Chritopher LaJon Morgan
Nov 8, 2013

Homework 11

CS 465

Buffer Overflow Defenses; Answer the following:

▪ **List at least 5 defenses against buffer overflow attacks and provide a sentence or two describing what they are or how they work.**

1. Write Correct Code
   The issue exploited by a buffer overflow attack is simply one that exists because a programmer didn't enforce his assumptions. In essence it is a programmer's faulty or irresponsible code that creates the overflow issue. If the programmer simply wrote code that checked and sterilized all input to insure that it matched expectations this attack would be useless. So the write correct code defense is to write code that enforces expectations onto its input and that checks bounds, etc.

2. Non-executable Buffers
   One possible aspect of a buffer overflow attack can be to write code onto the stack, jump to it, and then execute it. This allows the attacker to inject code directly into the system and then execute it. However, the non-executable stack defense prevents the execute part of this attack. The defense works by parking parts of memory as non-executable. This means that if the CPU is ever asked to execute code in a non-executable area it will throw an exception. This prevents an attacker from writing his code onto the stack and then executing it.

3. Array bounds checking
   Every flavor of buffer overflow attack is possible because come array pointer is allowed to be incremented outside of its bounds and then have data written there. Therefore, the array bounds checking defense simply enforces a bounds check on all array pointers. Therefore, if an array pointer is ever incremented outside of its bounds an exception is thrown. This prevents all flavors of the buffer overflow attack.

4. Code pointer integrity checking
   This defense focuses on the idea that most buffer overflow attacks target control code, therefore, this defense places a known values around code that the system wants to ensure will not be altered. If these values are ever found altered this signals corruption of the control code and throws an exception. These known values are often referred to as canaries and there are several different types of canaries (e.g., random, static, random XOR).

5. Address space randomization (ASLR)
   A common practice for buffer overflow attacks is to direct control of the program to an address in the stack containing injected exploitation code or an address of some function or library. This attack requires the attacker to know or roughly

guess the address of the desired code. If the stack is always laid out in a predictable or predetermined way it becomes fairly easy to guess where to direct the CPU to, therefore, ASLR strives to ruin this predetermined or predictable stack layout. ASLR lays the stack out randomly putting code, the program stack, heap, environment variables, and library code in random locations. This makes it hard for an attacker to guess the right address at which to direct the return address.

- **We learned in class about a null terminator canary and a random canary. What is a limitation of the null terminator canary?**

The article from www.phrack.com talks about synthesizing an arbitrary pointer. In other words if a arbitrary pointer can be synthesized to point directly to the return address there is no need to write memory all the way through to it. In other words, one can simply bypass the canary all together and rewrite the return address. Now it is known that have the canary dose limit the number of programs that can be attacked but there are still programs that are vulnerable because one can simply skip over the canary.

Other article as discuss that the null terminator canary is vulnerable because it can be known. Therefore, theoretically an attack can simply overwrite the canary with itself and processed directly on to the return address with out the stack guard even knowing the difference.

- **What limitation of the random canary led to the development of the XOR canary??**

The article from www.phrack.com talks about synthesizing an arbitrary pointer. In other words if a arbitrary pointer can be synthesized to point directly to the return address there is no need to write memory all the way through to it. In other words, one can simply bypass the canary all together and rewrite the return address. The XOR canary prevents this attack because it allows stack guard to validate the return address also. So simply modifying the return address will not do the trick. This is because the canary is an XOR with a random number and the original return address, therefore, the return address and the canary must match with and XOR of a random number.