P1L2 Software Security

Software Vulnerabilities and How They Get Exploited

Example: Buffer Overflows

Stack buffer overflows - inserting extra instructions into a command to force an overflow that inserts calls to malware

stacks are used:
- in function/procedure calls
- for allocation of memory for:
  - -local variables
  - -parameters
  - -control information (return address)

A legitimate call is manipulated by an attacker to gain unauthorized access to data

Vulnerable Checking Program

Password checking passwords do not keep copy of your password.

A vulnerable password checking program allows attackers to overflow a buffer by allowing the user to insert a longer password, which bypasses the check procedure.

Understanding the Stack

The stack shrinks and grows with the pushing and popping of data on and off the stack.

If the user provides a password that meets one of the criteria listed the password checking program can be defeated:
- -Any password of length greater than 12 bytes that ends in ‘123’
- -Any password of length greater than 16 bytes that begins with ‘MyPwd123’

Attacker Code Execution

If the attacker guesses the password - login is allowed
If the attacker puts in the wrong password - login is not allowed

When the function call is made, the system pushes the address onto the stack.
Must also push the return address onto the stack so the program knows where to return.

12 bytes are allowed for the password and the target password.

If the password is less than 12 bytes, it will remain in allocated stack space. IF it is longer than 12 bytes it will overwrite the allow login space. If it is more than 12 bytes it will overwrite the return address space.

The attacker can than allow login and a different return address. An address can be inserted that points to malware. The malware will be executed, with the legitimate return address. The user will be unaware of any this.

What was the vulnerability that allowed this to happen?

The code did not check the input and reject password strings longer than 12 bytes.

ShellCode

The code the attacker wants to launch is called shellcode.

Shellcode:

1. creates a shell - it is short. The shellcode must be in machine code so that can be inserted directly into memory.
2. It must have a return address that is the legitimate return address.

When control is transferred to the shellcode whose privileges are used? Recall it is running on behalf of the user.
The privileges that are used are:

-the host program's (this is the program that is exploited by the shellcode)

-system service or OS root privileges - if the program is a system service the shellcode (and therefore the attacker) will have access to much of the system.

Variations of Buffer Overflow

Return-to-libc: the return address is overwritten to point to a function in a library. The function can then be executed with parameters of the attacker's choice. For example the attacker can launch a command shell.

Heap Overflows: long lived data gets stored on the heap (like global variables). The heap does not store return addresses. Data can be tables of function pointers. So the attacker can modify a function pointer to point to malware. This is more sophisticated than buffer overflow.

OpenSSL Heartbleed Vulnerability: the attacker can also read data. Suppose the attacker asks for more data than usual. This may expose sensitive data. An example of this is the vulnerability in the OpenSSL code.

Defense Against Buffer Overflow Attacks

Programming language choice is crucial to prevent buffer overflows.
Languages that have the following characteristics can prevent buffer overflows:

-should be strongly typed
-should do automatic bounds checks
-should do automatic memory management

Examples of safe languages: Java and C++
Low level languages are not safe.

Safe Languages

Drawback of saft languages -- reduction of performance and flexibility

Unsafe Languages

Sometimes unsafe languages must be used… what is the defense against buffer overflow attacks?

1. Check all input -- assume all input is evil
2. Use safer functions that do bounds checking. For example - checks the length of a string to prevent buffer overflow.
3. Use automatic tools to analyze code for unsafe functions. The tools use known patterns to compare the software to flag potentially vulnerable areas. There can be a number of false negatives with these tools.

Strongly vs Weakly Typed Languages

Strong languages:

-Any attempt to pass incompatible data is caught at compile time or generates an error at runtime

-It is impossible to do “pointer arithmetic” to access arbitrary areas of memory.

Weak languages:

-An array index operation by be allowed even though k is outside the range of the array

Analysis Tools

Check the source code by running it through the tools. If all you have is binary file, the tools may not always be helpful

Thwarting Buffer Overflow Attacks

Stack canaries: a canary values is written into the stack frame, just before the return address. IF the canary value is modified, then there is a good possibility the return address has been changed. Thus an overflow is detected.

Address Space Layout Randomization (ASLR): this method randomizes the stack, heap, libc. So an attacker will have a harder time finding the return address.

Non-executable Stack: use this with ASLR. This requires hardware support.

In these cases - the hacker's job becomes harder, rather than trying to plug every vulnerability.

Stack canaries prevent return-to-libc attacks
ASLR does not prevent read-only buffer overflow attacks
OpenSSL Heartbleed vul. cannot be avoided with a non-executable stack.