such as tinyriscv(liangkangnan. tinyriscv. 2022), and then expect to play a role in multiple processor environments such as HummingBird(Jayden et al. 2019) and Rocket (Asanović 2016), explore the general method of effective combination with RISC-V processor platform, and strengthen the universal adaptability in different environments.
4. Internal optimization and adjustment. The current version of the BOP design scheme focuses more on the realization of functionality, while there is still much room for optimization in terms of performance, resource consumption, etc. For example, the instruction encoding format can be optimized to further improve the decoding and execution efficiency; By optimizing the management logic of NX register group, reduce the use of registers or expand the range of managed address space, etc.

Conclusion

In this paper, we first describe the threat of buffer overflow to the operating system, and analyze the features of several existing security schemes. Then, combined with the features and requirements of RISC-V architecture, we analyze the basic idea of implementing memory security in RISC-V system, and on this basis, propose a RISC-V memory security defense system based on instruction set extension BOP method. We purposefully designed a set of RISC-V instruction set extension Xibop that can be used to implement the BOP method. Taking buffer overflow detection and defense as an example, we discussed the way RISC-V system supports at both the software and hardware side. In the future, we will continue to improve the BOP system, support more memory security mechanisms, and gradually realize internal optimization and adjustment. We also plan to carry out research on more processor platforms, expand the service scope of BOP method, and explore general implementation solutions for RISC-V architecture as a whole.

Acknowledgements
Here, we would like to express our heartfelt thanks to the review teachers and colleagues who have given all kinds of support and valuable suggestions to the work of this paper.

Author contributions
CL: Propose the theory, design the ISA extension method, run the project, write the paper. Y-JW: Give advices to ISA extension design, support the project. J-ZW: Give advices to the theory and the project, help to improve the paper. CZ: Support the project.

Funding
Strategic Priority Research Program of CAS (XDC05040000).

Availability of data and materials
Not applicable.

Declarations

Competing interests
The authors declare that they have no competing interests.

Author details
[1]Intelligent Software Research Center, Institute of Software, Chinese Academy of Sciences, Beijing, China. [2]University of Chinese Academy of Sciences, Beijing, China. [3]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China.

Received: 8 February 2023 Accepted: 5 June 2023
Published online: 06 September 2023

References

Asanović K, et al (2016) The rocket chip generator. EECS Department, University of California, Berkeley Available: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html
CNNVD (2022) Vulnerability report. Available: https://www.cnnvd.org.cn/home/report
Cowan C, et al (1998) StackGuard: automatic adaptive detection and prevention of buffer-overflow attacks. In: Proceedings of the 7th USENIX security symposium
Davis B, et al (2019) CheriABI: enforcing valid pointer provenance and minimizing pointer privilege in the POSIX C run-time environment. In: Proceedings of ASPLOS'19
Gao Y-C et al (2013) Research on windows DEP data execution protection technology. Inform Secur Commun Privacy 7:4
Jayden, et al (2019) The ultra-low power RISC-V core. Available: https://toscode.gitee.com/riscv-mcu/e203_hbirdv2
Krerk P, et al (2008) Secure bit enhanced canary: hardware enhanced buffer-overflow protection. In: 2008 IFIP international conference on network and parallel computing. pp 125–131. https://doi.org/10.1109/NPC.2008.49
liangkangnan. tinyriscv (2022). Available: https://gitee.com/liangkangnan/tinyriscv
Marco-Gisbert H et al (2019) Address space layout randomization next generation. Appl Sci 9(14):2928
Mohan V, et al (2015) Opaque control flow integrity. In: Proceedings of the 22nd annual network and distributed system security symposium
Molnar I (2003) Exec shield, new Linux security feature. Available: https://lwn.net/Articles/31032/
One A (1996) Smashing the stack for fun and Profit. Available: http://www.phrack.com/issues.html?issue=49&id=14&mode=txt
Pan C-X et al (2021) Method against process control-flow hijacking based on mimic defense. J Commun 42(1):37–47
Wang F-F et al (2019) Overview of control-flow hijacking attack and defense techniques for process. Chin J Netw Inform Secur 5(6):10–20
Waterman A, et al (2023) The RISC-V instruction set manual. volume II: privileged architecture. Available: https://riscv.org/technical/specifications

Watson RNM, et al (2019) An introduction to CHERI. Computer Laboratory, University of Cambridge. ISSN 1476-2986. Available: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-941.pdf
Wikipedia (2022) W%5EX. Available: https://en.wikipedia.org/wiki/W%5EX
Zhao C-Y, et al (2021) Research and implementation of real-time detection method for code reuse attacks. University of Electronic Science and Technology of China

Publisher's Note