Buffer Overflow

What it is?

A buffer overflow occurs when a program or process tries to store more data in a buffer (temporary data storage area) than it was intended to hold. Since buffers are created to contain a finite amount of data, the extra information - which has to go somewhere - can overflow into adjacent buffers, corrupting or overwriting the valid data held in them. - http://searchsecurity.techtarget.com/definition/buffer-overflow

More

The attacker will need to be able to identify a buffer overflow vulnerability in some program and understand how that buffer will be stored in the process memory, and know what other adjacent memory items he could write to.

- Could DOS the system

To do this, could use a technique called fuzzing.

Fuzzing

Fuzzing involves sending malformed strings into application (and web) input and watching for unexpected crashes. There are lots of interesting tutorials as to how to do this. Many times after finding this malformed input, you would then use assembly to figure out how to deliver shellcode (or how to deliver the exploit).

Fuzzing does have practical usage in software development, but it is also a tool used by hackers to find vulnerabilities in applications.

Many fuzzing tools: ComRaider (activeX), see fuzztools on security distro.

Why are programs vulnerable to buffer overflows?

- poor programming
- computer doesn’t care... up to programmer to make sure memory is contained
- low level languages

How to protect against buffer overflow

- compile time defenses
- choose a higher level language (downsides?)
Examples

- Vulnerability Database
- This is interesting
- bufbomb example = make program do something it wasn’t originally designed to do
- Exim example = access to shellcode

Secure Programming

- SDLC (esp maintenance and testing)
- Least privilege principle
- Never trust user input (input validation)
- Reduce attack surface area (remove unused functions)
- Fail securely (try/catch)
- Patch and fix issues correctly

Software security

Software security is closely related to software quality and reliability.

Defensive programming: intended to ensure the continuing function of a piece of software in spite of unforeseeable usage of said software. Sometimes this is called “Secure Programming”. Never make any assumptions about code.

What to do?

Most programmers focus on solving a problem, (the algorithm involved) rather than considering every point of failure. Often make assumptions about the input and the environment it executes in. Defensive programmers should:

- validate all input
- handle potential failures

Software security has to be implemented in design phase of the development rather than as an afterthought.

Handling program input

One of the most common failures in software security.

The input is not always explicitly known. Making assumptions about the input can be bad. You should always handle the input cautiously to make sure that it isn’t invalid.

Two key areas of concern for input are:

- size of input
- meaning and interpretation of input

Don’t make any assumptions, validate everything.

Inputs
Injection attack: Essentially this is where an attacker provides non-expected input in order to do something that they shouldn’t. It influences the control or execution of a program. Commonly seen in PHP or other wescripts.

**Expected input = expected results, but what if the attacker injects a command, i.e.** 
```
  xxx; echo success; ls -l
  finger *
```