ECE568 Assignment 1

Mingqi Hou

999767676

Formal Code Verification

1. Briefly describe the idea behind formal code verification.

   The key idea behind formal code verification is to validate the correctness of a program with respect to a certain formal specification. The formal specification describes the exact behaviors of a program in the form of mathematical formulas. Formal code verification is performed by proof checking the program against the mathematical formula that defines the behaviors. Formal code verification is capable of verifying that a program achieves its specification with the same level of certainty that mathematicians prove theorems. As a result, formal code verification prevents defect in the program. In addition, there is a logical relationship between a statement and the preceding one in formally verified program. Overall, it is impossible to hack a formally verified program with existing technology.

2. Briefly describe two challenges that have prevented it from being widely adopted for commercial software.

   1) Technical difficulties

      There have been several technical challenges in developing formally verified programs, which prevented it from being widely adopted. One of them is the fact that turning a high level specification into mathematical definition is tricky. It is very difficult and labor intensive to transform the idea of behaviors into a formal specification, which must exclude all possible incorrect scenarios. In addition, the lack of proper tools in the early stages prevented commercial software from adopting it. A formal verified program could be as much as five times longer than the transitional program that performs the same task.

   2) Lack of need

      Before the age of internet, it was doubted whether formal code verification was necessary. Formal code verification prevents coding errors, which could lead to catastrophic failures. However, occasional crushes were tolerated by users and the industry. Instead of spending significant amounts of time, money and resources in formal code verification, commercial software developers considered the occasional crushes a good tradeoff. The tolerable software defects were not

considered security risks until the internet was widely adopted, which connected the systems that were previously isolated.

Return-Oriented Programming

1. Briefly describe "W ⊕X" protection, and explai
   smashing attacks.

   "W ⊕X" memory protection is a memory protection policy. It states that a memory location can be marked as writable or executable, but never both. This protection prevents attackers from injecting malicious code to take control over a process. The injected code is considered as data in memory. Diverting the control flow to execute the memory location marked as writable will result in a processor exception. Stack-smashing attacks are prevented as there is no location in memory available for attackers to inject and execute shell code.

2. Briefly explain what "gadgets" are, and how they are used in ROP.
   A gadget is the organizational unit of a return-oriented attack. It is an arrangement of words on the stack that carries out a well defined task when invoked. In another word, gadget is a set of carefully chosen machine instructions. The instructions are arranged in a particular sequence to perform a specific task, such as load operation. They allow attackers to perform arbitrary operations on a machine protected by "W ⊕X".

   ROP executes the gadgets by pointing the stack pointer to the first word in the gadget. The set of gadgets is chained together by pointing stack pointer to the next gadget when the ret instruction is executed on the current gadget. Together, ROP executes one gadget after another, allowing attackers to perform arbitrary operations.

Computer Virus-Antivirus Coevolution

1. Describe the challenge(s) does polymorphic viruss pose for signature-based detection.

   A virus signature is the sequence of machine-language instructions that is unique to the virus. Signature-based antivirus programs detect viruses by searching for fixed signatures. The polymorphic virus is designed to mutate itself each time it infects a new program. It consists of two major components, the decryption routine and the static portion of the virus. The decryption routine is generated by the mutation engine of the virus. A new decryption routine is produced when a new program gets infected. Signature-based detection is incapable of detecting the decryption routine of a polymorphic virus as it has no fixed signature. The static portion of polymorphic virus is encrypted by the complementary encryption routine generated by the mutation engine. As a result, signature-based detection is ineffective for polymorphic virus as the virus decryption routine varies for every infection and the virus body is encrypted.

2. Describe two different techniques that a polymorphic virus can utilize to make a GD antivirus program less effective.

   1) Employing “do-nothing” instructions

      A GD antivirus program detects polymorphic viruses by loading an executable file into a virtual machine. If the file is infected, the polymorphic virus will decrypt itself in the VM, allowing the antivirus program to detect the unencrypted body of the virus in a controlled environment. However, it is not realistic for A GD antivirus program to emulate the entire suspect program, especially if the file is clean. To solve this, ECM decides how long to emulate the program before giving up the scan and mark the program as “clean”. A polymorphic virus can take advantage of this mechanism by adding “do-nothing” instructions to the decryption routine to conceal its presence. The GD antivirus program will fail to detect the virus and terminate emulation prematurely if the ECM does not emulate enough instructions.

   2) Employing different machine languages

A GD antivirus program emulates the executable file for a particular CPU architecture. For example, by constructing decryption routine utilizing 80286-specific machine instructions, polymorphic virus will evade detection if the emulator is designed for 80486 series CPU. The emulator will not be able to emulate and decrypt the virus. Although the virus doesn't work in 80486 machines, it is capable of infecting the older 80286 machines.

Programming Error

1. What is the vulnerability? What does it allow the attacker to do?
   It has a stack overflow vulnerability. It allows the attackers to overwrite variables, including return pointer, on the stack. The control flow is diverted once the return pointer is overwritten. By taking over the control flow, attacker can perform arbitrary operations such as executing injected shell code.

2. What line(s) would you change and what would you change them to correct the vulnerability?
   Line 17, "strcpy(outbuf, buf);" is the cause of the stack overflow vulnerability. It performs string copy from a source array that is longer than destination array. To eliminate the vulnerability, line 17 (strcpy) must be changed to a new method that accounts for the size of the destination array, which is smaller than source array.

   The following 2 lines of code can be used to replace line 17 to correct the vulnerability.

   ```
   17: strncpy(outbuf, buf, 62);
   18: outbuf[63] = NULL;
   ```