CSE 509 Lecture 5

Prof. Rob Johnson, Scribe:Arun Rathakrishnan

September 17, 2013

1 Review - Buffer Overflow

1.1 Code Injection Attack

Causing a program to violate trust, by providing input which causes a buffer to overflow, thereby forcing the machine to execute untrusted code.

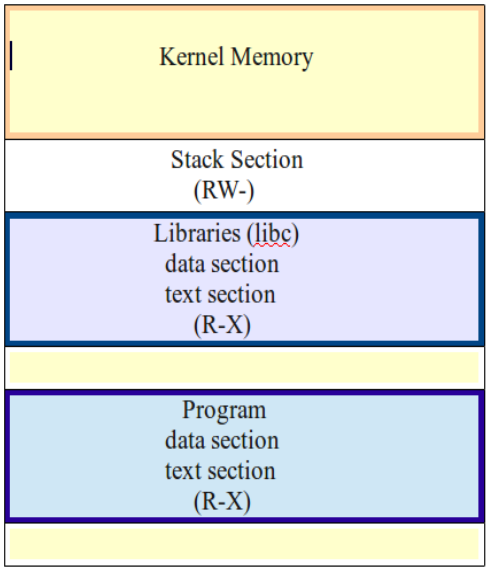
1.2 Set up

- Consider a machine in which the activation records (arguments local variables and return address for a function which is being called) are stored on a stack. The machine executes code in a page which contains the address pointed to by the Instruction Pointer (IP). IP can also point to a byte in stack segment.
- A local variable allocated in stack, can point to a buffer allocated in stack. When the user input is used to fill up contents of the buffer, the input can overwrite parts of stack that contain the return address and arguments from the caller and beyond.
- By providing a malicious input overwriting the return address of the current function to the beginning of the malicious program on the stack, an attacker can force the machine to jump after the execution of the vulnerable function, to the beginning of a malicious program in stack and start executing it.

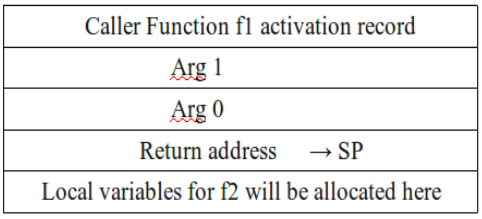
1.3 How to prevent?

Use NX (Non-executable) bit to mark pages that are not executable. Only pages in the text section of a process are executable. Stack section has NX bit set. So even if control is transferred to the stack by changing the IP, the system will be forced to page fault and terminate the process.

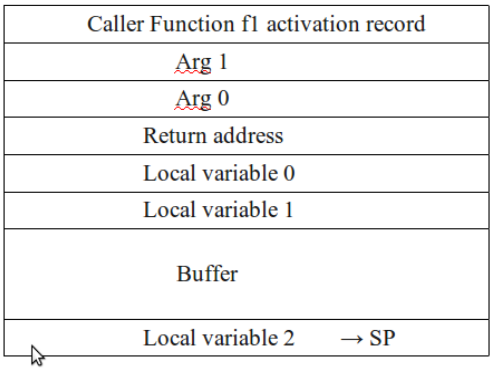
1.4 VM Layout



1.5 Activation Records in Stack



At the beginning of f1 calling f2.



The callee function allocates local variables

2 Return to libc Attack

Cause buffer overflow so that the new return address of the vulnerable function is changed to a libc function (usually system) with arguments overwritten in such a way so as to gain unauthorized access to the system.

For example, one can provide a shell command and overwrite the return address on input to a webserver running as a root process to begin a root shell. In this way, an attacker can make use of the machine code in text section to attack a machine irrespective of NX bits.

2.1 Attack

Caller function f1 activation record	Caller function f1 activation record
	End of Input
	cmd: /bin/sh
Arg 1	Pointer to cmd (arg 0 for system)
Arg 0	Return address for **system**
Return address	Address of **system**
Local variable 0	Garbage
Local variable 1	
Buffer	Start of user input
Local variable 2 → SP	Local variable 2 → SP

(Left) Expected Stack. (Right) Stack after buffer overflow

2.2 How to find address of system function?

Libraries are loaded only at page boundaries. Within a library the relative position of a function is the same in the address space. By searching a limited number of such page boundaries and finding the correct offset, one can locate the system function' address.

2.3 How to prevent?

- Not loading system function to libc if the text section of a process does not call this.
- Address Space Randomization.

3 Return to libc Attack in Practical Machines

Some real machines, expect the arguments to be placed in special registers (like RDA, RDB, etc) and not in stack. In such machines attackers are forced to place the arguments in RDA by exploiting gadgets, such that by the time, control goes to the overwritten address, RDA has the desired arguments.

3.1 Gadget

A gadget is a basic block of machine code, usually without a branch till it returns. For our example attack, we consider the below gadget code, which pops the stack pointer into the RDA register. When gadget is executed, no activation record is pushed to stack and the stack is usually not changed. Only the IP is changed.

```
pop %rda
ret
```

Using the gadgets, the trick is to overwrite the return address with the address of the above gadget, followed by the argument's address (so that the gadget copies the argument pointed by SP to RDA). This is followed by the overwriting the return address after gadget's execution to system function's address as usual.

3.2 Attack

<table>
<tr><td>Caller function f1 activation record</td><td>Caller function f1 activation record</td></tr>
<tr><td></td><td></td></tr>
<tr><td></td><td>End of Input</td></tr>
<tr><td></td><td>cmd: /bin/sh</td></tr>
<tr><td></td><td></td></tr>
<tr><td>Arg 1</td><td>Address for system (used after gadget rets)</td></tr>
<tr><td>Arg 0</td><td>Pointer to cmd (arg 0 for system)</td></tr>
<tr><td>Return address</td><td>Address of gadget</td></tr>
<tr><td>Local variable 0</td><td rowspan="3">Garbage

Start of user input</td></tr>
<tr><td>Local variable 1</td></tr>
<tr><td>Buffer</td></tr>
<tr><td>Local variable 2 → SP</td><td>Local variable 2 → SP</td></tr>
</table>

(Left) Expected Stack. (Right) Stack after buffer overflow

4 Return Oriented Programming

Return oriented programming enables us to perform arbitrary computations even in machines with NX bit. It involves,

- Loading registers.
- Computing addresses (by changing the return address to middle of an instruction word)
- Inserting code off the stack.
- Making use of gadgets (exploit the fact that gadgets have fewer instructions before ret and x86 machines have variable length instructions).

To achieve our aim of loading RDA register with the desired argument, we find a suitable gadget which has the instruction for loading RDA as a substring in its code. Consider the following gadget snippet.

```
0x400006:f7c7000000 test 0x007 %edi
0x40000B:0f9545c3   set setnzb -61 (%edi)
```

Whenever the gadget is invoked the jump is to $0x400006$. But we can invoke the gadget by following the same trick to change the return address of a vulnerable function. But instead jump to $0x400007$, causing the machine to treat the second byte of the gadget as the first one thus wrongly interpreting the instructions.

```
0x400007:c70000000f movi
0x40000B:95         xchg RDA, SP
0x40000C:45         inc
0x40000C:c3         ret
```

We can look through the gadgets for instructions to copy to RDA (95 in this example) and ret ($c3$) and change the jump address to choose a suitable gadget to exploit for the attack. That gadgets are short and thus assist attackers to execute the attack code and return quickly.