Buffer Overflow Attack and Prevention for an FPGA Based Soft-Processor System

7th International Conference on Innovation in Electronics and Communication Engineering(ICIECE'18)

Chamandeep Singh[1] Sripadam Satish[2]
Jubin Mitra[3] Sandeep Shukla[4]

[1,2] NIT, Tiruchirappalli, Tamil Nadu, India

[3,4] IIT Kanpur, Uttar Pradesh, India

[1] *divergentchaman@gmail.com*,
[3] *jubinmitra@cse.iitk.ac.in*

July 28, 2017

Overview

Problem Statement

- **Motivation :** Embedded systems are exposed to various types of attacks, of those major vulnerability is the Buffer Overflow Attack on the processor used.
- **Attack Activity :** The buffer overflow attack corrupts the return address of a function or process and subsequently changes the execution order.
- **Vulnerable Hardware :** For rapid prototyping of designs, FPGA is the most preferred solution. Such designs often rely on a *soft-processor in the FPGA*.
- **Goal :**
  - Study the effect of buffer overflow attack in an embedded processor.
  - Demonstrate the attack on a full chain of embedded system.
  - Providing cost-effective mitigation solution to prevent this type of attack.

Popular Buffer Overflow Attacks

In the late 1980s, Robert T. Morris designed a worm using a buffer overflow vulnerability in UNIXs fingerd program. Nearly, 10% of the internet nearly came to a halt.

In 2013 a buffer overflow vulnerability in the OpenSSL cryptography library was disclosed to the public. This flaw came to be known as Heartbleed. Exposed hundreds of millions of users of popular online services.

One more example of this attack is the Apache HTTP servers htpasswd.c program, which manipulates password file for Apache HTTP server. The vulnerable part of the code takes user supplied name and copies it without sanity check to a fixed size local buffer using strcpy.

Benjamin Kunz-Mejri in 2017 discovered that skype software of Microsoft is also vulnerable to buffer overflow attack. The fault present in skype software was exposed to attacks by remote attackers without the authenticated users knowledge and was possible even with the basic skype account.

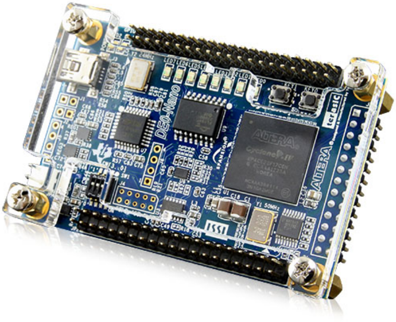
Attack Definition

What is Buffer Overflow Attack?

Buffer overflow attack overflows a buffer in function call activation record to manipulate the return address and hijack the control flow of the program function pointers to change the flow of the program.

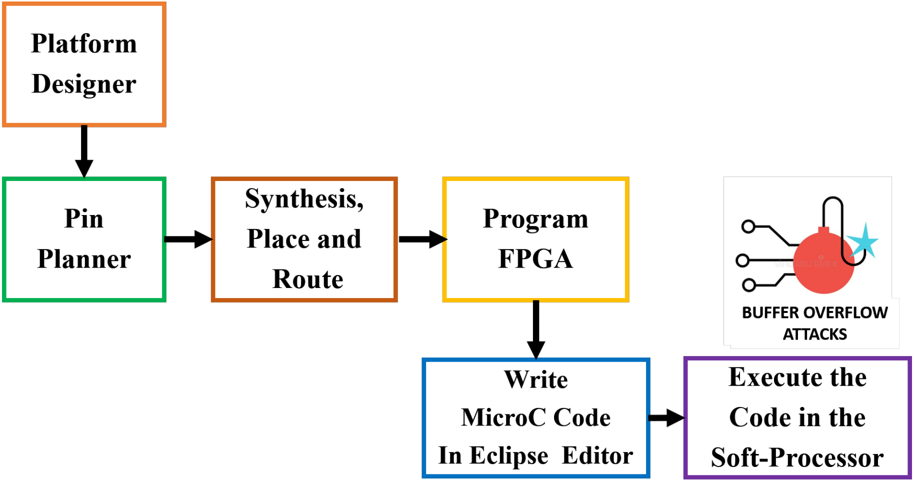
Stack Smashing Attack

Emulation Hardware



1. Intel Cyclone IV FPGA
2. 22,320 Logic elements (LEs)
3. 594 Embedded memory (Kbits)
4. 66 Embedded 18 x 18 multipliers
5. 32MB SDRAM

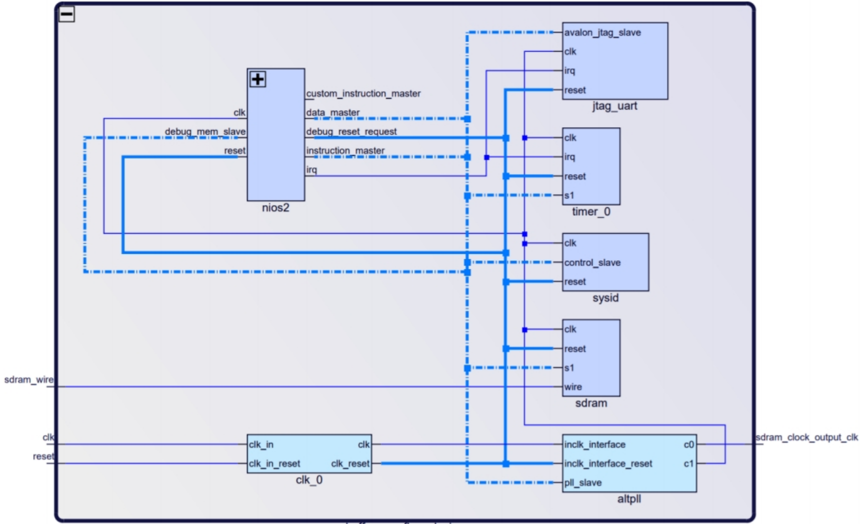
Design Flow



Platform Designer
Pin Planner
Synthesis, Place and Route
Program FPGA
BUFFER OVERFLOW ATTACKS
Write MicroC Code In Eclipse Editor
Execute the Code in the Soft-Processor

Platform Designer

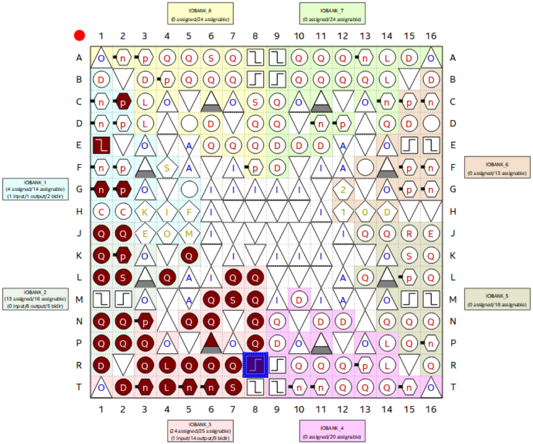
Nios II is the **soft-processor** provided by Intel. It is a RISC based processor.



Pin Planner

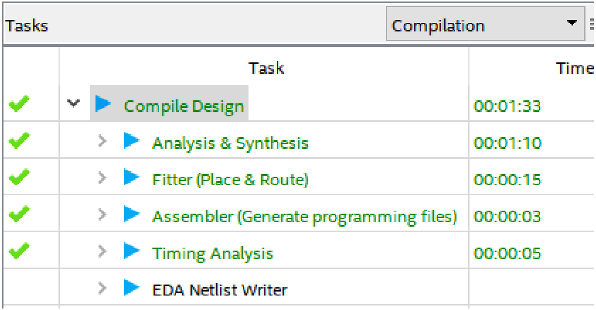
Top View - Wire Bond

Cyclone IV E - EP4CE22F17C6



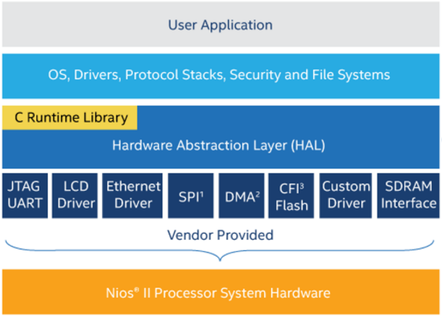
Assigning the pin connections on the FPGA board.

Synthesis, Place & Route, Bit File Generation



Hardware Abstraction Layer (HAL)

Showing the development stack where MicroC code is written after soft processor implementation



Examples of HAL API functions	
_exit()	Open()
Close()	Opendir()
Closedir()	Read()
Fstat()	Readdir()
Getpid()	Rewinddir()
Getimeofday()	Sbrk()
Ioctl()	Settimeofday()
Isatty()	Stat()
Kill()	Usleep()
Iseek()	Wait()

MicroC code

Function "task2" is never called in the original code flow

```
#include <stdio.h>
#include "includes.h"
#include "string.h"

/*Definition of Task Stacks*/
#define TSK_STKSZ 2048
OS_STK task1_stk[TSK_STKSZ];
OS_STK task2_stk[TSK_STKSZ];

/* TASK PRIORITY */
#define TASK1_PRIORITY      1
#define TASK2_PRIORITY      2

/* data passed is copied into local
     buffer */
void user_input(char *data) {
 char name[4];
 strcpy(name, data) ;
 return;

}

/* TASK1 */
void task1(void* pdata)
{
user_input("\x3a\xf0\x4a\x29\xe8\
    x02\x00\x02");
  while (1)
  {
    printf("Credential Mismatched\n
    ");

    OSTimeDlyHMSM(0, 0, 3, 0);
  }
}
void task2(void* pdata)
{
  while (1)
  {
    printf("Credential Matched\n");

    OSTimeDlyHMSM(0, 0, 3, 0);
  }
}

/* The main function creates
   two task and starts
   multi-tasking */
int main(void)
{

printf("MicroC/OS-II based Buffer
    Overflow\n");

  OSTaskCreateExt(
  task1,
  NULL,
  (void *)&task1_stk[TSK_STKSZ-1],
  TASK1_PRIORITY,
  TASK1_PRIORITY,
  task1_stk,
  TASK_STACKSIZE,
  NULL,
  0);

  OSStart();
  return 0;
}
```

Listing 1.1. An example of a standard

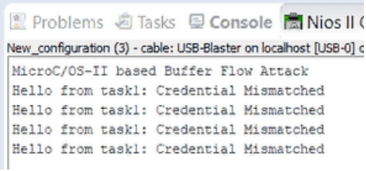
NOP Sled Technique to do Buffer Overflow

- Attacker passes NOPs to evaluate the length of string the program malfunctions.
- Nios II implements NOP as ‘add r0, r0, r0’, as r0 is a constant register with a value of zero. But ‘add r0, r0, r0’ when encoded will have null bytes in it. Hence in order to avoid this problem, the NOP is implemented as ‘xor r5, r5, r5’.
- As the Nios II architecture is little endian, the equivalent input string for the ‘xor r5, r5, r5’ instruction is `\x3a\xf0\x4a\x29`.

Showing the effect of NOP Sled Buffer Overflow Attack

With 1 NOP.

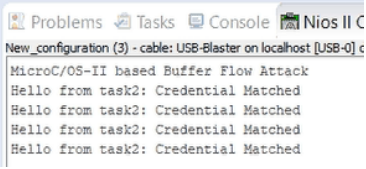
\x3a\xf0\x4a\x29



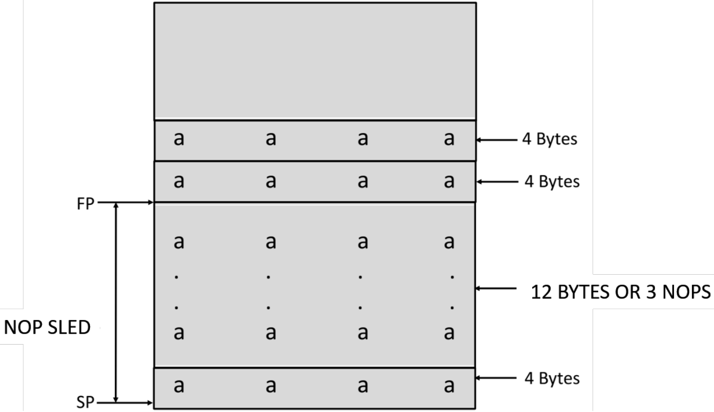
The string for which the code malfunctions

\x3a\xf0\x4a\x29
\x3a\xf0\x4a\x29
\x3a\xf0\x4a\x29
\x08\x02\x00\x02

Last line is **address of func. task2**



How the NOP Sled Buffer Overflow Attack is possible



Mitigation Method

C is vulnerable to this attack due to **direct access to memory** and **lack of strong object typing**.

The string handling functions like **strcpy** and **strcat** copy a string into a buffer and then append the contents of one buffer upon another, respectively. These two are unsafe because they dont check any bounds on the target buffer, and **would write past the buffers limits** if given enough bytes to do so.

So it is suggested to use their associated **strn-** versions. These versions only write to the maximum size of the target buffer.

Conclusion

The platform based defenses against buffer overflow is missing in the platform, and hence if the user writes such code, he will be exposing himself to vulnerabilities.

Future Plan

Hardware based mitigation solutions as they are most effective and realiable.

Acknowledgment

The research work is funded by the DST for C3I center at Department of Computer Science, IIT Kanpur.

Thank You