INS Module 4

* Software Flaws

→ A software flaw in network security refer to an unintended behaviour in the software components or systems that are meant to protect a network.

→ Typically a result of a vulnerability within the system.

* Quality-related terms

① Error - Refers to human mistakes in performing activities related to securing the network.

② Can occur while defining security policies, configuring firewall rules, or writing code for security protocols. These errors can potentially introduce vulnerabilities.

③ Fault :- Fault refers to an incorrect step, command or data definition within security infrastructure or software

May lead to network being in an incorrect state.

③ Failure : The situation when a network departs from its desired, secure state

Failure is often a result of exploiting of a fault or unintended changes

→ An error can cause multiple faults, these faults exist in different parts of the security infrastructure

→ However, not every fault necessarily leads to a failure. Some faults may remain dormant and not be externally observable. Only when a fault triggered or exploited, results in failure

① Non Malicious Flaws

① Buffer overflow
② Incomplete Mediation
③ Time-of-check (TOC) or Time-of-Use (TOU)

② Malicious flaws.

a) Virus, Worm, Rabbit
b) Trojan Horse, trapdoor
c) Logic Bomb, time Bomb.

Page No.
Date

→ A buffer, or data buffer is an area of physical memory storage used to hold data while its being moved from one place to another.

→ Buffer overflow: Buffer overflow occurs when data sent to a buffer exceeds storage capacity, leading to overwriting of adjacent memory.

can be caused by malformed inputs or inadequate buffer space allocation.

→ Buffer overflow Attack: In a buffer overflow attack, attackers intentionally exploit buffer overflow vulnerabilities to overwrite application memory.

By doing this they can manipulate the program's execution path, potentially causing damage to files by exposing sensitive information.

→ Coding Languages like C and C++ are more susceptible to buffer overflows because they lack built-in protections against access and overwriting of data.

* Types of Buffer overflows.

① Heap-Based Attacks: These attacks manipulate dynamically allocated memory (heap) and can lead to unauthorised access or control.

② Stack-based Attacks: These attackess target the program's call stack, attempting to overwrite return addresses or function pointers.

③. ret-to-libc Attacks: In these attacks, attackess redirect the program's control flow to pre-defined library functions, enabling malicious actions without injecting new code.

* Understanding Buffer.

→ Memory Allocation: CPU allocates virtual memory to program processes, organised into sections such as kernel, text, data, stack and heap.

→ Stack Purpose: The stack is used to manage functions, procedures, and their local variables and parameters, it supports function execution by creating stack frames.

which store return addresses, local variables and function arguments.

Stack Frames: Stack Frames are structures created to facilitate function execution and are stored in CPU registers, small data stores within the processor. The stack can be allocated at the top or bottom of the address space, depending on CPU architecture, and may grow upward or downward.

* Process Memory organisation.

A processes memory is organised into several regions

① Kernel: The top of memory holding command line parameters and environment variables.

② Text: contains program code (instructions) for execution. Its read-only, and writing to it triggers segmentation violations.

③ Data: Holds static variables, both initialised and uninitialised declared in the program.

④ Stack: Crucial for function calls and local variable allocation, passing function parameters and storing return addresses.

Stacks usually grow down in memory (LIFO) uses push, pop and a stack pointer (SP) pointing to the top.

Heap: A dynamically allocated memory area used for managing data and objects

* Stack frames in Function Calls.

① Each function call is allocated an new stack frame in memory

② A stack frame contains function parameter local variables, return address and an instruction pointer.

③ Global variables are stored in a seperate data memory area, visible to all functions

④ Return addresses determines where the control flow goes after the function's execution, facilitating recursion.

⑤ C and C++ returns a single value from a function's execution.

⑥ Debuggers can trace function calls by walking backward through the

stack frames, creating a stack trace for debugging purposes.

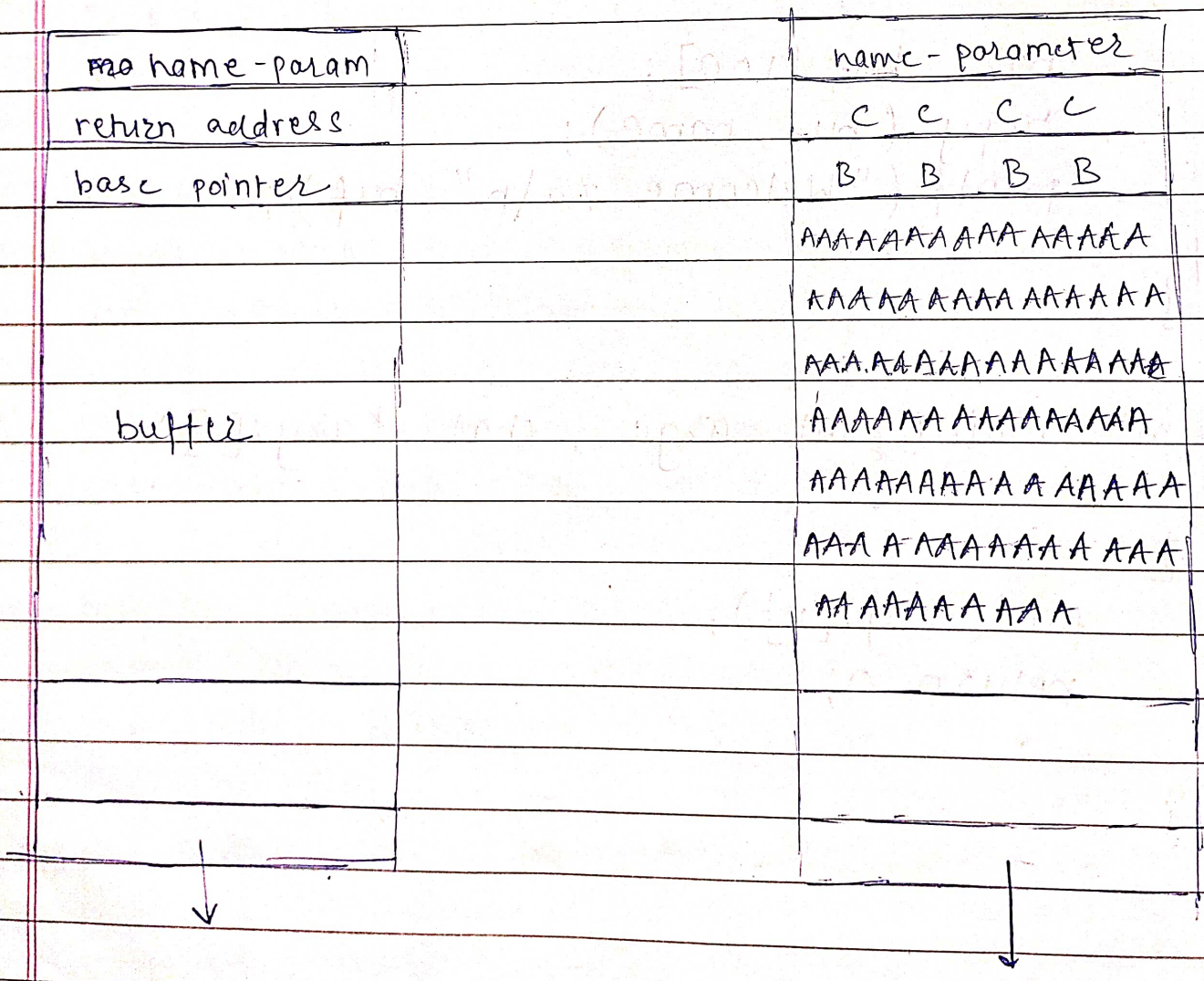
* stack overflow.

```
#include <stdio.h>
#include <string.h>

void func (char *name)
{
    char buf[100];
    strcpy (buf, name);
    printf ("Welcome %s\n", buf);
}

int main(int argc, char *argv[])
{
    func(argv[1]);
    return 0;
}
```

→ An overflow is triggered by input that is 108 characters long composed of 100 A's, 4 'B's and 4 'C's.

→ 100 'A's fill the buffer. 'B' overwrites the Extended Base Pointer (EBP) and the 'C's overwrite the return address.

→ This triggers a segmentation fault: an indication of incorrect access of memory.



Date

* Attacks on Buffer overflow.

① Stack Smashing
② Crafting Attack.

* Stack Smashing Attack.

→ In the Stack Smashing Attack, the attacker overwrites the return address with a specific address that point to malicious code, often shellcode.

→ The ideal choice for the return address is is the start address of the memory block. designated for the input string or buffer used by the function.

→ By using the start address of the buffer the attacker gains control over the data that goes into the buffer.

→ Intruder can insert executable code on the buffer and overwrite the return address with the start of this code.

→ This attack will succeed if the targeted program must have a buffer overflow vulnerability.

Difficulties while Stack Smashing

Successful attacks require predicting the correct return address location on the stack, meaning to determine the spacing or offset b/w local variables and the return address.

Fuzzing can aid in this process, and when a segmentation fault occurs, analyzing the core file can reveal the input string's memory location.

Knowing the address of malicious code is crucial. Memory may shift during program execution.

- So an NOP-sled is used, which contains (NOP) instructions which slides the CPU's execution flow to the next memory address.

- If the return address lands in the NOP-sled, it will slide along until it reaches the start of the shellcode.

- By inserting multiple NOPs before the evil code and repeating the desired return address, increases the likelihood of successfully jumping to the shellcode.

Page No.
Date

② Crafting Attack.

→ In a buffer overflow attack when payload is 108 bytes, and shellcode is 25 bytes, the remaining 83 bytes must be filled with NOP (no-operations) instructions.

→ The payload structure consists of

① NOP-sled
② SHELL CODE
③ FILLER.

→ NOP-sled is placed at the start of the code.

→ The filler is any character repeated multiple times, and will be replaced by the address towards the shell code.

→ Size of Filler should atleast be 4 bytes to contain a full memory address, and a larger NOP-sled increases the chances of successfully hitting it with the memory jump even if memory is re-allocated during runtime.

* Protection Against Buffer overflow.

① Input Validation.

a) Create a trust boundary around the application, validate and sanitize all input crossing that boundary.

b) Most buffer overflows occur due to misplaced trust in all forms of input, including files, registry, network and environment data.

② Choice of Programming Language

a) → C and C++ are popular but lack built in array boundary checking

b) Use modern languages like .Net, Java, Python.

③ Use safe Libraries.

a) Avoid using CRT libraries, instead use safer libraries like STL, or MFC.

b) Kernel patches can substitute unsafe library calls.

Page No.
Date

④ Static Analysis

a) Employ code analyzers to detect buffer overflow.

b) Note that checking for arbitary code overflows is challenging and relies on programmer expertise.

⑤ Stack - Smashing Protection using canary value.

a) Use runtime stack checking to prevent stalk smashing attacks

b) Add special values (canary values) between stalk variables and the return addresses.

× ×

* Incomplete Mediation:

→ Failure to perform 'sanity checks' on data can lead to random or carefully planned flaws in a programs operation.

→ Incomplete Mediation can make buffer overflow conditions exploitable

Examples of Incomplete Mediation.

① Failure of Input validation: Neglecting to validate and sanitize input data can lead to security vulnerabilities.

② format string vulnerability: If program doesnt properly sanitize user inputs.

③ Integer overflow: Failure to handle integers overflow can result in incomplete mediation.

→ Example of Input validation failure.

strcpy (buffer, argv[1])

Now buffer overflow occurs if len(buffer) < len(argv[1])

→ Format String Effects.

Many developers use data from un-trusted users as the format string.

Now, attackers can write format strings to cause many problems.

```
#include <stdio.h>

int main (int argc, char *argv[])
{
    if (argc > 1)
    {
        printf (argv[1]);
        return 0;
    }

}
```

→ printf function took an input string that caused it to expect two arguments to be pushed onto the stack prior to calling the function.

→ The %x specifiers enable you to read the stack, four bytes at a time

→ An invalid pointer access or unmapped address.

→ Eg. printf ("%s %s %s %s")

→ The %s conversion specifier displays memory at an address specified in the corresponding argument on the execution stack.

→ No arguments are specified.

* Viewing stack content.

→ Allows attackers to view portions of the stack & memory.

→ By using a formal string like

printf ('%.08x, %.08x, %.08x, %.08x, %.08x \n")

An attacker can instruct the printf function to retrieve and display 5 parameters from the stack as 8-digit padded hexa decimal numbers.

→ This can be exploited to view offsets

* Overflowing Memory.

→ Formatted output functions can be dangerous because many programmers may not be aware of their capabilities

→ The %.n conversion specifier is designed to help align formatted output strings

Page No.
Date

Example

```
printf ("hello %n\n", (int *)&i);
```

Would result in variable i being assigned the value 5.

%n specifier returns the count of characters.

* Integer overflow.

→ An integer overflow occurs when an integer exceeds it's maximum value or goes below the minimum value.

→ Types of overflow.

① Signed overflow: Happens when a value carries over into the sign bit, affecting the sign of the integer.

② Unsigned overflow: occurs when the underlying representation can no longer represent a value, typically reaching the maximum value.

→ Signed range goes from $-2^{(n-1)}$ to $2^{(n-1)}-1$, where 'n' represents the number of bits used to represent the signed type.

→ Unsigned integer have values ranging from zero to max calculated as $2^n - 1$.

---

* Race condition.

→ A 'race' refers to the race between an attacker and the next stage of the process. Race condition can arise when security-critical processes are divided into stages.

→ Attacker exploits Races: Attackers can exploit race conditions by making changes b/w stages of a process. Typically b/w the stage that grants authorisation and the stage that transfers ownership.

→ Example UNIX 'mkdir' command.

Page No.
Date

* 'mkdir' Attack.

→ Intruder allocates space for a new directory, and during this time, establishes a link from a file, to the allocated space before the directory's ownership is transferred to the intruder

→ Attacker's timing is crucial in these scenarios.

→ Race conditions are pretty common but harder to exploit than buffer overflows.

→ To prevent race conditions, security-critical processes should ideally be made atomic, happening all at once rather than in stages.

× ×

* Software-based Attacks

① Salami Attack
② Linearization Attack.

* Salami Attack.

→ A 'Salami Attack' is a cybercrime tactic often employed for financial crimes where criminals steal money or resources gradually, taking small amounts at a time from financial accounts.

The incremental theft are challenging for victims to detect.

→ Typically executed by insiders, ie. employees having access to financial systems.

→ In a Salami Attack, individual transactions are often minuscule, making them virtually unnoticeable. Victims may not immediately recognize the problem.

→ Example - A programmer working for a bank calculates interest on customer accounts and siphons off tiny fractions of a cent from the calculations. No individual customer notices the miniscule deductions.

→ However, the perpetrator accumulates significant sums.

Page No.
Date

* Identifying a Salami Attack.

① System security Enhancement: Corporations must prioritize enhancing system security to identify and patch any vulnerabilities.

② Customer Reporting: Banks should actively advise their customers to promptly report any unauthorised money deductions, regardless of amount.

③ Data Storage Awareness: Customers are encouraged to avoid storing bank details online. However this may be challenging due to the nature of banking transactions.

---

* Linearisation Attack

① Program checks serial numbers character by character (eg. 'S123N456') for efficiency.

② Attackes Exploitation :- Attackers can exploit this by noting that the corect number takes longer to check than an incorrect one.

③ Intruder begins with guesses for the first character, in this case "s". When the system takes longer to respond. Bingo they have found the correct character. This process is repeated for all the characters.

Also known as Iterative attack.

④. Intruder can recover the character one at a time, exploiting the delay in program's response to confirm each character.

⑤ Similar to the trial and error in lock picking.

⑥ By brute-force approach, an 8-char serial number takes 256^8 to 2^{64} possible scenarios, so the correct one can be found in $2^{64}-1$ tries.

x x

Page No.
Date

* Malwares (Malicious Softwares)

→ Malware, short for malicious software, encompasses any harmful program or code designed to infiltrate or harm the computer without owner's consent.

→ Sources of Malware: Malware primarily gains access to devices through Internet or email. Some malware can be unknowingly installed when users visit infected websites or download content from web.

→ Purpose of Malware: Malware has various purposes like.

a) Subject users to unwanted advertising

b) Launching Distributed Denial of Services attacks on other services.

c) Spreading Spam.

d) Tracking user activity.

e) Spreading FUD (Fear, Uncertainity, Doubt)

Symptoms of Malware.

① Sluggish Performance: A noticeable slowdown in your computer, or internet connection speed.

② Browser Issues: Frequent Browser crashes or complete browser failure.

③ Popup Ads: Sudden Appearance of unwanted popup advertisements on your screens.

④ Frequent System crashes: Unexplained and recurrent system or program crashes.

⑤ Disk Space Reduction: If available disk space decreases unexpectedly.

---

* Types of Malware

① Virus: Self-Replicating Malware that attaches to other programs modifying them and spreading when executed.

② Worm: A Standalone program which actively spreads through networks.

③ Trojan Horse: Deceptive Malware that appears useful but grants unauthorised access to attackers, enabling data theft.

Page No.
Date

④ Trapdoor / Backdoor: Provides unauthorised access to a computer potentially by priviledged users.

⑤ Rabbit: Self-replicates excessively to consume resources on the computer.

⑥ Time Bomb: Malware with malicious impact triggered by specific conditions.

⑦ Spyware: Secretly observes user activities and steals information

⑧ Adware: Unwanted software that displays intrusive advertisments, often disguising ~~as~~ ~~for~~ as a legitimate software.

⑨ Ransomware: Locks the user out and demands ransom for access.

⑩ Keylogger: Records keyboard keystrokes, capturing sensitive information.

* Malware Detection.

→ Malware detection refers to the process of detecting the presence of a malware on a host system.

Mainly of 3 types.

① Signature Detection.

→ Malware is identified by unique code signatures. When a file reaches a computer, a malware scanner collects the code and sends it to a cloud database.

→ Signature components: Signatures can be bit strings in executable files, and may also use wildcards, hash value and more.

→ Example: W32/Beast virus signature has a signature like "83EB 0274 EB0B 740A 81EB 0301 0000".

→ However only applicable to known malware.

Page No.
Date

Advantages

① Effective on "ordinary" malware.

② Minimal burden for users / administrators

Disadvantages

① Signature file can be large.

② Not effective on new malware.

② Change Detection:

→ Detecting Virus Presence: Viruses need a location to reside. Detecting changes in files can be an indicator of possible infection.

→ Method: To detect changes, files are hashed and hash values are securely stored. Periodic re-computation of hashes allows for comparison. If hash changes, files might be infected.

→ Advantages

① Virtually no false negatives

② Can detect previously unknown malware.

→ Disadvantages -

① Many files change often.

② Many false alarms.

③ Anomaly Detection.

→ Anomaly detection involves monitoring a system for any unusual or potentially malicious behaviour that doesnt align with the 'normal' system.

→ Example of anomaly: Unexpected change in files, system malfunctions, unanticipated network activities.

→ Advantages

① Chance of detecting unknown malware.

→ Disadvantages.

① No proven track record.

② Can make abnormal look normal.

Page No.
Date

* Defense against Malware

① Regular Updates and Patches: Keeping software updated at all times.

② Anti-Malware and firewall Software: Install and run robust anti-malware and firewall software that can detect, quarantine, and remove various softwares which are malicious.

③ User Awareness: Educate users on how to avoid malware entering their systems

④ Regular Data Backups: Perform data backups to prevent loss of data.

⑤ Firewall and IPS: Implement firewalls and intrusion detection systems to monitor and control network traffic, blocking suspicious activity.