Scribe Notes 11/28/2017

Stack overflow buffer

overflow buffer, overwrite key control
attacker can control how program is executed
overwrite return address, program returns to wrong location
crash likely
return address point to evil code (injection attack)

Attacker must know
- Address of evil code
- Location of return address on stack

Solution
- precede evil code with NOP "landing pad"
- insert return address many times via trial and error to launch attack, attacker needs to know that buffer overflow exists.

** Not all buffer overflows are exploitable, things must match up perfectly **

Example:
program asks for serial number that attacker does not know, and src code is not known by attacker, nor the executable file.

- Enter wrong serial number, then program does not execute.
- Attacker enters very long string for serial number to see if overflow is present.
- See the last 2 bytes cause overflow to occur.
- Goal, exploit buffer overflow so that it jumps to address 0x401034 where serial number is deemed valid.
- Notice that 0x401034 is hex for @^P4
- Try long serial number again with last 4 chars as @^P4, however it doesn't work because ** Byte order is reversed (little endian) **
- Should be 4^P@, attacker did not need src code.
- The tool used was a disassembler.

Attacker does not need access to the src code, only thin binary.
If src code was present then it would have made things easier, the serial number would have been in plain text.
Attackers generally try to gain shell control with overflow attacks.

Buffer overflow defense
- Design system so no buffer overflow occurs. Use modern high level language like Java.
- avoid potentially unsafe languages such as C.
- There are safety extensions in C, but at the cost of performance, and need a special compiler

Stack protection
- Add function entry and exit code to check for signs of corruption
- Save / check safe copy of ret address

Use Random canary
- Canary in memory location between return address and buffer
- if overflow occurs, then canary will be overwritten
- canary can be a constant or depend on the return address
- Microsoft has a canary called a security cookie that when changed checks for the handler

Address Space Randomization
- don’t know where program will be loaded, difficult for attacker.
- randomize location of heap buffer, standard library functions.

Address Space Layout Randomization (ASLR)
** No known attack on ASLR **

Guard Pages
- can place between stack frames and heap buffers
**We must be constantly vigilant of buffer overflows**

** Buffer Overflow attacks are the most important type of software vulnerabilities **

** Follow Good coding practices **

Chapter 22 Internet Security Protocols and Standards
S/MIME = secure / multipurpose Internet Mail Extensions
** cannot be used to secure real time events such as online transactions **
Content of mail protected from eavesdropping, integrity, and where it originated from.
** when you access email via gmail you're using SSL not S/MIME **
System admins would still have access to view emails when using SSL but not with S/MIME

CLASS QUESTIONS

What big hacker attacks that have gained a lot of attention on national media is a stack buffer overflow attack? Just curious about examples
- Too many to mention, most security attacks are buffer overflow

how does c++ compare with java and c in terms of safety?
- C, C++ are similar in safety, C++ has protections that C does not. Java is safer, but it comes down to the compiler, you could design a C++ compiler to be safer.

what is a buffer underflow?
When a buffer is being written to at a lower speed than it is read from.

why don't compilers do boundary check to avoid buffer overflow at compile time? Java do this at run time and slows everything down. Why can't it be done only at the compile time?
- Because modern computing system loading we don't know address in memory the program will be assigned.

how do you protect against stack overflows besides making the buffer larger?
- Canary or Address Space Randomization

Is there any difference in the effectiveness of a real-time protocol as opposed to a protocol that is not real-time?

- Both are important in stopping buffer overflows. Neither is considerably more effective than the other.

Is checking if input is too large the only way to defend against stack overflow attacks?

- No, its one of the ways another is good coding practices.

I have heard of using a stack canary to help detect if the return address has been overwritten during a buffer overflow attack. Suppose I was trying to perform this attack. Is there a way to ensure the stack canary remains unchanged while still overwriting the return address?

- No, Developer knows when canary is overwritten.