Code Injection Attacks on Harvard-Based Architecture Devices

Aurélien Francillon
Claude Castelluccia

INRIA
Outline

● Introduction
  ● Standard code injection
  ● Von Neumann and Harvard architectures

● Attack Description
  ● Assumptions and Building Blocks
  ● Building Blocks
    ● Return Oriented Programming
    ● Bootloader
    ● Fake stack injection
  ● Attack description
    ● Attack Overview
    ● Step by step attack description

● Conclusion
  ● Results
  ● Future work
Standard Code Injection Technique

- Standard Code injection technique
  - Overflow a buffer allocated on stack
  - Overwrite return address
  - Inject instructions on the stack
  - Return to those instructions

- Possible on Von Neumann architecture
  - Can be prevented by making Stack Memory non executable (NX, PAX...)
  - Not possible on Harvard Architectures

- But Return Oriented Programming still enables to perform some « actions » (more details to come)
  - It’s less powerful than code injection
Von Neumann Architecture

- Instructions, data form the same memory
- An attacker can inject code in the stack and execute it
- The most common CPU architecture
Harvard Architecture

- Instructions, data from separate memory
- An attacker can inject code in the stack but **cannot** execute it
- Common in embedded systems (AVR, PIC) and DSP
Harvard Architecture

- It has been a common belief that code injection into a Harvard architecture is impossible!!…

- We show that this is not only possible but that code injection can also be performed on Harvard-Architecture constrained devices

- We were able to inject code into a Micaz WSN node
  - Constrained memory 4KB data 128KBytes program memory
  - Small network packets (28Bytes) on a IEEE 802.15.4 radio
  - Harvard Architecture (ATMEL AVR Atmega128)
Outline

- Introduction
  - Standard code injection
  - Von Neumann and Harvard architectures

- Attack Description
  - Assumptions and Building Blocks
  - Building Blocks
    - Return Oriented Programming
    - Bootloader
    - Fake stack injection
  - Attack description
    - Attack Overview
    - Step by step attack description

- Conclusion
  - Results
  - Future work
Objective: perform remote code injection on a Micaz

Assumptions:
- A traditional stack based buffer overflow exists
- Program Memory contents are known
- A boot loader is present

Building blocks:
- Return Oriented Programming
  - To overcome the Harvard architecture
- Bootloader
- Fake stack injection
  - To overcome the packet and stack size limitation
Return Oriented Programming (ROP)

● Generalization of “return to libc” attacks
  ◆ introduced by H. Shacham at CCS 2007 on the x86

● Useful when Code cannot be executed in the stack..

● But the control flow can be modified
  ◆ Executes code already present in the program memory
  ◆ By chaining sequences of instructions terminated by a return
    • Called Gadgets
  ◆ Gadgets can be used to perform any action
    • When a Turing Complete Gadget set is available
ROP on Embedded Systems

- ROP is not practical on embedded systems
  - Packets and stack size limitations
  - Code size limitation (gadgets availability)
    - Finding a *Turing Complete* Gadget set is unlikely

- But is an useful tool to perform code injection
Bootloader: a key element of the attack

- **Bootloader** is the program used for code update
- **Code update** is a must... otherwise
  - A device with a bug is “dead”
  - Think about pacemakers, sensors in cars ...
- The bootloader consists in:
  - Getting the image (from serial port, local storage, network ...) 
  - Copying the image to program memory 
  - Uses dedicated instructions to copy a page from data to program memory
- We assume that if *remote* code update is present images are authenticated
  - *Otherwise attack is trivial*
Injecting code using ROP on sensors

- To inject code into a device, the attacker has to send a specially-crafted packet that contains:
  - Addresses of the gadgets
  - Gadgets’ parameters
  - The malware code to be injected
  - Padding
- I.e. 305 bytes if malware is 256 bytes long

But this is not possible on Sensors…

- Packet and stack sizes are limited,
  - TinyOS packet payload maximum size is 28 bytes!
  - The number of gadgets that can be linked is limited
Fake stack injection

- Instead of sending one large packet with the large stack in it we send several small specially-crafted packets
  - To build a fake stack

- Each packet:
  - Contains one byte of the fake stack
  - Triggers a buffer overflow
  - Copies its fake stack byte into unused data memory, using ROP

- A final packet
  - Triggers a buffer overflow
  - Changes the Stack Pointer to the fake stack, using ROP
  - Executes several gadgets that copy the malware (contained into the fake stack) into the program memory
Attack detailed overview

- Injecting code in persistent flash memory
Illustration

- A specially-crafted packet (that contains the first byte of the fake stack) is sent
Illustration (2)

- This packet triggers a buffer overflow that copies its fake stack byte into the RAM memory
- And then reboots the node to restore consistent state
Writing a byte into memory (simplified)

- A gadget chain

Program Memory

Vulnerable function
ret

pop r2
ret

pop r1
ret

st r2,@r1
ret

stack contents

Data

Buffer

Data Memory (SRAM)

Program Memory

PC

Vulnerable function
ret

pop r2
ret

pop r1
ret

st r2,@r1
ret

stack contents

Data

10

0x1000

R2=10
R1=0x1000
*R1=R2

Buffer

@gadget 1

10

@gadget2

0x1000

@gadget3

0

SP

Data Memory (SRAM)
Illustration (3)

○ A specially-crafted packet (that contains the second byte of the fake stack) is sent
Illustration (4)

- This packet triggers a buffer overflow that copies its fake stack byte into the RAM memory... and reboots the node.
Illustration (5)

- After N packets the fake stack is in memory!
Illustration (6)

- A final specially-crafted packet is sent…
Illustration (7)

- that triggers a buffer overflow and changes the SP to the fake stack
Illustration (8)

- The fake stack is then “executed” / interpreted
Illustration (9)

- The malware is copied from RAM to FLASH
Outline

- Introduction
  - Standard code injection
  - Von Neumann and Harvard architectures

- Attack Description
  - Assumptions and Building Blocks
  - Building Blocks
    - Return Oriented Programming
    - Bootloader
    - Fake stack injection
  - Attack description
    - Attack Overview
    - Step by step attack description

- Conclusion
  - Results
  - Future work
Main Results

- Harvard Architecture does not prevent code injection
- Return Oriented Programming can be used on constrained platforms
  - For a limited set of “actions”
- We showed that ROP can be used to inject code on AVR cpu
  - With small program size
  - Validated on 10 TinyOS applications
- The attack can be automated
  - We designed a tool that builds data injection payload automatically
- Worms and Viruses are realistic threats to Wireless sensor networks
Future work

Improving the attack

● Current attack uses gadgets on the Program memory
  ◆ They are sub-optimal
    • Can only copy 1 byte into data memory per packet

● We propose to optimize the attack:
  ◆ Injecting an optimized gadget
    • Using described attack
  ◆ Injecting malware using this optimized gadget
    • Can copy Several bytes into data memory per packet
    • reduced number of packets necessary,
    • less reboots,
    • Less risks of packet lost
Counter measures

Existing Counter measures:

◆ Safe TinyOS
  ◆ very promising but not a “silver bullet”
  ◆ Manual source code annotation
  ◆ Coding errors still possible

Possible Counter Measures:

◆ Binary Randomisation
  ◆ too Small address space ?

◆ Random stack canaries?
  ◆ Would make stack based buffer overflow more difficult

◆ Erase whole data memory between reboots
  ◆ Reboots are not required
Questions ?