Buffer Overflow Attacks

Tom Chothia
Computer Security, Lecture 15

Introduction

- A simplified, high-level view of buffer overflow attacks.
  – x86 architecture
  – overflows on the stack
- Exploiting buffer overflows using Metasploit

Introduction

- In languages like C, you have to tell the compiler how to manage the memory.
  – This is hard.
- If you get it wrong, then an attacker can usually exploit this bug to make your application run ***arbitrary code.***
- Countless worms, attacks against SQL servers, Web Servers, iPhone Jailbreak, SSH servers, …

USS Yorktown

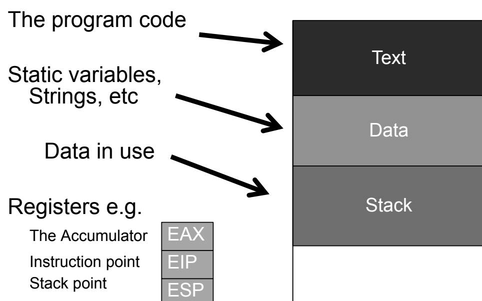
US Navy Aegis missile cruiser

Dead in the water for 2 and a half hours due to a buffer overflow.

"Because of politics, some things are being forced on us that without political pressure we might not do, like Windows NT. If it were up to me I probably would not have used Windows NT in this particular application."

Ron Redman, deputy technical director Aegis

The x86 Architecture



The Stack

The stack part of the memory is mostly "Last In, First Out".

We can only write and read to the top of the stack.

…
PUSH 12345
PUSH 678245
POP EAX
….

Data

Stack

EAX:
EIP: 7797F9CD
ESP: 0018F9B0

The Stack

You write to the stack with push

You read and remove an item from the stack with pop

…
PUSH 12345
PUSH 678245
POP EAX
….

Data

Stack

123456

EAX: 678245
EIP: 7797F9CF
ESP: 0018F9B1

Function calls

```
void main () {
  function (1,2);
}
```

```
PUSH <2>
PUSH <1>
CALL <function>
```

- Arguments 1 & 2 are passed on the stack.
- The CALL instruction runs a function

Function Calls

```
PUSH <arg2>
PUSH <arg1>
CALL <function>
```

`CALL` writes the instruction point (EIP) onto the stack and then sets the EIP to to equal the code for the function.

Later a return instruction restores the old EIP and the program continues

	Old EIP	Arg1	Arg2	Stack

Buffer Overflows

- The instruction pointer controls which code executes,
- The instruction pointer is stored on the stack,
- I can write to the stack … ☺

Buffers

1. Functions called with "Hello World"
2. Arg and EIP written to stack
3. Function runs
4. Buffer allocated
5. String copied

```
…
function (user input);
…

function (char *str) {
  char buffer[16];
  strcpy(str,buffer);
}
```

	Hello World	Old EIP	Hello World	Stack

Buffer Overflow

If user input is more than 16 bytes

1. Runs as before
2. But the string flows over the end of the buffer
3. EIP corrupted, segmentation fault

```
…
function (user input);
…

function (char *str) {
  char buffer[16];
  strcpy(str,buffer);
}
```

	Hello WorldXX	XXXXIP	Hello WorldX XXXXXXXXX	Stack

Once more, with malice

1. Runs as before
2. Attack send a very long message, ending with the address of some code that gives him a shell.
   – The attackers code could also be part of the message
3. The attackers value is copied over the old EIP
4. When the function returns the attacks code is run

	Hello WorldXX	7797F9	Hello WorldX X7797F9	Stack

Metasploit

- Metasploit is a framework for testing buffer overflow attacks.
- http://metasploit.com/

Over Writing Other Values

Attacking the instruction pointer (EIP) is the most powerful technique. However, any memory value can be attacked:

- Over write arguments on the stack
  – e.g. change the parameters to a chmod call
- Overflows on the heap
  – e.g. rewrite a password in memory

Defenses

- Stack canaries:
  – values placed on the stack, which are later tested.
  – if the stack is over written then the value test will fail.
- Randomisation
  – Layout of the memory is randomised.
  – This makes it very hard for the attack to find the memory to overwrite or code to jump to.

For more information see the Secure Programming Module

Recommend Paper:

- "Smashing the Stack for Fun and Profit"
  Elias Levy (Aleph One)

A simple introduction to buffer overflows from the mid 90s.

Standard defences now stop the attacks in this paper, but it gives an excellent introduction.

Conclusion

Buffer overflows are the result of poor memory management in languages like C

- even the best programmers sometimes make mistakes.

Buffer overflow attacks exploit these to over write memory values.

This often lets an attack execute arbitrary code.