SmashClean: A Hardware level mitigation to stack smashing attacks in OpenRISC

Manara Alam, Debapriya Basu Roy, Sarani Bhattacharya, Vidhya Govindan
Rajat Subhra Chakraborty and Debdeep Mukhopadhyay
alam.manaar@gmail.com, vidya.mazhur@gmail.com, sarani.bhattacharya@cs.iitkgp.ernet.in, deb.basu.roy@cs.iitkgp.ernet.in, rschakraborty@cs.iitkgp.ernet.in, debdeep@cs.iitkgp.ernet.in

Secure Embedded Architecture Lab, Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, India

INTRODUCTION

- Security threats to embedded systems
  - Hardware and Software vulnerabilities
  - Performance-efficient languages such as C and C++ widely used for embedded applications
  - Vulnerable to memory corruption due to lack of secure management
- Buffer Overflow: Trigger malicious code execution by overwriting correct memory content
  - Software level countermeasures may be easily bypassed
  - Need hardware level countermeasures, e.g., hardware-based protection of the function return address
  - Target platform for existing architectures different from the OpenRISC ISA processor

ATTACKING CONTROL FLOW

Return Address Modification

```
int func(char* user, int len) {  
  char buff[100];  
  memcpy(buff, user, len);  //Vulnerability
}
```

Format String Vulnerability

```
int n;  
printf("%10cks", 'A', 4n);
```

Example: Assembly Code for Stack.c

```
vuln:
1.lwz r4, -40(r2) # SI load
1.ori r5, r4, 0 # move reg to reg
1.lsw -40(r2), r3 # SI store
1.nop # nop delay slot
1.cfi_endproc
```

- If the address provided by a malicious user causes buffer overflow to modify r0, then the control flow gets transferred to the malicious code

Our Objective

- Hardware-based Mitigation of Memory Corruption and Ensuring Control Flow Integrity for the OpenRISC ISA Processor

Our Contributions

- Prevention of all forms of memory corruption and buffer overflow attacks on OpenRISC architecture
- Combination of compiler and hardware modification
- Introduction of new instructions via hardware modification for compiler to detect and prevent memory corruption via buffer overflow

Secure memcpy

```
void memcpy(char* buff, user, int len) {  
  *user = &olddata;
}
```

PREVENT MEM. CORRUPTION

- We introduced hardware enforced secure memcpy
  - This protection prevents buffer overflow by hardware induced bound check and prevents any memory corruption due to buffer overflow.

Example: Assembly Code for Priv.c

```
vuln:
1.sw -40(r2), r3 # SI store
...
1.sw -36(r2), r3 # SI store
...
1.nop # nop delay slot
1.lwz r4, -44(r2) # SI load
1.addi r3, r2, -32 # addi3
1.ori r5, r4, 0 # move reg to reg
1.lwz r4, -40(r2) # SI load
1.jalr # call_value_internal
1.nop # nop delay slot
```

- The first instruction (l.addi r3, r2, -32) transfers the starting address of the buffer (r2 – 32) to r3. The address of the latest new variable in this case is r2 – 16. Subtracting this two will give us buffer size which in this case is 16.
- The next instruction l.ori transfers the function argument count to r3, which denotes the number of memory locations to be updated in this instruction.
- Now, we will check whether the instruction l.ori r5, r4, 0 returns the count value greater than the buffer size or not.

NEW INSTRUCTIONS

- Least3 This instruction will be inserted by the compiler just before memcpy function is declared in C code to protect buffer overflow. This instruction sets a specific flag inside the processor and observes the occurrence of l.addi and l.ori which are required for computation of buffer size. If the buffer size is less than the argument count a smash_detect flag is set and the value of the count argument is updated with the buffer size. Thus this instruction ensures both detection and prevention of buffer overflow.
- Least4 This instruction resets the smash_detect flag.
- Least5 This instruction induces a lock on latest variable address location to preserve it from intermediate function calls. This can be alternatively achieved by maintaining a hardware stack for latest variable locations for each function call.
- Least6 This instruction removes the aforementioned lock.