Daniel Willard
433 Network Security
15FEB2024
Homework 2

1) Describe how the following virus works. (4 pts)

```
Program V :=
{goto main;
1234567;
subroutine infect-executable :=
        {loop: file:= get-random-executable-file;
                if (first-line-of-file = 1234567) then goto loop
                else prepend V to file;
        }
subroutine do-damage :=
        { whatever damage is to be done
        }
subroutine trigger-pulled :=
         { return true if some condition holds
         }
main:
         main-program :=
         {infect-executable;
          if trigger-pulled then do-damage;
          goto next;
          }
next:
}
```

AWS: It is a Modification program on a program and is a Logic Bomb. This virus operates by infecting executable files, checking for a trigger condition, and causing damage if the trigger condition is met.

Chapter 3, Problems

- 4 (4pts) **4.** A program is written to compute the sum of the integers from 1 to 10. The programmer, well trained in reusability and maintainability, writes the program so that it computes the sum of the numbers from *k* to *n*. However, a team of security specialists scrutinizes the code. The team certifies that this program properly sets *k* to 1 and *n* to 10; therefore, the program is certified as being properly restricted in that it always operates on precisely the range 1 to 10. List different ways that this program can be sabotaged so that during execution it computes a different sum, such as 3 to 20.
    - **Buffer Overflow** if 55 is the largest int then the program might have only assigned 6 bits if 3-20 is summed to 207 we could buffer overflow and overwrite important data. Also if the check is only that k is one digit and n is 2 digit we could sum 1-99 and overwrite a lot.
    - **Time of check to time of use (TOCTTOU)** lets assume that there is no buffer overflow and the user can put in a large values in and the code is running in parallel this could cause and infinite looping as is check checking and adding over and over
    - Direct Modification of Variables
    - **Input Manipulation**
    - **Injection Attacks**
    - Exploiting Weaknesses in Input Handling
    - Tampering with Program Execution
    - Exploiting Vulnerabilities in Dependencies
    - Backdoor Insertion
    - Integer overflow
    - Etc.
- 6 (4pts) 6.One way to limit the effect of an untrusted program is confinement: controlling what processes have access to the untrusted program and what access the program has to other processes and data. Explain how confinement would apply to the earlier example of the program that computes the sum of the integers 1 to 10.
    - Access control and resource limitations would be the main ways to limit the effects of untrusted programs. This could be done by executing the program within a constrained environment, such as a sandbox or a virtual machine, where its access to system resources and data is restricted. The sandbox environment would limit the program's ability to interact with other processes and sensitive data, reducing the potential impact of any malicious behavior. The practice of Least privilege would ensure that the program doesn't have access to anything unneeded or extra. Confinement using the policy of least privilege could involve setting limits on the resources that the program can consume, such as CPU time, memory usage, and file system access. By imposing these restrictions, the program's ability to perform malicious activities, such as consuming excessive system resources or accessing sensitive files, is constrained. You could also isolated environment where it has minimal interaction with other processes and data.
    - confinement plays a critical role in mitigating the risks associated with untrusted programs by limiting their capabilities and access to resources. By implementing

confinement measures such as access control, resource limitations, isolation, least privilege, and monitoring, organizations can reduce the likelihood of unauthorized access, data breaches, and system compromises.

- 22 (6pts)22. Provide answers for these two situations; justify your answers.
    - 1)You receive an email message that purports to come from your bank. It asks you to click a link for some reasonable- sounding administrative purpose. How can you verify that the message actually did come from your bank?
        - Verify that the email address from which the message was sent matches the official domain of your bank.
        - Hover over the link provided in the email (without clicking on it) to see the actual URL it leads to. Ensure that it directs to the official website of your bank and not a phishing site.
        - Instead of clicking on the link provided in the email, independently visit the bank's official website or contact them through their verified contact information to inquire about the email.
        - Often, phishing emails contain grammatical errors, spelling mistakes, or unusual language usage. Authentic emails from legitimate organizations usually undergo proofreading and are free of such errors.
        - Legitimate emails from banks often include security indicators such as encryption, digital signatures, or specific information about your account that only you and the bank would know.
    - 2)Now play the role of an attacker. How could you intercept the message described in part (a) and convert it to your purposes while still making both the bank and the customer think the message is authentic and trustworthy?
        - Send a fake email to the recipient using a spoofed email address that closely resembles the bank's official domain.
        - Set up a phishing website that mimics the bank's login page.
        - When the recipient clicks on the link in the email and enters their login credentials on the phishing website, capture this information.
        - After capturing the user's credentials, redirect them to the bank's legitimate website to avoid raising suspicion.
        - Erase any traces of the phishing website and email spoofing activities to avoid detection.

Chapter 5, Problems:

- 3 (4pts) **3.** Give an example of an object whose sensitivity may change during execution. Describe the difficulties of an operating system handling such a change in sensitivity. For example, how does it deal with a subject who, having originally qualified for access to an object, now should lose that access right?
    - one example of an object whose sensitivity may change during execution is a software configuration file. Initially, when the software is deployed, the configuration file may contain sensitive information such as database credentials, API keys, or encryption keys. Due to the sensitive nature of this information, the configuration file is assigned a high sensitivity level. However, during the operation of the software, certain events or conditions may occur that necessitate changes to the configuration file's sensitivity.
    - The software may offer certain features that can be enabled or disabled via configuration settings.
    - Changes in regulatory requirements or industry standards may necessitate adjustments to the configuration settings to ensure compliance.
    - Periodically, the software may rotate its database credentials or API keys for security reasons.
    - Operating systems and software platforms must be capable of handling such changes in sensitivity by enforcing access control policies, managing permissions, and ensuring the secure management of configuration files to protect sensitive information from unauthorized access or disclosure.
- 9 (4pts) **9.** What are some other modes of access that users might want to apply to code or data, in addition to the common *read*, *write*, and *execute* permission?
    - Append – allows the file to be added to without modify existing data.
    - Delete – remove file or directory.
    - Traverse – enables users to traverse a directory tree to access files or directories within a parent directory.
    - Ownership - This permission allows users to transfer ownership of a file or directory to another user or group.
    -
-
- 17 (4pts) **17.** Describe a mechanism by which an operating system can enforce limited transfer of capabilities. That is, process A might transfer a capability to process B, but A wants to prevent B from transferring the capability to any other processes.
    - One mechanism by which an operating system can enforce limited transfer of capabilities is through the use of "cascading capabilities" or "delegated capabilities"
    - Initially, process A is granted a capability by the operating system. Process A transfers the capability to process B through a designated system call or mechanism provided by the operating system. This transfer of capability allows process B to perform the privileged operation on behalf of process A.
    - However, when process B receives the capability from process A, the operating system imposes a restriction on subsequent transfers of the capability. This restriction ensures that process B cannot further transfer the capability to any

    other processes. Additionally, the operating system retains the ability to revoke the capability from process B if necessary, based on predefined conditions or security policies. This prevents process B from retaining the capability indefinitely and potentially abusing it.
  - The capability transferred from process A to process B may have a limited scope or duration, meaning it is only valid for a specific period of time or for a specific set of operations
- 18 (4pts) **18.** List two disadvantages of using physical separation in a computing system. List two disadvantages of using temporal separation in a computing system.
  - Physical separation:
    - Cost – Implementing physical separation in a computing system often involves duplicating hardware components, such as servers, storage devices, and network infrastructure, to create isolated environments. These items are costly
    - Complexity - Managing and maintaining physically separated environments can be complex and resource-intensive.
  - temporal separation:
    - resource Utilization - Temporal separation involves sharing physical resources, such as CPU, memory, and storage, among multiple virtualized environments running on the same hardware platform.
    - Dependency on virtualization - Temporal separation relies on a hypervisor or virtualization layer to manage and allocate resources among virtualized environments. If the hypervisor experiences a failure or becomes compromised, all virtualized environments running on the affected hardware platform may be impacted.
- 21 (6pts) **21.** A flaw in the protection system of many operating systems is argument passing. Often a common shared stack is used by all nested routines for arguments as well as for the remainder of the context of each calling process.
  - 1)Explain what vulnerabilities this flaw presents.
    - Data leakage - If one routine fails to properly sanitize or protect its arguments, sensitive information such as passwords, cryptographic keys, or personal data may be exposed to other routines on the stack.
    - Buffer Overflow – If a routine fails to validate the size of incoming arguments or context, an attacker may exploit this vulnerability by providing excessively large inputs, leading to buffer overflow and potential code execution exploits.
    - Tampering with context – By modifying the return addresses, local variables, or function pointers stored on the stack, an attacker may hijack control of the program and execute arbitrary code.
    - Denial of service - Malicious or unexpected inputs passed through the shared stack could lead to DoS attacks.
  - 2)Explain how the flaw can be controlled. The shared stack is still to be used for passing arguments and storing context.
    - Input validation – implement strict input validation mechanisms to ensure that arguments passed through the shared stack are within expected ranges and formats

- Memory protection – Utilize memory protection techniques such as stack canaries, address space layout randomization (ASLR), and non-executable stacks to mitigate buffer overflow exploits and protect against unauthorized access or modification of the stack.
- Contextual integrity checks – Implement checks to verify the integrity of the context stored on the shared stack
- Privilege separation – Limit the privileges of routines accessing the shared stack to reduce the potential impact of security breaches.