Dalton Murray

INT 7223 Cybersecurity

Dr. Hany Othman

June 12, 2023

Chapter 10 - 12 Review

Question

10.1 Define buffer overflow

Buffer overflow, which is also known as buffer overrun as well as buffer overwrite is defined as “A condition at an interface under which more input can be placed into a buffer or data holding area than the capacity allocated, overwriting information” (Stallings & Brown, 2018. p. 321). This means that buffer overflow is when we have a set capacity limit of something and then we try to put more into something than there is capacity. For example, a variable is set to only hold 8 characters but we try to put 12 characters into it.

Buffer overflows can occur for a variety of reasons but will often occur as a programming error (Stallings & Brown, 2018. p. 321). A buffer overflow will also cause adjacent memory to be overwritten which will cause data loss, it could also corrupt data used by the program, memory access violations, and even program termination, however when used as an attack can be used to execute code on the system (Stallings & Brown, 2018. p. 321).

10.2 List the three distinct types of locations in a process address space that buffer overflow attacks typically target

The three distinct types of locations in a process address space that buffer overflow attacks typically target are: the stack, the heap, and the data section of the process (Stallings & Brown, 2018. p. 321).

10.3 What are the possible consequences of a buffer overflow occurring?

The possible consequences of a buffer overflow occurring are: corruption of data that the program uses, unexpected transfer of control in the program, memory access violations, and program termination (Stallings & Brown, 2018. p. 321). On top of all of these another consequence is the ability for an attacker execute arbitrary code with privilege of the attacked process (Stallings & Brown, 2018. p. 321).

10.4 What are the two key elements that must be identified in order to implement a buffer overflow?

The two key elements that must be identified in order to implement a buffer overflow are: identifying a buffer overflow vulnerability in a program which can be triggered by using externally sourced data which is under the attackers control, and to understand how the buffer will be stored in the process memory hence the potential for corrupting adjacent memory locations and altering the flow of the execution of the program (Stallings & Brown, 2018. p. 324).

11.1 Define the difference between software quality and reliability and software security

Software quality and reliability – With software quality and reliability the concern is with the accidental failure of a program as a result of something which is in theory random, an unanticipated input, system interaction, or the use of incorrect code (Stallings & Brown, 2018. p. 359 - 360). Often these failures will follow a form of probability distribution, and in order to improve software quality you will typically use a form of structured design and testing in order to identify and eliminate as many bugs as you reasonably can (Stallings & Brown, 2018. p. 360). When testing with software quality and reliability you will try many variations of likely inputs and common errors and the concern is not the total number of bugs but how often they are triggered (Stallings & Brown, 2018. p. 360).

Software security – With software security the attacker will choose the probability distribution and target specific bugs which result in a failure that allows them to exploit them (Stallings & Brown, 2018. p. 360). The bugs they exploit will often be able to be triggered by inputs that differ from what is used in a day-to-day scenario and are unlikely to be spotted during normal software quality and reliability testing (Stallings & Brown, 2018. p. 360). We want to pay attention to all aspects of how a program executes, the environment it executes in, and the type of data it processes (Stallings & Brown, 2018. p. 360). In software security nothing can be assumed and all potential errors are to be checked (Stallings & Brown, 2018. p. 360).

11.2 Define defensive programming

Defensive programming, also known as secure programming, is defined as “the process of designing and implementing software so it continues to function even when under attack. Software written using this process is able to detect erroneous conditions resulting from some attack, and to either continue executing safely, or to fail gracefully. The key rule in defensive programming is to never assume anything, but to check all assumptions and to handle any possible error states” (Stallings & Brown, 2018. p. 360). In other words, this means that in defensive programming we have the goal of designing and implementing software which allows it to function properly even when an attacker is attempting to attack the system. We want to be able to make a system so that it is able to detect conditions which result from an attack and then continue to execute safely or allow the system to fail gracefully and have no loss of data or other permanent problems. Again, we want to never assume anything and make sure we check all assumptions and to handle every possible error state.

11.3 List some possible sources of program input

Some possible sources of program input are: user keyboard or mouse entry, files, network connections, data supplies to the program in the execution environment, the values of any configuration or other data which is read from files by the program, and any values which are supplied by the operating system to the program (Stallings & Brown, 2018. p. 362).

12.1 What are the basic steps needed in the process of securing a system?

The basic steps needed in the process of securing a system are: assessing risks and planning for the system deployment, securing the underlying operating system and its key applications, ensuring any critical content which is required is secured, ensure all appropriate network protection mechanisms are actively being used, as well as ensuring that appropriate processes are being used to maintain security (Stallings & Brown, 2018. p. 399).

12.2 What is the aim of system security planning?

The aim of system security planning is to maximize security while also minimizing costs (Stallings & Brown, 2018. p. 400). With system security planning, it is much more expensive and difficult to retro-fit security so you want to plan for it first in the initial deployment process (Stallings & Brown, 2018. p. 400). As a result of proper planning we can then use this to guide the selection of appropriate software which meets our costs and levels of security as well as identify who will install it, if more measures for hardening are required, and what configurations should be (Stallings & Brown, 2018. p. 400).

12.3 What are the basic steps needed to secure the base operating system?

The basic steps needed to secure the base operating system are: “Install and patch the operating system”, “Harden and configure the operating system to adequately address the identified security needs of the system by: Removing unnecessary services, applications, and protocols. Configuring users, groups, and permissions. Configuring resource controls”, “Install and configure additional security controls, such as anti-virus, host-based firewalls, and intrusion detection systems (IDS), if needed” and “Test the security of the basic operating system to ensure that the steps taken adequately address its security needs” (Stallings & Brown, 2018. p. 401).

Install and patch the operating system – It’s extremely important to install all updates and patch operating systems. If an operating system is left outdated or does not have the patches required the entire system is vulnerable and may be easily broken into, especially if it’s a known vulnerability.

Harden and configure the operating system –

Remove unnecessary services, applications, and protocols – We want to make sure that when we installed the system we installed it in a minimal form, but we also want to removed anything that is unused or unnecessary. The more things installed, the more potential and risk there is.

Configure users, groups, and permissions – We want to ensure that all users, groups, and permissions are configured properly so that everyone is able to do only what they are supposed to do and nothing extra. This prevents people from

accessing things they shouldn't but also allows for if an attacker gains access to an account to not be able to do everything on the system.

Configure resource controls – Permissions need to be properly set for resources and limits need to be stablished.

Install and configure additional security controls – Additional security controls such as antiviruses, intrusion detection and prevention systems, firewalls all should be installed as it will assist in hardening the system.

Test the security – Finally we have to actually test the system, see if anyone can gain access to things they shouldn't and if you can try to gain access to the system externally.

Problem

11.5 Investigate the functions available in PHP, or another suitable Web scripting language, to sanitize any data subsequently used in an SQL query.

Thankfully, PHP is a decently well-documented language and I am able to find the functions available in PHP to sanitize data used in an SQL query fairly easily. Following this link: https://www.php.net/manual/en/filter.filters.sanitize.php we can see there are a number of different functions. These functions, and generally what they do are as follows:

FILTER_SANITIZE_EMAIL – With this filter, it removes all characters except for letters and digits as well as the following special characters: !#$%&'*+-=?^_`{|}~@.[].

FILTER_SANITIZE_ENCODED – With this filter, it filters URL-encoded strings, and can optionally strip or encode special characters

FILTER_SANITIZE_MAGIC_QUOTES – With this filter, it applies the addslashes() function which quotes the entire string and adds backslashes for characters that need to be escaped (deprecated)

FILTER_SANITIZE_ADD_SLASHES – With this filter, it applies the addslashes() function (not-deprecated)

FILTER_SANITIZE_NUMBER_FLOAT – With this filter, it removes all characters except for digits and the characters: +- and optionally .,eE

FILTER_SANITIZE_NUMBER_INT – With this filter, it removes all characters except digits and +-

FILTER_SANITIZE_ SPECIAL_CHARS – With this filter, it removes HTML characters ' "<>& and characters with an ASCII value less than 32, and can optionally strip or encode other special characters

FILTER_SANITIZE_FULL_SPECIAL_CHARS – With this filter, it is the equivalent of calling HTML’s htmlspecialchars() function with ENT_QUOTES set, if a sequence of bytes is detected that makes up an invalid character in the current character set then the entire string is rejecting which results in a 0 length string

FILTER_SANITIZE_STRING – With this filter, it strips tags and HTML encoding double and single quotes, and can optionally strip or encode special characters (deprecated)

FILTER_SANITIZE_STRIPPED – With this filter, it is an alias of the STRING filter (deprecated)

FILTER_SANITIZE_URL – With this filter, it removes all characters except letters, digits, and the characters $-_.+!*'(),{}|\\^~[]`<>#%";/?:@&=.

FILTER_UNSAFE_RAW – With this filter, it does nothing, but can optionally strip or encode special characters

Keep in mind that these are the general filters for sanitization, and there are individual flags for everything that can be individual used which is quite an extensive list, most of which can be found here: https://www.php.net/manual/en/filter.filters.php here https://www.php.net/manual/en/ref.filter.php and here https://www.php.net/manual/en/book.filter.php

References

Stallings, W., & Brown, L. (2018). *Computer security: Principles and practice*. Pearson.

I have neither given nor received unauthorized aid in completing this work, nor have I presented someone else's work as my own.

Dalton Murray