Task 2.1

The **Common Vulnerabilities and Exposures** (**CVE**) system provides a reference-method for publicly known information-security vulnerabilities and exposures.

**CVE Usage**

CVE identifiers are intended for use with respect to identifying vulnerabilities:

Common Vulnerabilities and Exposures (CVE®) is a dictionary of common names (i.e., CVE Identifiers) for publicly known information security vulnerabilities. CVE’s common identifiers make it easier to share data across separate network security databases and tools, and provide a baseline for evaluating the coverage of an organization’s security tools. If a report from one of your security tools incorporates CVE Identifiers, you may then quickly and accurately access fix information in one or more separate CVE-compatible databases to remediate the problem.

**Things we need to know before knowing the vulnerability:**

**Stack buffer overflow:**

In software, a **stack buffer overflow** or **stack buffer overrun** occurs when a program writes to a memory address on the program's call stack outside of the intended data structure, which is usually a fixed-length buffer.Stack buffer overflow bugs are caused when a program writes more data to a buffer located on the stack than what is actually allocated for that buffer. This almost always results in corruption of adjacent data on the stack, and in cases where the overflow was triggered by mistake, will often cause the program to crash or operate incorrectly. Stack buffer overflow is a type of the more general programming malfunction known as buffer overflow (or buffer overrun).Overfilling a buffer on the stack is more likely to derail program execution than overfilling a buffer on the heap because the stack contains the return addresses for all active function calls.

[Best example :<https://en.wikipedia.org/wiki/Stack_buffer_overflow>]

[Little about **Call Stack,**

A **call stack** is a stack data structure that stores information about the active **subroutines** of a computer program. This kind of stack is also known as an **execution stack**, **control stack**, **run-time stack**, or **machine stack**, and is often shortened to just "the stack". A call stack is used for several related purposes, but the main reason for having one is to keep track of the point to which each active subroutine should return control when it finishes executing.

( **Subroutine:** is a sequence of program instructions that perform a specific task, packaged as a unit. This unit can then be used in programs wherever that particular task should be performed. Subprograms may be defined within programs, or separately in libraries that can be used by multiple programs. In different programming languages, a subroutine may be called a **procedure**, a **function**, a **routine**, a method, or a **subprogram**. a subroutine behaves in much the same way as a computer program that is used as one step in a larger program or another subprogram. A subroutine is often coded so that it can be started (called) several times and from several places during one execution of the program, including from other subroutines, and then branch back (*return*) to the next instruction after the *call* once the subroutine's task is done.)

]

**CVE-2015-7547**

CVE-2015-7547, is a **stack-based buffer overflow** vulnerability in glibc's DNS client-side resolver function **getaddrinfo()** that is used to translate human-readable domain names, like google.com, into a network IP address.

This can be exploited in a variety of scenarios, including man-in-the-middle attacks, maliciously crafted domain names, and malicious DNS servers.

[ Little about **getaddrinfo(),**

The functions **getaddrinfo()** convert domain names, hostnames, and IP addresses between human-readable text representations and structured binary formats for the operating system's networking API.

## getaddrinfo()

getaddrinfo() converts human-readable text strings representing hostnames or IP addresses into a dynamically allocated linked list of **struct addrinfo** structures. The function prototypes for these functions are specified as follows:

#include <sys/types.h>

#include <sys/socket.h>

#include <netdb.h>

int getaddrinfo(const char\* hostname,

const char\* service,

const struct addrinfo\* hints,

struct addrinfo\*\* res);

* **hostname** can be either a domain name, such as "example.com", an address string, such as "127.0.0.1", or NULL, in which case the address 0.0.0.0 or 127.0.0.1 is assigned depending on the hints flags.
* **service** can be a port number passed as string, such as "80", or a service name, e.g. "echo". In the latter case, **gethostbyname()** is used to query the file */etc/services* to resolve the service to a port number.
* **hints** can be either NULL or an **addrinfo** structure with the type of service requested.
* **res** is a pointer that points to a new **addrinfo** structure with the information requested after successful completion of the function.

The function returns 0 upon success and negative if it fails

]

**How Does the Flaw Work?**

The buffer overflow flaw is triggered when the **getaddrinfo()**library function that performs domain-name lookups is in use, allowing hackers to remotely execute malicious code.

The flaw can be exploited when an affected device or app make queries to a malicious DNS server that returns too much information to a lookup request and floods the program's memory with code.

This code then compromises the vulnerable application or device and tries to take over the control over the whole system.

It is possible to inject the domain name into server log files, which when resolved will trigger remote code execution. An SSH (Secure Shell) client connecting to a server could also be compromised.

However, an attacker need to bypass several operating system security mechanisms – like ASLR and non-executable stack protection– in order to achieve successful RCE(remote code execution) attack

Alternatively, an attacker on your network could perform **man-in-the-middle** (MitM) attacks and tamper with DNS replies in a view to monitoring and manipulating (injecting payloads of malicious code) data flowing between a vulnerable device and the Internet.

**Things happen to glibc :**

glibc reserves 2048 bytes in the stack through **alloca()** for the DNS answer at **\_nss\_dns\_gethostbyname4\_r()** for hosting responses to a DNS query. Later on, at **send\_dg()** and **send\_vc(),** if the response is larger than 2048 bytes, a new buffer is allocated from the heap and all the information (buffer pointer, new buffer size and response size) is updated.

Under certain conditions a mismatch between the stack buffer and the new heap allocation will happen. The final effect is that the stack buffer will be used to store the DNS response, even though the response is larger than the stack buffer and a heap buffer was allocated. This behavior leads to the stack buffer overflow.

**Affected Software and Devices:**

All versions of glibc after 2.9 are vulnerable. Therefore, any software or application that connects to things on a network or the Internet and uses glibc is at RISK.

The widely used SSH, sudo, and curl utilities are all known to be affected by the buffer overflow bug, and security researchers warn that the list of other affected applications or code is almost too diverse and numerous to enumerate completely.

The vulnerability could extend to a nearly all the major software, including:

* Virtually all distributions of Linux.
* Programming languages such as the Python, PHP, and Ruby on Rails.
* Many others that use Linux code to lookup the numerical IP address of an Internet domain.
* Most Bitcoin software is reportedly vulnerable, too.

**ASLR (Address Space Layout Randomization):**

Address space layout randomization (ASLR) is a memory-protection process for operating systems (OSes) that guards against buffer-overflow attacks by randomizing the location where system executables are loaded into memory.

ASLR randomly arranges the **address space** positions of key data areas of a process, including the base of the executable and the positions of the stack, heap and libraries.

[Little about **Address space,**

It defines a range of discrete addresses, each of which may correspond to a network host, peripheral device, disk sector, a memory cell or other logical or physical entity.  
For software programs to save and retrieve stored data, each unit of data must have an address where it can be individually located or else the program will be unable to find and manipulate the data. The number of address spaces available will depend on the underlying address structure and these will usually be limited by the compute architecture being used.]

**How it works:**

The success of many cyber-attacks, particularly **zero-day exploits**, relies on the hacker's ability to know or guess the position of processes and functions in memory. For example, attackers trying to execute **return-to-libc** **attacks** must locate the code to be executed, while other attackers trying to execute **shellcode** injected on the stack have to find the stack first. ASLR is able to put address space targets in unpredictable locations. If an attacker attempts to exploit an incorrect address space location, the target application will crash, stopping the attack and alerting the system.

[Little about **return-to-libc attacks,**

A **“return-to-libc” attack** is a computer security attack usually starting with a buffer overflow in which a subroutine return address on a call stack is replaced by an address of a subroutine that is already present in the process’ executable memory, bypassing the NX bit feature (if present) and ridding the attacker of the need to inject their own code.

On POSIX-compliant operating systems the C standard library is commonly used to provide a standard runtime environment for programs written in the C programming language. Although the attacker could make the code return anywhere, **libc** is the most likely target, as it is almost always linked to the program, and it provides useful calls for an attacker (such as the system function used to execute shell commands.)

]

**NX (No eXecute, non-executable stack):**

The NX bit is a feature of the Memory Management Unit of some CPU (including recent enough x86). It allows to mark each memory page as being "allowed" or "disallowed" for code execution. The MMU is under control of the kernel; the kernel code decides which pages get the execution privilege and which do not.

**Usage:**

is used to prevent certain types of malicious software from taking over computers by inserting their code into another program's data storage area and running their own code from within this section; one class of such attacks is known as the buffer overflow attack.

References:

1. <http://thehackernews.com/2016/02/glibc-linux-flaw.html>
2. <https://www.elastichosts.com/blog/cve-2015-7547-glibc-vulnerability/>
3. <https://en.wikipedia.org/wiki/Stack_buffer_overflow>
4. <https://en.wikipedia.org/wiki/Call_stack>
5. <https://en.wikipedia.org/wiki/Subroutine>
6. <https://en.wikipedia.org/wiki/Getaddrinfo>
7. <http://searchsecurity.techtarget.com/definition/address-space-layout-randomization-ASLR>
8. <https://en.wikipedia.org/wiki/Address_space_layout_randomization>
9. <https://en.wikipedia.org/wiki/Call_stack#STACK-FRAME>
10. <https://en.wikipedia.org/wiki/Return-to-libc_attack>
11. <https://en.wikipedia.org/wiki/Address_space>
12. <https://en.wikipedia.org/wiki/NX_bit>
13. <http://security.stackexchange.com/questions/47807/nx-bit-does-it-protect-the-stack>