Module 30
(Unix/Linux Security Issues II)

• At the end of this module, you'll know more about Unix/Linux security issues. You'll understand more exploits at a little more fundamental level. You'll also get more ideas about how to avoid writing malware-ready code.
Modern Stack Smashing Attacks

- Although proponents of the technique thought have Non-executable stack segments would stop buffer overflows, they were wrong.
- Return-Oriented Programing (ROP) writes arguments and returns to code (gadgets) that appear in libraries just before a return to get some work done. It's a tricky method to get code executed by adjusting the stack but without writing the code in the stack.
- Every concept people have had about inability to overcome technological barriers to exploiting buffer overflows has ended up being wrong. I attribute this to the fact that both data (locals and parameters) and code (return addresses) appear in the stack.
- Until we have machines that more closely mimic a Harvard architecture, its best not to write code that lets buffers overflow.
Format String Attacks

- Format string vulnerabilities, like buffer overflows, result from a program not correctly handling user data.
- These usually appear in C programs that use printf or sprintf.
- User input is inserted into a format string without modification and inserts unexpected format characters (%s, %n, etc.)
- In extreme cases, this can allow arbitrary code to be executed.
Avoiding Format String Errors

• “An ounce of prevention is worth a pound of cure.”

• Although numerous static code checking software has been written to identify format error problems, the best method is to avoid using printf and sprintf entirely.

• If you must write in C, write in the better C dialect of C++ and use cout rather than printf.

• Far fewer errors will result.
Input Validation in General

- *Hacking Exposed 7* cites the case of the Solaris 2007 telnet 0-day that allowed anyone to log in as root. Issuing
  \[
  \texttt{telnet -l "-f user" hostname}
  \]
  would cause login (run as root) to receive the command line switch setting `-f user` which would allow the program to run as that user without providing a password.

- Two ways to avoid input validation problems:
  - Black Listing: disallow bad input. (Not so good.) “Porter's a good boy. He minds well enough. I just can't think of enough things to tell him not to do.” – Ferrol Sams, *Run with the Horsemen*.
  - White Listing: allow only good input. (Better when implemented correctly.) To implement this correctly, the parser for whitelisting must be as powerful as the input language.
Arithmetic Error Attacks

- All data in hardware represented as binary strings.
- In signed data (usually twos complement) the most significant bit (MSB) is interpreted as the sign bit.
- Typical hardware does not know how large an operand is or whether it is signed or not. The code generated by compilers and interpreters must keep track of that.
- C conversion/coercion rules are notoriously difficult to understand fully, leading to many subtle errors.
- Highly exploitable integer errors are ones that can lead to buffer overflows. These usually involve loop control variables, or variables that keep track of the length of data structures.
Dangling Reference (Pointer) Attacks

- Dangling references occur when heap or stack allocated storage is freed, but pointers to the data are not set to NULL.
- One way problems occur is when the same storage is freed multiple times.
- It might seem surprising that this could cause problems, but a vulnerability in CVS (concurrent versioning system) used this method to allow a remote attacker to cause corruption of heap storage and execute arbitrary code.
- Managing storage manually is hard!
- Using a system with automated garbage collection is the best option for avoiding these kinds of problems. GC separates the concerns of dealing with storage from dealing with your program.