Stack Redundancy To Thwart Return Oriented Programming Attack

Adrien Michelet, Cyril Bresch, Laurent Amato, Thomas Meyer

Grenoble-INP Esisar, 50 Rue Barthelemy de Laffemas, 26000 Valence, France

Abstract

Software attacks are commonly used against embedded systems in order to access private data or to run restricted services. In this work, we demonstrate some vulnerabilities of usual processor which can be leveraged by hackers to attack a system. We, then, design a countermeasure at hardware level in order to mitigate such vulnerabilities. Finally, the countermeasure is implemented on a FPGA board and evaluated.

Introduction

❑ **Complex embedded systems are increasingly used to handle sensitive data or services**
- ➢ Designed for high-end security
- ➢ May not fulfill the IoT security's criteria
- ➢ Easy targets for Hackers
- ➢ Strong constraints in terms of power consumption and cost

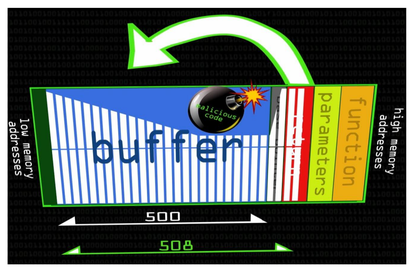
❑ **Necessary to design processor with security features which are easy to handle from a software point of view**
- ➢ No Dedicated compilation tools or coding style
- ➢ No modification on the software architecture
- ➢ Less complex as possible to reduce misuse
- ➢ reasonable associates cost in terms of area, power, design cost and also in terms of code size

❑ **We work with the stack pointer and the return address for both attack and defense as they are key components of the system which are accessible**

Privilege Escalation with Buffer Overflow

❑ **Bug obtain when a program tries to write outside his allocated buffer memory**
- ➢ Erase assembly instructions of the program, which become unstable
- ➢ Can be exploited to violate the security and gain privilege access [1,2]

❑ **Our Objective : Exploit this bug to get a shell with root access**
- ➢ We need to master the overflow to reach the return address
- ➢ Write a new return address which reach a function with malicious code



❑ **A C-code with 3 functions has been wrote**
- ➢ One of them, malicious_shell(), is never used but call "system(/bin/sh)"
- ➢ The goal is to replace the return address with the address of malicious_shell()

```
/ $ whoami
muscle
/ $ ./exploit.elf 13
Adresse de la fonction : 25e4
Valeur du pointeur 0x7f953e47
Valeur du pointeur 0x7f953e54
Valeur du pointeur 0

GOOT ROOOT ??!!


/ # whoami
root
/ #
```

- ➢ The function take as argument the offset needed

❑ **Results**
- ➢ A simple user got root access
- ➢ A software which look harmless generate a bug and exploit it

String Format Attack

❑ **Exploit base on the printf/scanf vulnerability**
- ➢ Putting a %x as a printf argument reveal the vulnerability
- ➢ %n is used by the printf() function to write into an argument the number of characters before %n

❑ **A Basic function calculate the 10 first terms of Fibonacci sequence**

❑ **Our objective is to re-write "10" to obtain a bigger loop**
- ➢ The first time we are fuzzing the function :

```
/ # ./string.elf
./string.elf
i = 10 = a is at address 0x48d8 and is the number of fibonacci loop

enter username for the database :

%x-%x-%x-%x-%x-%x
%x-%x-%x-%x-%x-%x
7fb41dbc-a-48d8-25782d25-782d2578-2d25782d
```

- ➢ Here the "a" is the "10" we are trying to overwrite
- ➢ We launch the function a second time with correct "username" to do so

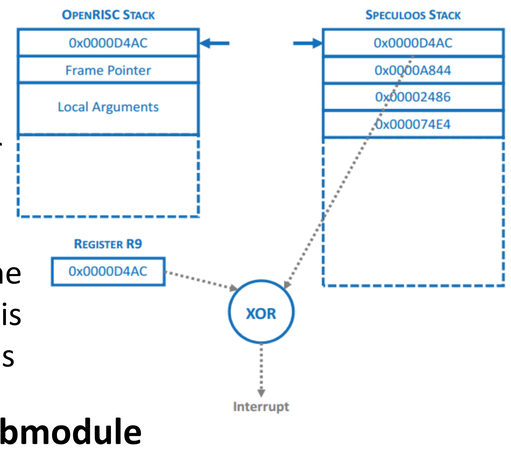
❑ **This vulnerability may seems harmless but can easily be used to escalate privileges and access sensible data**

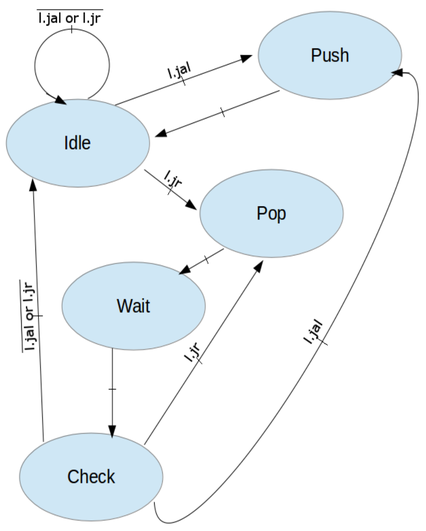
```
enter username for the database :

%x-%08x%n
%x-%08x%n
7ff93dbc-0000000a

calculation of fibonacci...

3
5
8
13
21
34
55
89
144
233
377
610
987
1597
2584
4181
6765
```

Defense : Finite State Machine & Stack

❑ **To secure, creation of a new stack, only for return address storage, and a finite state machine (FSM) to synchronize it with the OpenRISC architecture**
- ➢ When a function is called, the return address is push in the OpenRISC stack and in ours.
- ➢ When the function is terminated, the return address of its is loaded in the register R9
- ➢ The FSM apply a XOR between the R9 value and the address in the second stack
- ➢ Interrupt emits when they don't match

❑ **A new module : Speculoos**
- ➢ 3 Submodules : Stack, FSM and Observer
- ➢ The FSM manages all interactions with the stack and raise interrupt
- ➢ Observer manages the link between the OpenRISC architecture and the FSM, it is capable to detect jal, jalr or jr instructions





❑ **FSM submodule**
- ➢ 5 States : Push, Pop, Wait, Check and Idle
- ➢ Observer wake the FSM up by detecting jump in Idle state
- ➢ A variation of the module Speculoos without Observer has been wrote, Everything is checked within the Idle state

❑ **Observer submodule**
- ➢ Filters instructions to select only "Jump and Link" (jal & jalr) and "Jump Register" (jr)
- ➢ Detects when instruction flow jump to another location

❑ **This defense aim to detect and stop all attacks based on modifying the return address to execute malicious code**
- ➢ E.g. Buffer Overflow, String Format Attack, Return Oriented Programming

Results

❑ **All modules have been tested and validated with unitary tests**

❑ **Both Speculoos modules detect and interrupt the system while running an attack on simulation**

On going works

❑ **Make the defense working on the embedded system**
- ➢ Synchronize the program in execution with our defense
  - ➢ Work on the sequencing on Linux
  - ➢ Solution currently under adaptation to follow Linux specification

References

[1] M. Prandini and M. Ramilli, "Return-oriented programming," IEEE Security & Privacy, vol. 10, no. 6, pp. 84–87, Dec 2012.

[2] T. Bletsch, X. Jiang, V. W. Freeh, and Z. Liang, "Jumporiented programming: a new class of code-reuse attack," in ACM Computer and Communications Security, Oct 2011, pp. 30–40.

Esisar Team - CSAW ESC 2016