Buffer Overflows

A brief Introduction to the detection and prevention of buffer overflows for intermediate programmers.

By: Brian Roberts
What is a buffer overflow?

• In languages that deal with data structures such as arrays, there is the possibility of buffer overflows occurring due to poor array bounds checking.
• C and C++ do not perform bounds checking on arrays.
• Strcpy, gets and several other C functions do not protect against buffer overflows.
What is a Buffer?

1. Aleph One
Buffer Overflows

• When data that is located on the buffer is vulnerable to overwriting (such as is the case when memory outside the bounds of an array is accessed) it becomes possible to write data to the stack.

• This data can be crafted in such a way to result in the machine executing malicious code at the permission level of the program running.
void function(char *str) {
    char buffer[16];
    strcpy(buffer,str);
}

void main() {
    char large_string[256];
    int i;

    for( i = 0; i < 255; i++)
        large_string[i] = 'A';

    function(large_string);
}

2. Aleph One
Stack Smashing

• In the event that the program is running setuid root, the attacker can gain root access to a machine with only one exploit.

• In order for this to occur, the attacker must be able to inject malicious code to the stack, and also reroute the flow of control by rewriting the return address on the stack, or some other similar method.
How common are buffer overflows?

• Buffer overflows are VERY common, and many beginning programmers make it very easy for attackers by adopting sloppy programming practices and failing to use procedures to insure that code is properly secure.

• Ever used gets()? Ever used strcpy()? You might just have made yourself vulnerable.
How does a stack smashing buffer overflow work?

• The attacker who is performing the exploit will seek to overwrite some buffer in an suid root program, insert malicious code into the buffer (for example, a command to spawn a root shell) and then reroute control of the program to execute that malicious code.
How does a stack smashing buffer overflow work?

• The attacker does not necessarily need to find an exploit against a setuid program in order to gain access to elevated privileges so any program you are writing that is vulnerable could possibly be used as a method for an attacker to gain unauthorized access.
Difficulties with stack smashing.

- Architectures are different, and not all of them store return addresses in the stack. This means that exploits that work on one architecture won’t necessarily work on another architecture.
- Many programs halt execution when certain values are reached (for example NULL) this can result in the attacker having to write self-modifying code to avoid pushing a NULL onto the stack where it can be interpreted and result in the program crashing. This self modifying code can be difficult to write.
Difficulties continued.

• The attacker doesn’t know where the return address is going to be located on the stack. However, by performing a series of noop commands in assembly language, the attacker can work their way through a series of memory addresses until the return address is reached.
Defenses?

• There are several defenses that can be employed to prevent buffer overflow attacks.
  – First off, validate ALL I/O and perform bounds checking on ALL of your arrays.
  – Run programs with the least privilege necessary for them to perform their jobs.
  – Use libraries like libsafe that re-implement C functions that contain vulnerabilities.
Defenses Continued.

• Use libraries such as libverify that provide canary values for return addresses.
  – A canary value is a value that is placed on the stack. Occasionally this value is checked, and if it has been altered, it is a good indication of tampering. When this happens, the program is halted and crashes before further damage can be done.

• When you are in doubt as to whether an attack is occurring, crash the program. It is always better to terminate a program as soon as it exhibits dangerous behavior.
Defenses Continued.

- Use languages that provide array bounds checking such as Java
- Use compiler patches that provide bounds checking functionality.
- Have a code walk-through to check for security problems.
- Use static source code analysis tools to check for the existence of vulnerabilities.
References

  emacs_groks@yahoo.com

• Aleph One. “Smashing the stack for fun and profit.” Phrack 49
  <http://pintday.org/whitepapers/other/p49-14.shtml>