RISC-V: Bibliography

Mercier Romain

December 2023 - ??? 2024

Contents

Chapter 1

RISC-V Overview

Chapter 2

RISC-V Fault Injections

Chapter 3

RISC-V Countermeasures

3.1 Note about pointer attack from paper "A Survey on Thwarting Memory Corruption in RISC-V"

This paper target the memory safety against Control-flow hijack, data-oriented attack and information leak. According to this paper, these three attack start by the creation of an invalid pointer for read write operation. This pointer can then be used for three purpose:

1. Modify a code pointer to point on non-intended code. In this case the code pointer is used against the control-flow integrity and make a control-flow hijack attack.
2. Modify a data variable into a malicious value. In this case, the malicious data value is used to attack the control flow integrity, making a data-oriented attack.
3. Study and interpret the pointed data by get it out in order to make information leaks.

3.1.1 Access Validation

Attack against access validation can take two form according to this paper. These attack consist to use, for example, the declaration of a table in C. Two case can be distinguish: i) The spatial violation and ii) the temporal violation. While spatial violation consist to access out-of-bounds memory using the table, the temporal access consist to access to the data after the table is freeing. Several counter-measure a available in the literature to protect the memory against these violation:

- **Fat pointers:** This solution consist to increase the data contain in the pointer, including the bounds of the targeted object. In this way, at each access, the target can be check to be sure that it is in the bounds of the targeted object. Several paper are cited about this type of protection in the paper: Shakti-T, Shakti-MS, CHERI, ...
- **Segmentation:** This method consist to segment the program code and define which part of the memory the segment can access. The segmentation granularity can be adjusted, i.e. the number of segment, according to the sensitivity of the current program.Several paper are cited about this type of protection in the paper: IMPULP, HardScope, Dam, ...
- **Guarded buffers:** This protection is based on GuardLines paper. This method uses small memory regions that CPU exception during memory access. This method target both spatial (around the allocated zone) and temporal (inside the de-allocated zone) safety.

- **Hardware monitoring:** This method do not necessitate compiler support to work. It track conditions of the CPU while it execute program to detect anomalies using hardware monitoring. Several paper are cited about this type of protection in the paper: ARMOR, RiskiM, PHMon, ...

The rest of the section compare the different contribution in term of efficiency, security coverage and performances.

3.1.2 Information Obfuscation

Chapter 4

RISC-V Architecture Examples

4.1 CVA6 (CV32E)

4.2 CV32E40S

Chapter 5

Memory Corruption

5.1 Attack Taxonomy

The memory corruption can be split into three categories according to Szekeres et al:

- **Control-flow hijack** generally uses the rewrite of pointer in the stack memory to divert the execution flow at the end of a called function. This type of corruption can be executed through several different attacks:
  - **Code-injection attack**: The execution flow is diverted to malicious code inject by the attacker in a writable area. This attack can be managed by marking writable buffers as non-executable.
  - **Code-reuse attack**: Similar to the previous except that the control flow in divert toward existing code in the library. This code is not injected by the attacker.
  - **Return-Oriented Programming (ROP)**: Te attacker search for gadgets in the binary that are snippets of code ending by return statement. These gadgets are stitches by the attacker manipulating the return address.
  - **Jump-Oriented Programming (JOP)**: Same than ROP except that the attacker uses indirect jumps with wrong purpose.
- **Data-only attack** This type of attack consists to manipulate data in the code to manage the code execution without flow-execution modification. For example the variable tested by a if statement is modified. Several attack types can refer to these attacks:
  - **Direct Data Manipulation (DDM)**
  - **Non-control data attacks**
  - **Data-oriented Programming (DOP)**
- **Information leak**: In these attacks information are read from sensitive memory. Several attack types can be found:
  - **Format-string attack**: For example, the function printf() can be used. The first argument representing the format-string can be used to read memory by manipulating this format.
  - **Heartbleed bug**:An attacker request more data than was reserved for and access to data memory.

5.2 Test Metrics

This section presents the different metrics used in the literature to quantify the performance and efficiency of the RISC-V Architecture. The metrics can be split into three categories:

1. **Qualitative metrics**:
   - **Security coverage**: Indicates the proportion of memory corruption attack that is countered by the security solution.
   - **Transparent protection**: Indicates if the source code of the program is not required to enforce the protection.
   - **Fully automated**: Indicates if the procedure for adding support to a program is fully automated or not.
   - **OS-independent**: Indicates if the OS is necessary to explicitly support the protection scheme.
   - **Legacy support**: Indicates if unmodified binaries can stiff run on the processor.
   - **Open source**: Indicates if the hardware implementation of the processor is publicly available.
2. **Software-based metrics**:
   - **Binary size increase**: Indicates the overhead in terms of bitstream size to adapt the memory when software protection are added.
   - **Runtime overhead**: Indicates the runtime increase of the system.
3. **Hardware-based metrics**:
   - **Area overhead**
   - **Power overhead**