Research Report

BUFFER OVERFLOW & LONG PASSWORD DOS ATTACK

Guide: Mr Ali J & Mr Kuldeep L M

Jayashankar P _ Sypder 9

Red Team Intern @ Cyber Sapiens

Dec 14th 2024

BUFFER OVERFLOW VULNERABILITY

What is Buffer ?

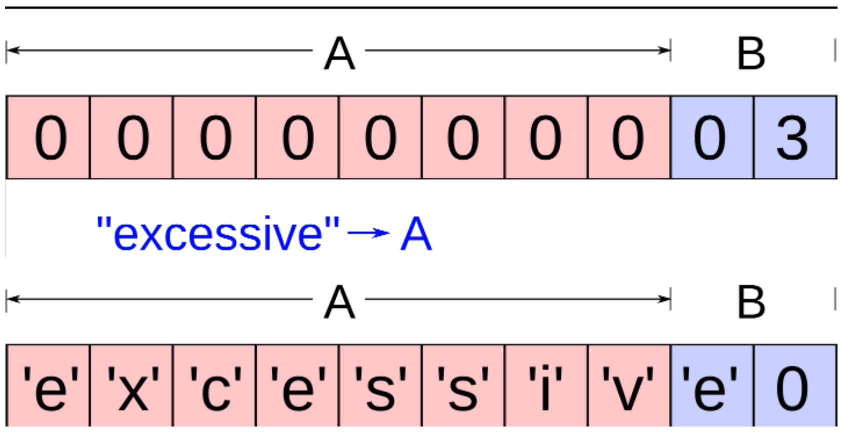
A buffer is a temporary storage area in memory used to hold data while it is being transferred from one place to another. It acts as an intermediary between two processes or devices that may operate at different speeds or have different data transfer rates.

Buffers are commonly used in various scenarios, including I/O operations (e.g., reading from or writing to a file), network communication (e.g., sending or receiving data over a network), and data processing (e.g., data transformation or manipulation).

The primary purpose of a buffer is to smooth out the differences in data transfer rates between the source and the destination, ensuring a steady flow of data and preventing data loss or corruption.

What is Buffer Overflow Vulnerability ?

Buffer overflow is a critical security vulnerability that occurs when a program attempts to write more data to a buffer (a region of memory) than it can hold. This excess data overwrites adjacent memory locations, potentially corrupting data, crashing the program, or even allowing attackers to execute malicious code.



Key points:

- ✓ Data Input: A program receives data from a user or another program.
- ✓ Buffer Allocation: The program allocates a buffer in memory to store the data.
- ✓ Data Overflow: If the input data exceeds the buffer's capacity, the excess data spills over into adjacent memory locations.

Example:

Consider a program that stores usernames in a buffer of 10 characters. If a user enters a username longer than 10 characters, the excess characters will overflow the buffer and overwrite adjacent memory locations.

Consequences of Buffer Overflow Vulnerability

- Category: Availability: Buffer overflows generally lead to crashes. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.
- Access control (instruction processing): Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy.
- Other: When the consequence is arbitrary code execution, this can often be used to subvert any other security service.

Types of Buffer Overflow Vulnerability

Stack-based	Heap-based
Stack-based buffer overflows are more common, and leverage stack memory that only exists during the execution time of a function.	Heap-based attacks are harder to carry out and involve flooding the memory space allocated for a program beyond memory used for current runtime operations
A stack-based buffer overflow occurs when a program writes more data to a buffer on the stack, than it can hold.	This can overwrite adjacent heap metadata, potentially leading to a crash or allowing an attacker to execute arbitrary code

How to exploit Buffer Overflow Vulnerability

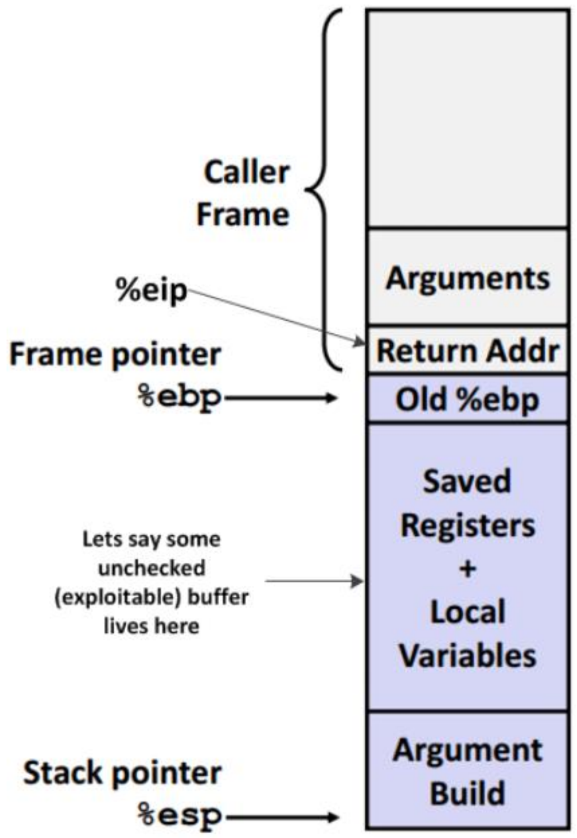
The techniques to exploit a buffer overflow vulnerability vary by architecture, operating system, and memory region. For example, exploitation on the heap (used for dynamically allocated memory), differs markedly from exploitation on the call stack. In general, heap exploitation depends on the heap manager used on the target system, while stack exploitation depends on the calling convention used by the architecture and compiler.

❖ Stack-based exploitation

There are several ways in which one can manipulate a program by exploiting stack-based buffer overflows:

- ✓ Changing program behavior by overwriting a local variable located near the vulnerable buffer on the stack
- ✓ By overwriting the return address in a stack frame to point to code selected by the attacker, usually called the shell code. Once the function returns, execution will resume at the attacker's shell code
- ✓ By overwriting a function pointer or exception handler to point to the shell code, which is subsequently executed
- ✓ By overwriting a local variable (or pointer) of a different stack frame, which will later be used by the function that owns that frame.

The attacker designs data to cause one of these exploits, then places this data in a buffer supplied to users by the vulnerable code. If the address of the user-supplied data used to affect the stack buffer overflow is unpredictable, exploiting a stack buffer overflow to cause remote code execution becomes much more difficult. One technique that can be used to exploit such a buffer overflow is called "trampolining". Here, an attacker will find a pointer to the vulnerable stack buffer and compute the location of their shellcode relative to that pointer. The attacker will then use the overwrite to jump to an instruction already in memory which will make a second jump, this time relative to the pointer. That second jump will branch execution into the shellcode. Suitable instructions are often present in large code. The Metasploit Project, for example, maintains a database of suitable opcodes, though it lists only those found in the Windows operating system.



❖ Heap-based exploitation

A buffer overflow occurring in the heap data area is referred to as a heap overflow and is exploitable in a manner different from that of stack-based overflows. Memory on the heap is dynamically allocated by the application at run-time and typically contains program data. Exploitation is performed by corrupting this data in specific ways to cause the application to overwrite internal structures such as linked list pointers. The canonical heap overflow technique overwrites dynamic memory allocation linkage (such as malloc meta data) and uses the resulting pointer exchange to overwrite a program function pointer.

Microsoft's GDI+ vulnerability in handling JPEGs is an example of the danger a heap overflow can present.

How to mitigate Buffer Overflow Vulnerability

Developers can protect against buffer overflow vulnerabilities via security measures in their code, or by using languages that offer built-in protection.

In addition, modern operating systems have runtime protection. Three common protections are:

- Address space randomization (ASLR)—randomly moves around the address space locations of data regions. Typically, buffer overflow attacks need to know the locality of executable code, and randomizing address spaces makes this virtually impossible.
- Data execution prevention—flags certain areas of memory as non-executable or executable, which stops an attack from running code in a non-executable region.
- Structured exception handler overwrite protection (SEHOP)—helps stop malicious code from attacking Structured Exception Handling (SEH), a built-in system for managing hardware and software exceptions. It thus prevents an attacker from being able to make use of the SEH overwrite exploitation technique. At a functional level, an SEH overwrite is achieved using a stack-based buffer overflow to overwrite an exception registration record, stored on a thread’s stack.

Security measures in code and operating system protection are not enough. When an organization discovers a buffer overflow vulnerability, it must react quickly to patch the affected software and make sure that users of the software can access the patch.

LONG PASSWORD DENIAL ATTACK

Introduction

Imagine a website where you log in with a password. Normally, short passwords are fine. But a hacker could try to log in with a ridiculously long password, like a million characters.

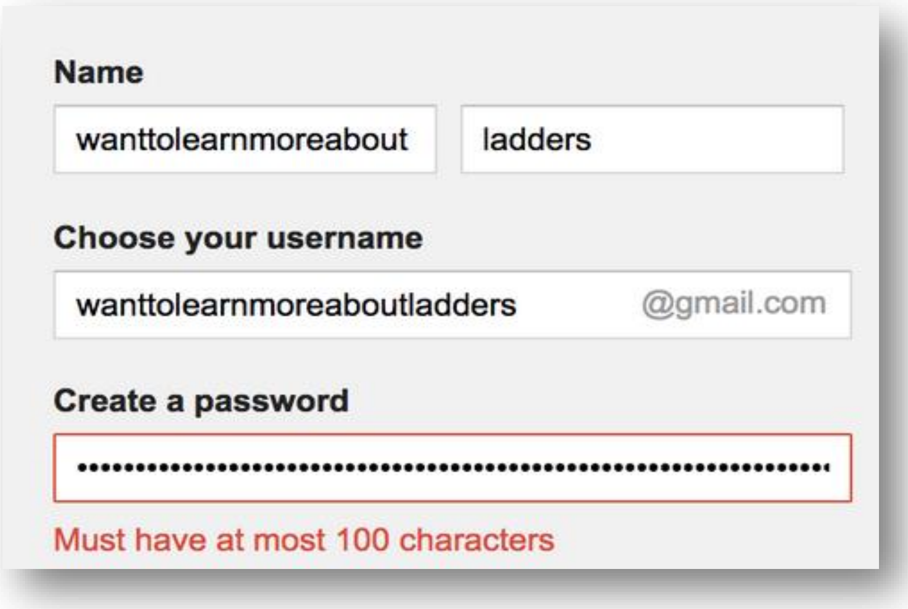
This super long password can trick the website's computer into working super hard to process it. It's like giving a tiny car a huge boulder to push – the car gets stuck and can't do anything else.

This makes the website slow down or even crash completely, so nobody else can use it. That's the "denial of service" part – the website is denied to normal users.

In simple terms:

- Hackers use a very long password to trick a website's computer.
- This tricks the computer into working too hard.
- The website gets overloaded and stops working for everyone else.

This is a sneaky way for hackers to make a website unusable, even though they don't actually want to log in.



Name

wanttolearnmoreabout | ladders

Choose your username

wanttolearnmoreaboutladders @gmail.com

Create a password

••

Must have at most 100 characters

By sending a very long password (1.000.000 characters) it's possible to cause a denial a service attack on the server. This may lead to the website becoming unavailable or unresponsive. Usually this problem is caused by a vulnerable password hashing implementation. When a long password is sent, the password hashing process will result in CPU and memory exhaustion.

This vulnerability was detected by sending passwords with various lengths and comparing the measured response times. Consult details for more information.

Impact of Long Password DOS Attack

The impact of a Long Password Denial of Service (DoS) attack can be significant:

- **System Unavailability:** The primary impact is the denial of service, preventing legitimate users from accessing the system or its services. This can disrupt business operations, cause financial losses, and damage reputation.
- **Resource Depletion**: The attack can drain system resources, such as CPU and memory, impacting other critical functions. This can lead to degraded performance for other users and applications running on the system.
- **Security Risk:** While the primary goal is to disrupt service, the attack might also reveal vulnerabilities in the system's password handling implementation. These vulnerabilities could be exploited by attackers to gain unauthorized access or compromise the system's security in other ways.
- **Lost Productivity:** For individuals and businesses, system unavailability caused by the attack can lead to lost productivity and wasted time.

In essence, a successful Long Password DoS attack disrupts normal system operations, impacts resource availability, and potentially exposes underlying security weaknesses.

How to test Long Password Denial Of Service

Testing for long password Denial of Service (DoS) involves examining how a system handles unusually long passwords during authentication. This test is crucial for identifying vulnerabilities that could lead to service interruptions or potential exploitation.

1. Analyze Input Limits:
   - Research the system’s password policy and input size restrictions (if disclosed).
   - Use tools or scripts to determine the maximum password length accepted by the application.
2. Craft the Payload:
   - Generate excessively long passwords (e.g., 10,000+ characters) using a script or automated tool.
   - Examples: Python: long_password = 'A' * 10000

     Bash: head -c 10000 < /dev/zero | tr '\0' 'A'
3. Send Authentication Requests:
   - Attempt to log in using a long password with an existing username or dummy user.
   - Observe the behavior of the system:
     - ✓ Response times.
     - ✓ Errors or exceptions.
     - ✓ CPU or memory spikes on the server.
4. Monitor System Behavior:
   - Use monitoring tools (e.g., top, htop, or server logs) to detect abnormal resource consumption.
   - Look for signs like:
     - ✓ Slower responses across the application.
     - ✓ Increased CPU or memory usage during login attempts.
5. Edge Cases:
   - Test variations such as:
     - ✓ Long passwords with different character sets (e.g., special characters, Unicode).
     - ✓ Empty password fields combined with long usernames.

Tools to test Long Password Denial Of Service

BURPSUITE	ZAP	Metasploit
BURP SUITE	**OWASP ZAP**	
For crafting and sending payloads	For automated vulnerability scanning	For advanced penetration testing

How to mitigate Long Password DOS Attack

- Enforce reasonable limits on password length (e.g., 256 characters).
- Validate input server-side to ensure uniform security.
- Apply rate limiting to authentication requests to prevent brute force and resource exhaustion.
- Use algorithms like bcrypt, Argon2, or PBKDF2 for password hashing, with input size checks to avoid computational bottlenecks.
- Implement CAPTCHAs.

References

1. https://www.geeksforgeeks.org/buffer-overflow-attack-with-example
2. https://en.wikipedia.org/wiki/Buffer_overflow
3. https://www.imperva.com/learn/application-security/buffer-overflow/
4. https://www.acunetix.com/vulnerabilities/web/long-password-denial-of-service/#:~:text=By%20sending%20a%20very%20long,a%20vulnerable%20password%20hashing%20implementation.