UNIT 3
Exploitation

Exploitation

In cybersecurity, an exploit is a piece of code, a sequence of commands, or a specific technique that takes advantage of a security flaw or vulnerability in a software or hardware system to perform unauthorized actions, such as gaining illegal access, stealing data, installing malware, or causing denial-of-service (DoS) attacks

How Exploitation Works

1.Vulnerability identification:

Cybercriminals or researchers find a security flaw (vulnerability) in a system.

2.Exploit development:

They then create a specific exploit, which is the tool or method designed to "strike" or exploit that particular vulnerability.

3.Attack execution:

The exploit is delivered to the vulnerable system, which then carries out the intended malicious action.

Shell code

- Shell code is part of the payload in the exploitation of a software vulnerability to take control of or exploit a compromised machine.
- The word *shellcode* literally refers to code that starts a command shell -- an instance of a command-line interpreter, such as the shell */bin/sh* on Linux or *cmd.exe* on Windows.

- ➢The term now also embraces any bytecode that can be executed once the code is injected into a running application, even if it doesn't spawn a shell.
- ➢Common shell code objectives include installing a rootkit or Trojan horse, stopping antimalware programs, obtaining sensitive data or downloading files to further compromise the targeted device.

How does a shell code exploit work?

Shell codes are injected into computer memory. After the exploit code causes what would normally be a critical error in the targeted program, the program jumps to the shell code and is tricked into executing the attacker's commands -- all with the privileges of the process being exploited.

A shell code exploit consists of two major components:

1)The **exploitation technique's** objective is to insert the shell code and divert the execution path of the vulnerable program to the shell code so that it can run the code in the payload.

2)The **payload** is the component that executes the attacker's malicious code.

Types of shell code exploits

1)Shell code can be either local or remote:

Local shell code is used when an attacker has physical access to a machine.

2)Remote shell code is used to target a vulnerable process running on another machine to gain access to it across a network.

➢Single buffers often have limited space where attackers can inject their entire payload into a remote target process. To overcome this restriction, hackers use various techniques, including the following:

- **Staged shell code.** Shell code can be executed in stages. The role of the stage one shell code is to download a more complex and larger stage two shell code into the process memory and execute it.
- **Download and execute.** This type of shell code instructs the device to download the attacker's malicious file from the internet and execute it. Not only does this enable the shell code to be small, but it does not need to spawn a new process on the target system.

The most common programming errors used to insert shell code exploits are buffer overflows. Buffers are temporary areas for data storage. A buffer overflow occurs when more data is put into a buffer than it has capacity for. There are two main types of buffer overflows:

1. A **stack-based buffer overflow attack** occurs when an application's stack -- the area that stores requests -- is exploited.
2. A **heap-based buffer overflow attack** occurs when an application's memory space is exploited.

How to protect against shell code exploits

- To protect against exploits that inject shell code into vulnerable programs, enterprises need a multilayered security strategy.
- Hackers use automated scanners to look for applications using known vulnerable code, so enterprise firewalls and device controls need to be set to stop unwanted and known malicious connections.
- Monitoring controls also need to be in place to spot and quarantine unusual activity on the network using static and behavioral AI to stop the attack before it can do any lasting damage.

What is an Integer Overflow?

An Integer Overflow occurs when the result of an arithmetic operation exceeds the maximum value that can be stored in a given integer type. When this happens, the value wraps around to the minimum value, leading to unexpected and potentially dangerous outcomes.

How does Integer Overflow Work?

Integer overflow works by exploiting the fixed-size nature of integer data types in most programming languages. When an arithmetic operation produces a result that exceeds the maximum or minimum value that can be stored in the allocated memory space, the value wraps around to the opposite end of the range. For instance, adding 1 to the maximum value of a 32-bit unsigned integer (4,294,967,295) results in 0.

What are the Potential Risks of Integer Overflow?

Understanding the potential risks of integer overflow is crucial for maintaining robust cybersecurity. Here are some of the key risks associated with this vulnerability:

System Instability: Integer overflow can cause unexpected program behavior, leading to system crashes or application failures.

Unauthorized Access: Exploiting integer overflow can result in buffer overflow vulnerabilities, allowing attackers to gain unauthorized access and execute arbitrary code.

Data Corruption: Integer overflow can lead to data corruption by causing wraparounds or undefined behavior, resulting in incorrect values being stored or used in computations.

•**Financial Losses:** Vulnerabilities caused by integer overflow can disrupt business operations, leading to significant financial losses due to system downtime or compromised data integrity.

•**Reputation Damage:** Companies affected by integer overflow vulnerabilities may suffer reputational harm, losing customer trust and facing potential regulatory scrutiny.

Example:

step by step into how **integer overflow vulnerabilities can occur in Banking** and how attackers exploit them.

Banking System – Fake Deposit via Integer Overflow

Scenario:

A bank’s old core software stores account balances in a **32-bit signed integer**.

1) Initial Balance:

Customer has ₹1,000.

Max value the system can hold: **2,147,483,647** (≈ ₹2.1 billion).

2) Attacker’s Action:

The attacker requests a deposit of ₹2,147,483,000 (a value close to the integer maximum).

3)What Happens in Code:

```
int balance = 1000;
int deposit = 2147483000;
balance = balance + deposit; // causes
overflow
```

•Actual math result should be: 2147484000.

•But because of **overflow**, the stored balance becomes **negative or a very small number**.

4) Exploit:

•The system mistakenly treats the negative value as a **huge positive balance** (depending on implementation).

•The attacker's account now shows **billions of rupees**.

5) Impact:

•The attacker can withdraw or transfer funds that don't actually exist.

•This is a **direct financial fraud enabled by integer overflow**.

prevention strategies for Banking against **integer overflow vulnerabilities:**

1) Use Safe Data Types

•Don’t use 32-bit int for balances.

•Use **64-bit integers** (long long, bigint) or **arbitrary-precision libraries** (e.g., BigDecimal in Java, Decimal in Python).

2) Input Validation

•Reject deposits, withdrawals, or transfers that exceed reasonable thresholds (e.g., ₹10 billion).

•Enforce **business rules** (e.g., customers can’t deposit more than daily transaction limits).

•**Code Reviews** focusing on arithmetic operations.

•**Strict Business Rules** embedded in logic (e.g., no “negative bills”).

stack based buffer overflow

A **buffer** is a temporary area for data storage. When more data (than was originally allocated to be stored) gets placed by a program or system process, the extra data overflows. It causes some of that data to leak out into other buffers, which can corrupt or overwrite whatever data they were holding.

In a **buffer-overflow attack,** the extra data sometimes holds specific instructions for actions intended by a hacker or malicious user; for example, the data could trigger a response that damages files, changes data or unveils private information.

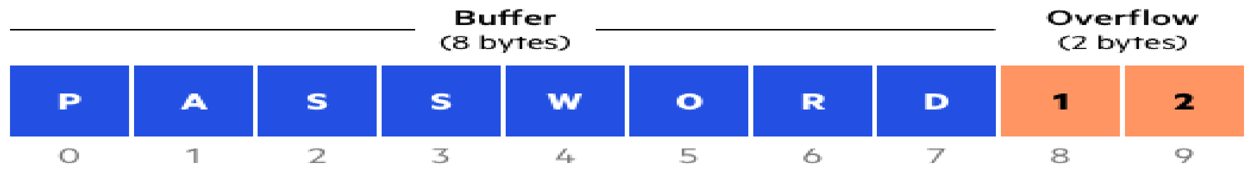
Attacker would use a buffer-overflow exploit to take advantage of a program that is waiting on a user's input. There are two types of buffer overflows:

1)stack-based

2)heap-based.

Heap-based, which are difficult to execute and the least common of the two, attack an application by flooding the memory space reserved for a program.

Stack-based buffer overflows, which are more common among attackers, exploit applications and programs by using what is known as a stack memory space used to store user input.



Buffer overflow example

How to Prevent Buffer Overflows

1.**Programming Practices**

- Use safer functions (strncpy, fgets, snprintf instead of strcpy, gets, sprintf).
- Input validation and bounds checking.

2.Compiler / OS Protections

- **Stack Canaries** (random values placed to detect corruption).
- **DEP / NX bit** (disallow execution of stack memory).
- **ASLR (Address Space Layout Randomization)** (makes memory addresses unpredictable).
- **Safe libraries** (Fortify source, checked libraries).

Example: Stack-Based Buffer Overflow in E-Commerce Systems

•E-commerce systems handle **user accounts, payments, and order processing**.

•Many older e-commerce platforms (especially in the early 2000s) were written in **C/C++** for speed and performance.

•If input validation is weak, stack-based buffer overflows can occur in areas like:

•**User login forms** (username/password fields)

•**Promo code / coupon validation modules**

•**Payment gateway connectors** (card details, transaction IDs)

Suppose an e-commerce system has a function to apply **promo codes** during checkout:

```
void applyPromoCode(char *input_code) {
    char promo_code[32];  // buffer for promo code

    // ❌ Unsafe: No length check
    strcpy(promo_code, input_code);

    if (validatePromo(promo_code)) {
        printf("Promo applied successfully!\n");
    } else {
        printf("Invalid promo code.\n");
    }
}
```

•**Problem:**

If an attacker enters a **promo code longer than 32 characters**, the buffer promo_code overflows.

•This can **overwrite the return address** on the stack and hijack control flow.

Impact on CIA Triad

Confidentiality: Customer data (credit cards, addresses) exposed.

Integrity: Product prices or order quantities manipulated.

Availability: Checkout/payment service crash, causing loss of sales.

How the Attack Happens

1.Crafted Input

- •The attacker submits a promo code like:

  "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

  AAAAAAAAA[malicious code]"

- •This fills up the buffer and overwrites the return address.

2.Control Hijack

- •When the function finishes, instead of returning normally, it jumps to the attacker’s injected **malicious code**.

3.What Attacker Can Do

- •**Bypass payment checks** → attacker sets order total to 0.00.
- •**Escalate privileges** → log in as admin without credentials.
- •**Steal sensitive data** → credit card details, session tokens, customer records.

Prevention in E-Commerce Systems

1)Secure Coding

•Replace unsafe functions (strcpy, gets, sprintf) with safe versions (strncpy, fgets, snprintf).

•Validate input length for all user-submitted fields (promo codes, usernames, addresses).

2)Operational Measures

•Regular **penetration testing** on checkout and payment systems.

•Use **modern frameworks** (Java, .NET, Python, etc.) instead of raw C for handling user inputs.

•Apply **PCI DSS compliance checks** for secure payment handling.

Format string vulnerabilities

•The Format string attack is an attack through which the user input is run as a command by the web application.

•Through this vulnerability, the attacker might run commands to extract information about the server, view the source code of the application.

•The attacker can crash the application by executing malicious code to create a segmentation fault and many more.

The string can be divided into 3 parts :

1. Format function - This includes printf, fprintf etc.
2. Format string - This is the argument for format function
3. Format string parameter - This defines the type of conversion.

•Format string vulnerability is possible because the application failed to sufficiently sanitize the user input. Applications that use format function improperly are most vulnerable to this attack.

•The format function is used to interpret formatting characters. Common formatting characters include %x (used to read stack data), %s (read process memory) and more.

The attacker can perform this attack through the following ways:

1. **Enumerate process stack:** The attacker uses %x and %p to view the stack organization of the application. Using this method, the attacker can leak sensitive information about the server.
2. **Control execution flow:** The attacker uses %n to overwrite the pointer variables used by the application. When the application calls these pointers, the pointer will send malicious code to the application.
3. **Denial of service:** The attacker uses %x followed by %x to crash the application and the server.

Example: Format String Vulnerability in Banking Login & Transaction Systems

- A legacy **online banking system** (written in C) handles:
    - **User login** (username/password check).
    - **Transaction logs** (keeping record of money transfers).
- To help with auditing, the system **logs user inputs directly** into the output stream.
- Unfortunately, it **uses user input as the format string** — a critical mistake.

Vulnerable Code:

Login Module

```
void login(char *username, char *password) {
    char stored_password[20] = "Bank@123";


    if (strcmp(password, stored_password) == 0) {
        printf("Welcome %s!\n", username);
    } else {
        // ❌ Vulnerable
        printf(username);   // user input directly as format string
        printf(" - login failed!\n");
    }}
```

Transaction Logging

```
void logTransaction(char *account, int amount)
{
    // ❌ Vulnerable
    printf(account);
    printf(" transferred %d dollars\n", amount);
}
```

How the Attack Works

Step 1: Exploiting Login

- Normal login failure:
    - Username = Alice
    - Output: Alice - login failed!
- Malicious login attempt:
    - Username = %x %x %x %x
    - Output: 4f2a7c 89ac12 fff123 ... - login failed!
    - **Result:** Attacker dumps sensitive **stack memory**.

Step 2: Authentication Bypass

•By using %n, the attacker can write arbitrary values to memory.

Example malicious input:

%x %x %x %n

•If the attacker aligns memory correctly, they can **overwrite the authentication flag** → system thinks login succeeded.

Step 3: Exploiting Transactions

•Malicious account input during transfer: %x%x%x%n

•This lets the attacker:

- •Leak **other customers' account numbers**.
- •Overwrite memory to **change transaction amount** (e.g., transfer $1 → $10,000).
- •Redirect funds to attacker-controlled accounts.

Prevention

1) **Safe Printing**

•Always use a fixed format string:

printf("%s", username); // ✅ safe

•Never trust user input as the format string.

2) **Compiler Protections**

•Enable warnings: -Wformat -Wformat-security.

3) **Secure Development in Banking**

•Use **logging libraries** instead of raw printf.

•Replace legacy C/C++ modules with memory-safe languages (Java, Python, Go).

•Perform **regular penetration testing** on login and transaction modules.

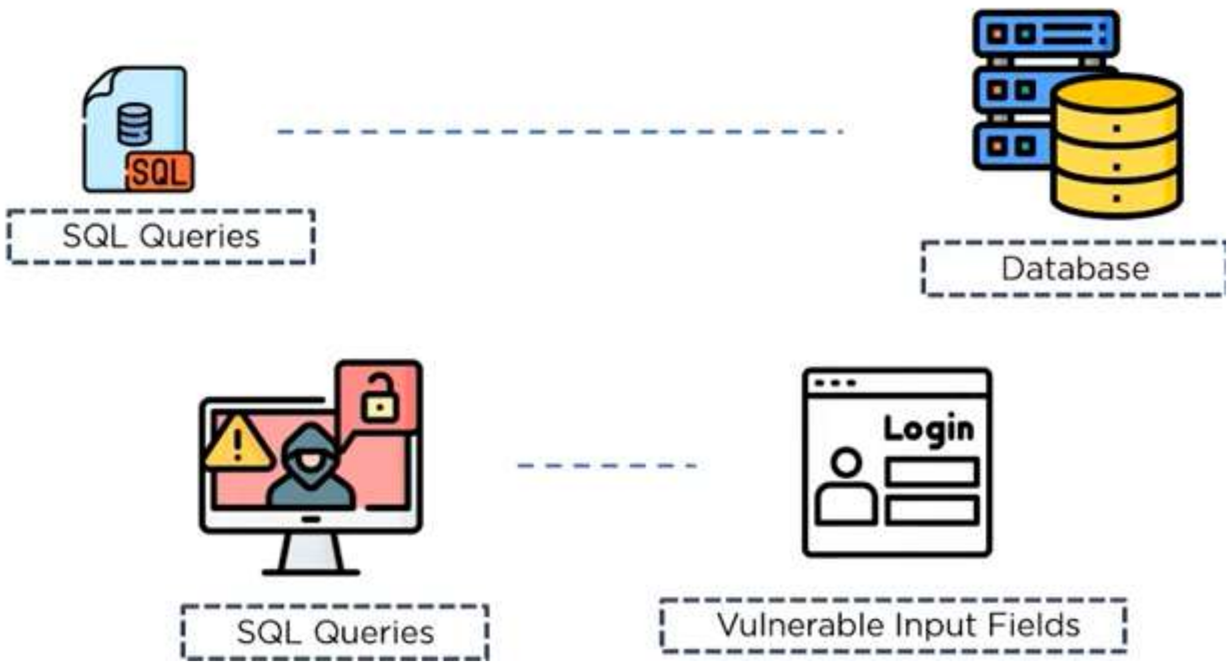
SQL injection

SQL injection is a code injection technique that might destroy your database.
SQL injection is one of the most common web hacking techniques.
SQL injection is the placement of malicious code in SQL statements, via web page input.

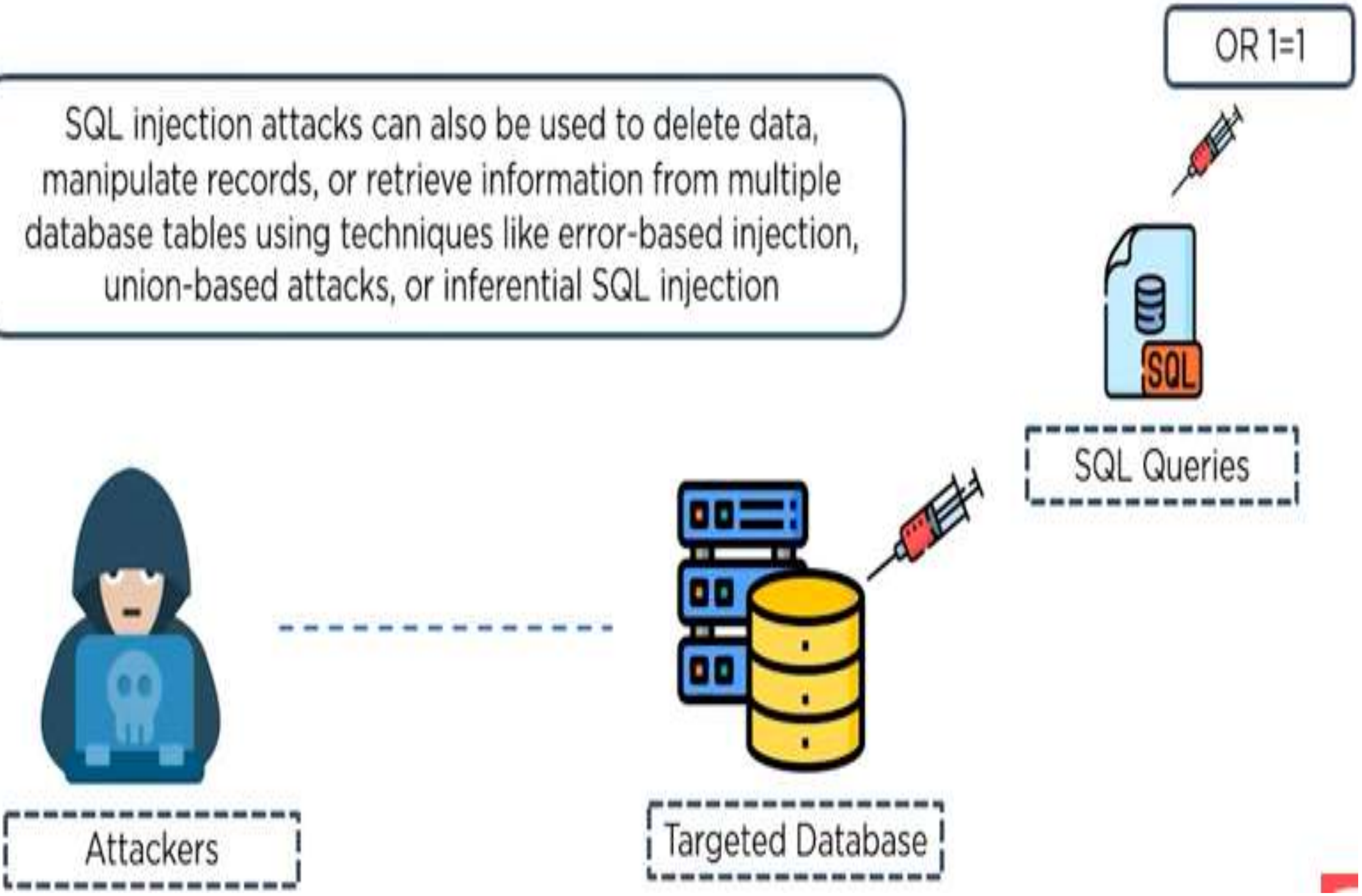
SQL injection is a technique used by attackers to manipulate a web application's database queries

By injecting malicious SQL code into input fields, attackers can execute unauthorized commands and potentially gain access to sensitive information

How Does a SQL Injection Attack Work?



SQL injection attacks can also be used to delete data, manipulate records, or retrieve information from multiple database tables using techniques like error-based injection, union-based attacks, or inferential SQL injection

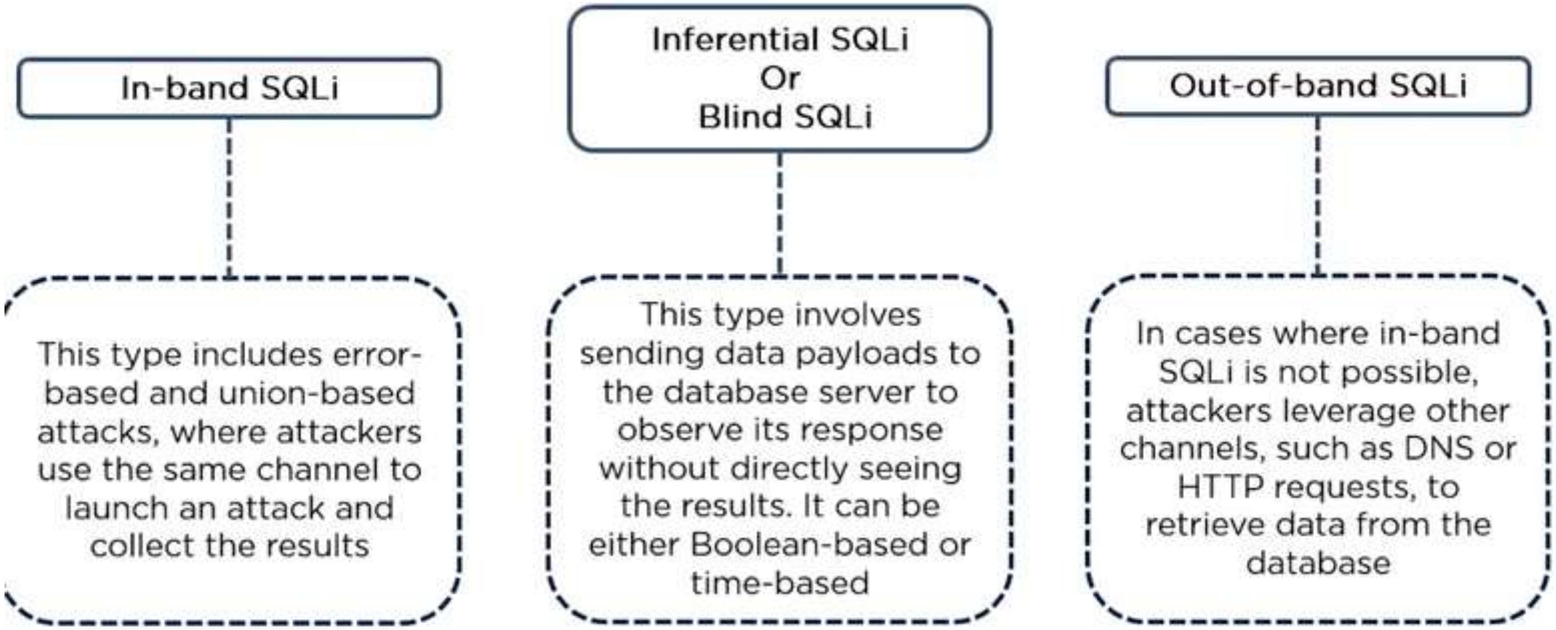


Website

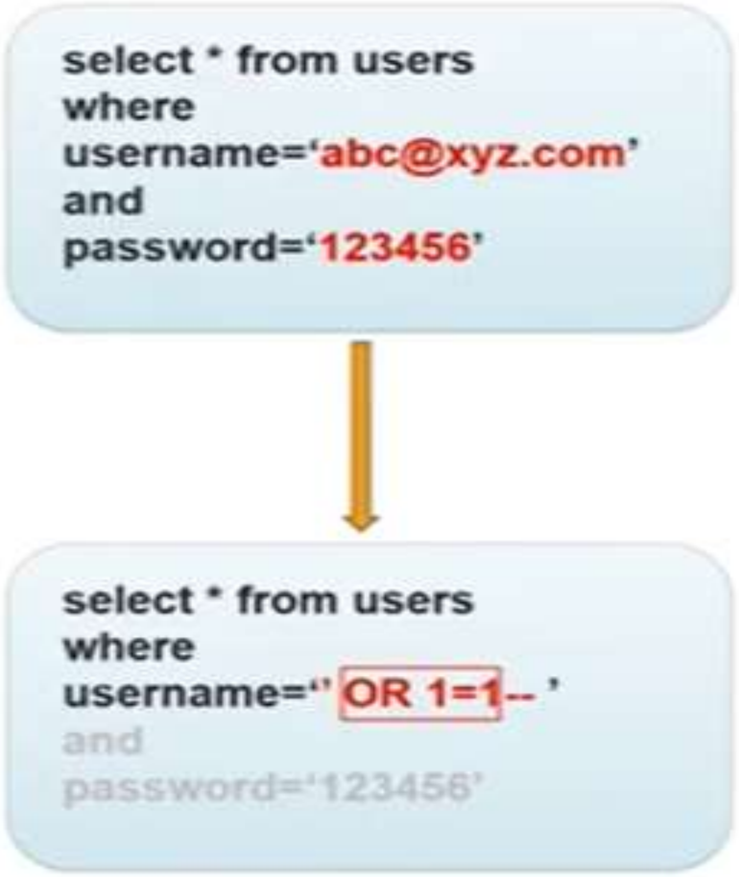
SQL Database

```
SELECT * FROM products WHERE name LIKE '%entered_keyword%';

' OR '1'='1'; --

SELECT * FROM products WHERE name LIKE '%' OR '1'='1'; --%';
```

Types of SQL Injection Attacks


In-band SQLi
This type includes error-based and union-based attacks, where attackers use the same channel to launch an attack and collect the results
Inferential SQLi Or Blind SQLi
This type involves sending data payloads to the database server to observe its response without directly seeing the results. It can be either Boolean-based or time-based
Out-of-band SQLi
In cases where in-band SQLi is not possible, attackers leverage other channels, such as DNS or HTTP requests, to retrieve data from the database

HOW SQL INJECTION WORKS?



```
select * from users
where
username='abc@xyz.com'
and
password='123456'
```

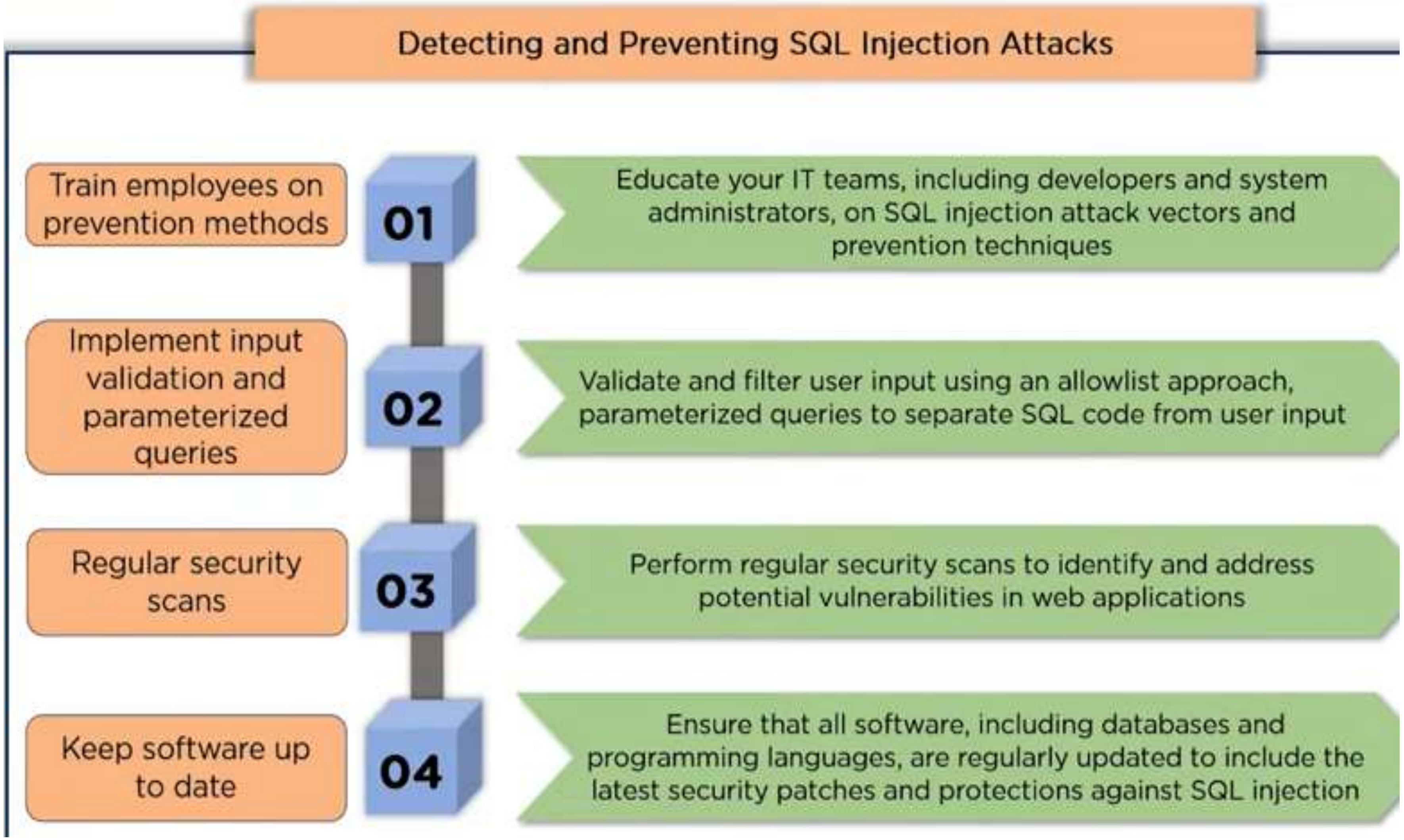
```
select * from users
where
username='' OR 1=1-- '
and
password='123456'
```

HOW TO USE SQL INJECTION?

GET Data is sent in the URL of the request.

```
localhost/index.php?username=admin&password=admin
```

POST Data is sent in the request body of the request.



Detecting and Preventing SQL Injection Attacks

Train employees on prevention methods	01	Educate your IT teams, including developers and system administrators, on SQL injection attack vectors and prevention techniques
Implement input validation and parameterized queries	02	Validate and filter user input using an allowlist approach, parameterized queries to separate SQL code from user input
Regular security scans	03	Perform regular security scans to identify and address potential vulnerabilities in web applications
Keep software up to date	04	Ensure that all software, including databases and programming languages, are regularly updated to include the latest security patches and protections against SQL injection

Impact of SQL Injection Attacks

- Unauthorized access to sensitive information and resources
- Potential data breaches and exposure of confidential data
- Data manipulation or deletion
- Network infiltration and system compromise
- Loss of customer trust and decreased revenue
- Reputational damage and long-term consequences for the business

Race Condition

➢A race condition is a situation that may occur inside a critical section if the critical section is not properly protected.

➢This happens when the result of multiple thread execution in a critical section differs according to the order in which the threads execute.

➢Race conditions in critical sections can be avoided if the critical section is treated as an atomic instruction. Also, proper thread synchronization using locks or atomic variables can prevent race conditions.

What are the types of race conditions?

There are a few types of race conditions. Two categories that define the impact of the race condition on a system are referred to as critical and noncritical:

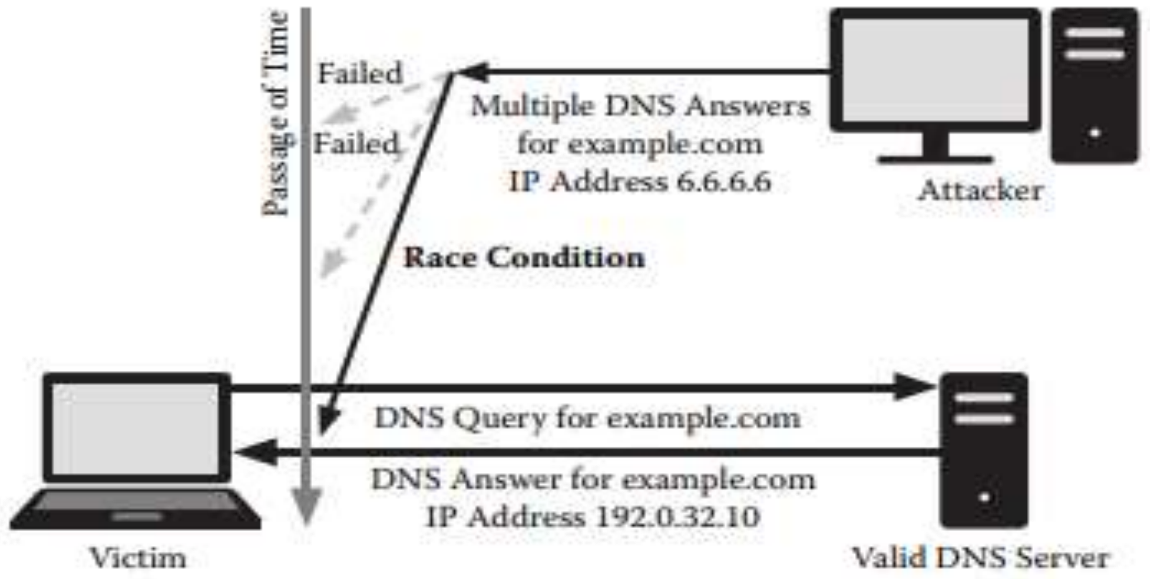

Passage of Time
Failed
Failed
Multiple DNS Answers for example.com IP Address 6.6.6.6
Attacker
Race Condition
DNS Query for example.com
DNS Answer for example.com IP Address 192.0.32.10
Victim
Valid DNS Server

Exhibit 3-18 A domain name service (DNS) answering a race condition over the User Diagram Protocol (UDP).

1. A **critical** race condition will cause the end state of the device, system or program to change. For example, if flipping two light switches connected to a common light at the same time blows the circuit, it is considered a critical race condition. In software, a critical race condition is when a situation results in a bug with unpredictable or undefined behavior.
2. A **noncritical** race condition does not directly affect the end state of the system, device or program. In the light example, if the light is off and flipping both switches simultaneously turns the light on and has the same effect as flipping one switch, then it is a noncritical race condition. In software, a noncritical race condition does not result in a bug.

How to identify race conditions

➢Detecting and identifying race conditions is considered difficult. They are a semantic problem that can arise from many possible flaws in code. It's best to design code in a way that prevents these problems from the start.

➢Race conditions are sometimes produced by data races, which occur when two threads concurrently target the same memory location and at least one is a write operation.

➢Data races are easier to detect than race conditions because specific conditions are required for them to occur. Tools, such as the Go Project's Data Race Detector, monitor for data race situations. Race conditions are more closely tied to application semantics and pose broader problems.

How do you prevent race conditions?

Two ways programmers can prevent race conditions in operating systems and other software include:

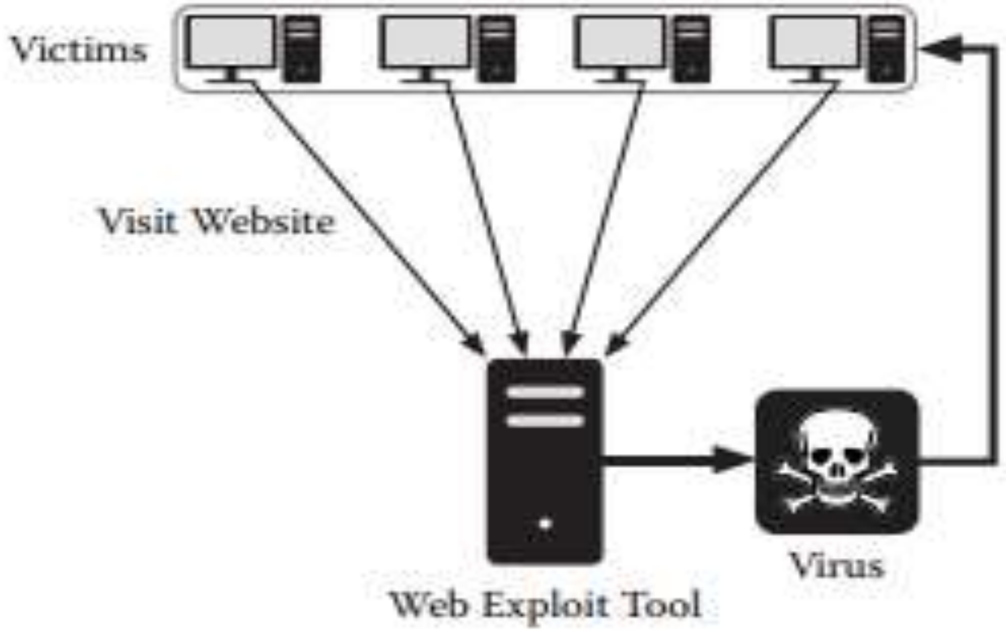
1. **Avoid shared states.** This means reviewing code to ensure when shared resources are part of a system or process, atomic operations are in place that run independently of other processes and locking is used to enforce the atomic operation of critical sections of code. Immutable objects can also be used that cannot be altered once created.
2. **Use thread synchronization.** Here, a given part of the program can only execute one thread at a time.

brute force and dictionary attack

Feature	Brute Force	Dictionary Attack
Approach	Tries all possible combinations	Tries common/likely passwords from a list
Speed	Slow (time-consuming)	Faster
Success Rate	Guaranteed (eventually)	Limited (depends on password strength)
Best Against	Any password (given enough time)	Weak, common, or reused passwords
Resource Usage	High (CPU/GPU intensive)	Low to medium

Web Exploit Tools

- Web exploit tools (or exploit kits) give attackers the ability to execute arbitrary code using vulnerabilities or social engineering
- To attract visitors to the malicious websites, attackers often compromise other servers to append IFrame tags, which direct visitors to attackers’ Web exploit tools.
- Most Web exploit tools are so simple that the operator needs only to supply the executable virus for installation. Web exploit tools usually handle hiding, exploitation, and statistics automatically



shows how attackers use Web exploit tools to install malicious code on victims’ computers.

Features for Hiding:

- ➢ Many exploit tools hide exploits through encoding, obfuscation, or redirection. Exploit tools attempt to hide exploits to prevent both detection and analysis.
- ➢Exploit tools also use JavaScript or HTTP headers to profile the client and avoid sending content unless the client is vulnerable.
- ➢Exploit tools often try to detect multiple vulnerabilities to determine which will be effective and if the tool should attempt multiple attacks against a website visitor.

- HTTP headers like user-agent and browser variables like navigator or app reveal information that gives attackers the necessary information.
- If the exploit tool determines that the client is not vulnerable, it may redirect him or her to a benign URL or display an empty page.
- Attackers can also configure tools to check the language of the victim’s browser and the victim’s geographic location, or if the victims arrived at the exploit tool through a valid source.

Commercial Web Exploit Tools and Services:

➢ There are a large variety of commercial Web exploit tools and services available to install malicious code on victims' computers.

➢Pay-per-install services allow actors to buy and sell installations, which is the easiest way for a customer to install malicious code. Some examples of pay-per-install services include IFrameDollars and Loads.

➢Pay-per-traffic services allow attackers to attract a large number of victims to their Web exploit tools. Attackers can then either install their own viruses or sell installs via the pay-per-install model.

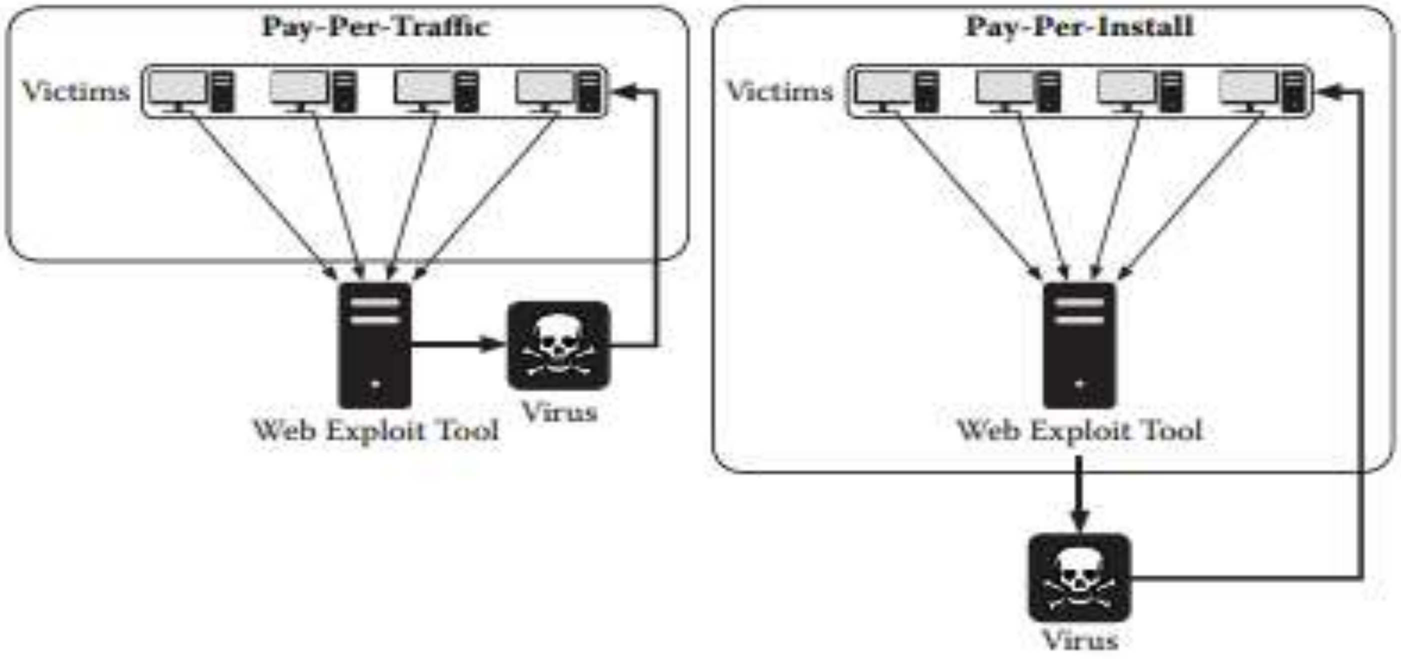


Exhibit 3-22 Pay-per-traffic and pay-per-install commercial markets.

Proliferation of Web Exploit Tools Despite Protections:

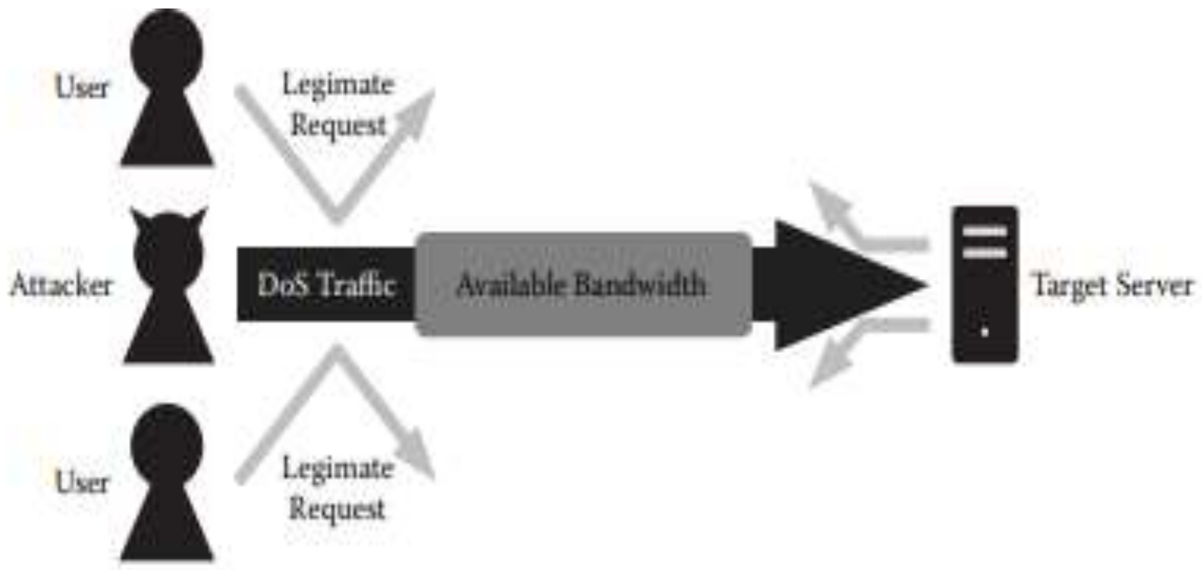
Some Web exploit tools contain protections to prevent copying and modification. The protections include the following:

- Source code obfuscation
- Per-domain licenses that check the local system before running
- Network functionality to confirm validity (license checks)
- An end-user license agreement (EULA)

Attackers using these tools collect victim trends and statistics that allow them to focus their efforts to be successful in the future. The division of traffic exploits and installs will likely continue as each area improves.

DoS Conditions

- DoS is a general term to describe a lack of access to a service. The lack of access can occur for many reasons and at different points between the client and server.
- Points subjected to a DoS condition are network segments, network devices, the server, and the application hosting the service itself.
- The conditions necessary to cause a DoS at each of these points differ, but all result in a disruption in activity between the client and server.
- A network can only send and receive a certain amount of data at one time



xhibit 3-24 A bandwidth-consuming denial of service (DoS) attack.

- One form of DoS occurs when an attacker sends so much traffic to the target that it consumes all of the available bandwidth. In this situation, most requests cannot make it to their intended destinations, which results in a DoS for legitimate clients.
- To consume the available bandwidth of a target, an attacker uses a technique known as flooding. Flooding describes the overwhelming traffic used to saturate network communications.
- Attackers use communication protocols such as User Datagram Protocol (UDP), Internet Control Message Protocol (ICMP), and Transmission Control Protocol (TCP) to inundate the target with network traffic

- DoS detection requires monitoring of network traffic patterns and the health of devices.
- Intrusion detection or prevention systems, Netflows, and other network traffic logs can provide an indication of DoS conditions in the event of an increase in network activity or alerts.
- Monitoring the health of devices can also detect if a DoS is underway.
- If a system's available resources, whether they are memory, CPU utilization, and/or another resource, reach a critical level of utilization, then the cause of such exhaustion needs mitigation to avoid a DoS condition

Cross-Site Scripting (XSS)

Social Engineering

WarXing

DNS amplification

UNIT 4
Malicious code

Malicious code is any software, script, or piece of data intentionally created to perform harmful, unauthorized, or deceptive actions on a computer system, network, or user device. Its purpose is usually to damage, steal, disrupt, or gain control — often without the owner's informed consent.

Common types (high-level)

Virus: attaches to host files and may spread when those files are executed.

Worm: self-replicates and spreads across networks without needing a host file.

Trojan: disguises itself as legitimate software to trick users into running it.

Ransomware: encrypts or blocks access to data and demands payment to restore it.

Spyware: collects information about users or systems secretly.

Rootkit: hides the presence of other malicious components and provides persistent control.

Adware (malicious variants): forces unwanted ads or tracking for profit or spying.

Self-Replicating malicious code

Worms

- Worms are similar to a virus but it does not modify the program. A computer worm is a type of malware that can replicate itself and spread across computers and networks without needing a host file or any user interaction.
- It is self-contained, meaning it doesn’t need to attach to other programs or files to function. Worms can be controlled by remote. The main objective of worms is to eat the system's resources .

virus :

A computer virus is a type of malicious software (malware) designed to attach itself to a legitimate file or program in order to spread from one system to another. Much like a biological virus, it needs a host to survive and replicate.

Basis of Comparison	Worms	Viruses
Definition	A Worm is a form of malware that replicates itself and can spread to different computers via a Network.	A Virus is a malicious executable code attached to another executable file that can be harmless or can modify or delete data.
Objective	The main objective of worms is to eat the system's resources. It consumes system resources such as memory and bandwidth and makes the system slow in speed to such an extent that it stops responding.	The main objective of viruses is to modify the information.
Host	It doesn't need a host to replicate from one computer to another.	It requires a host is needed for spreading.

Harmful	It is less harmful as compared.	It is more harmful.
Detection and Protection	They can be detected and removed by the Antivirus and firewall.	Antivirus software is used for protection against viruses.
Controlled by	They can be controlled by remote.	They can't be controlled by remote.
Execution	They are executed via weaknesses in the system.	They are executed via executable files.
Comes from	Worms generally come from the downloaded files or through a network connection.	They generally come from shared or downloaded files.

Symptoms	1. Hampering computer performance by slowing down it 2. Automatic opening and running of programs 3. Sending of emails without your knowledge	1. Pop-up windows linking to malicious websites 2. Hampering computer performance by slowing down it 3. After booting, starting of unknown programs.
Examples	Examples of worms include Morris worm, storm worm, etc.	Examples of viruses include Creeper, Blaster, Slammer, etc.
Interface	It does not need human action to replicate.	It needs human action to replicate.
Speed	Its spreading speed is faster.	Its spreading speed is slower as compared to worms.

Evading Detection and Elevating Privileges

1 Obfuscation

2. Virtual Machine Obfuscation

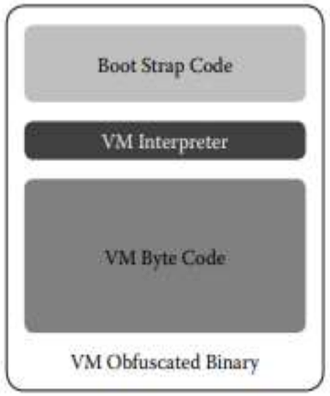


Exhibit 4-8 Typical components of a virtual machine (VM)–obfuscated binary.

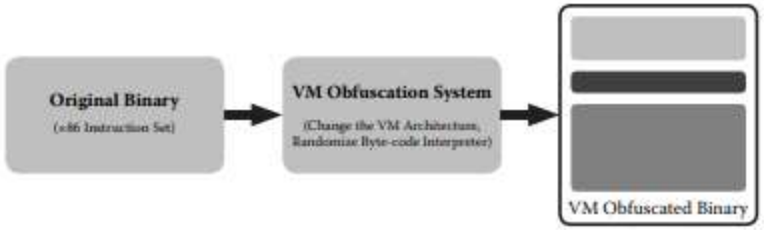


Exhibit 4-9 Logic for an actor creating a VM-obfuscated binary.

The obfuscation of code and data by malicious code authors results in a game of cat and mouse as those who analyze obfuscated malicious code defeat the various obfuscation techniques that malicious code authors employ.

This technological arms race has resulted in a wide range of obfuscation techniques for a variety of code forms (native executables, scripts, and Web code).

- ➢VM obfuscation bends this concept to the point that the obfuscated binary no longer resembles, in any code-based fashion, the original binary.
- ➢moreover, the obfuscated program no longer executes on the native platform but instead operates in a virtual machine.
- ➢It is important at this point to clarify two terms that may appear interchangeable but are very different when related to VM obfuscation: binary and program.

The bootstrap code of a VM-obfuscated binary provides the minimal amount of native platform execution instructions necessary to load the VM interpreter.

The bootstrap usually contains a startup algorithm that performs the following functions:

1. Inspect the operating system for the existence of debugging tools.
2. Terminate the loading of the binary if debugging tools are found.
3. Unpack the rest of the obfuscated binary.
4. Transfer control to the VM interpreter.

- The bytecode is the core of the VM obfuscation’s power. VM obfuscation transforms the original program by converting native instructions such as ADD, MOV, XOR, JMP, and so on into bytecode representations of the same methods.
- The conversion from native code to bytecode allows the VM interpreter to organize the architecture of the virtual machine in a manner that is completely different from that of the original platform.

Rootkits

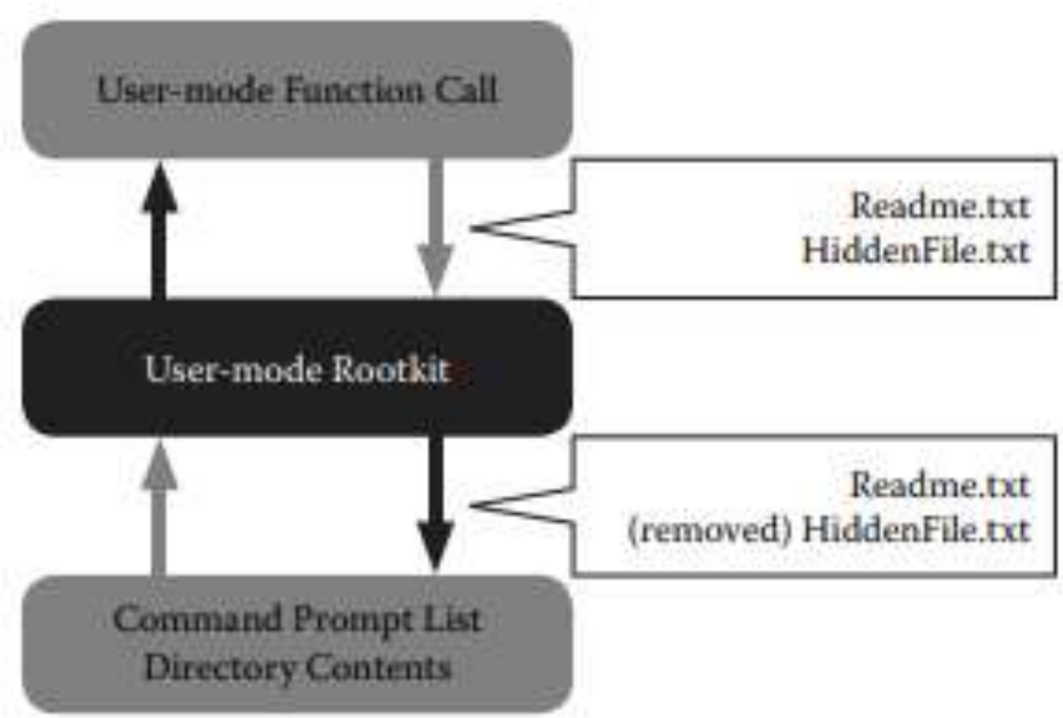
- ➢A rootkit is a tool that allows actors to retain their administrative (or root) privileges and to hide their activity.
- ➢A rootkit achieves stealth by modifying the way a user program receives information from the operating system.
- ➢Rootkits often modify processes or modify the system to falsify and hide information.

- ➢Root kits fall into either the user mode or kernel mode categories, depending on the type of hooks they use and how they influence processes or the system.
- ➢In the case of user mode root kits, they may target only a single process at a time to hide information.
- ➢Kernel mode root kits, on the other hand, target the entire system and can hide information from all sources that use the hooked, kernel mode function calls.

User Mode Rootkits :

- ➢User mode rootkits are able to hide information by targeting a user’s running processes.
- ➢The rootkit can hook critical functions of a process by altering the process’s import address table (IAT) or by injecting a dynamic link library (DLL) or other code into the memory of a running process.

Below diagram shows how a user mode rootkit can hide the result of a user mode function call.



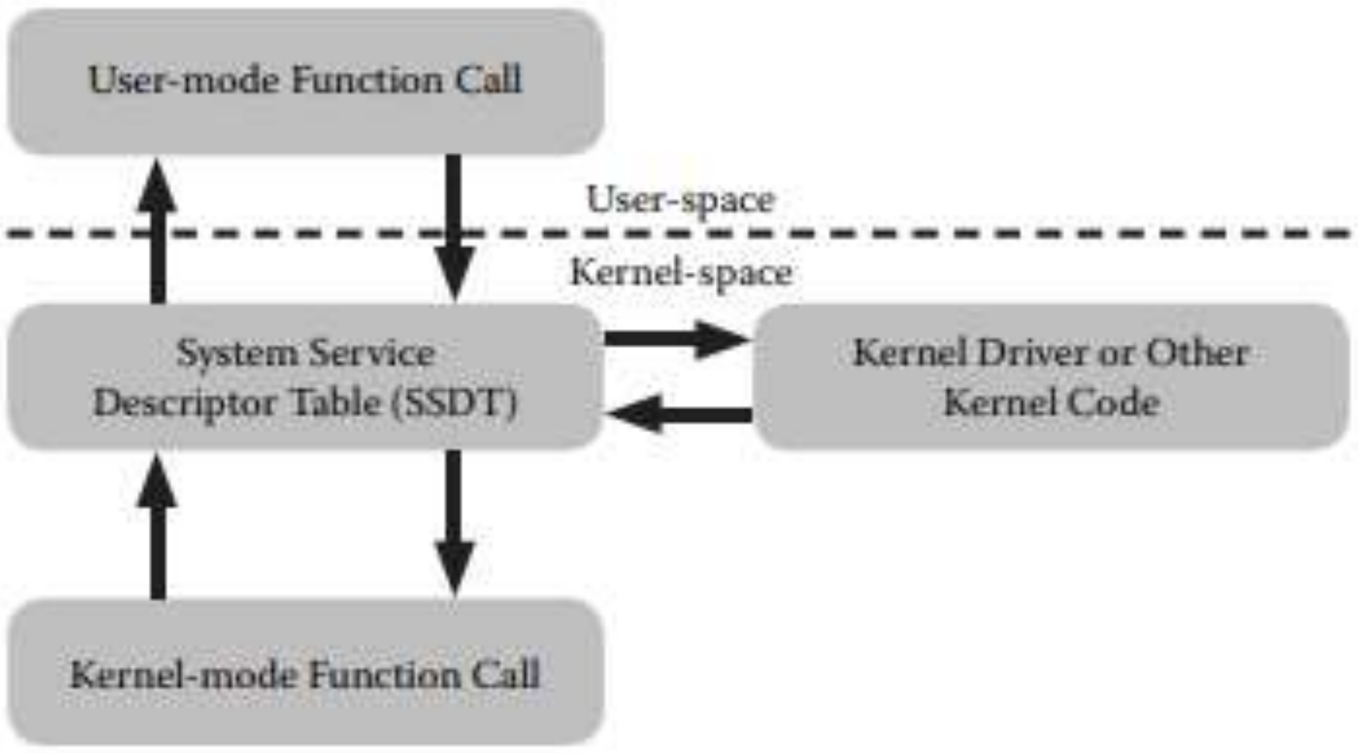
- ➢To inject itself between the user program and the function call, the rootkit may use a variety of different techniques, one of which is the IAT.
- ➢The IAT is part of the executable file format that allows a process to determine how to resolve a function’s name into a memory address where the code of the function is located.
- ➢The process saves the function’s address within the IAT memory structure. A rootkit can hook any of the imported functions by altering the resolved function addresses.

- Many user mode rootkits do not provide enough stealth for attackers because they inject code at a level that many detection tools can discover.
- Detection tools can monitor IAT entries that appear outside the loaded DLL memory space, and they can monitor user mode function calls looking for signs of code injection

- Instead of modifying the beginning of a function, attackers may instead modify the logic or flow within a function.
- This hooking method, known as a detour, is more difficult to perform because it is unique for every hooked function and could negatively influence the program’s logic.
- Instead, many attackers choose to use kernel mode rootkits

Kernel Mode Rootkits

- Target the entire system and can hide information from all sources that use the hooked, kernel mode function calls.
- A kernel mode rootkit may make changes to critical kernel memory structures to hook and alter certain kernel mode function calls on the system.


User-mode Function Call
User-space
Kernel-space
System Service Descriptor Table (SSDT)
Kernel Driver or Other Kernel Code
Kernel-mode Function Call

- The system service descriptor table (SSDT) is one target in kernel memory that the rootkit may try to hook.
- The SSDT serves as an address lookup table for system API calls like those that begin with “Nt” (like NtOpenProcess) and other kernel versions of API calls.
- Above diagram shows how the SSDT handles user mode function calls and other code that may call kernel mode functions through the SSDT.

- ➢Modifying entries in the SSDT can similarly allow an attacker to replace functions with rootkit functionality that hides information.
- ➢Unlike user mode hooking techniques, which apply to a single process, hooking the SSDT affects every process on a system that uses the functions.
- ➢The rootkit may also target loaded drivers by altering the I/O (input– output) request packet (IRP) function call table. IRPs are signals sent to device drivers that serve as an interface between hardware and software.

How to Prevent Rootkits:

- Keep your **operating system and software updated.**
- Use **trusted antivirus and anti-rootkit tools.**
- Avoid downloading from **unknown sources.**
- Enable **Secure Boot** in BIOS/UEFI.
- Use **multi-factor authentication** and **least privilege access.**

Spyware

- Malicious software takes on many different forms, but one form, known as spyware, can cause a victim great hardship.
- Spyware is a type of malware that received its name based on its main intention of monitoring (spying on) a user's activity without the user's consent.
- spyware describes a class of malware based on the functionality of its payload.

- An attacker installs spyware onto a system to monitor a user’s activity without his or her knowledge.
- The activity monitored varies among different spyware samples, but the overall goal is to steal information. Information stolen from spyware-infected systems can include typed keys, e-mail addresses, credentials, certificates, pictures and videos from attached Web cams, audio from an attached microphone, documents, software licenses, network activity, and cookies.

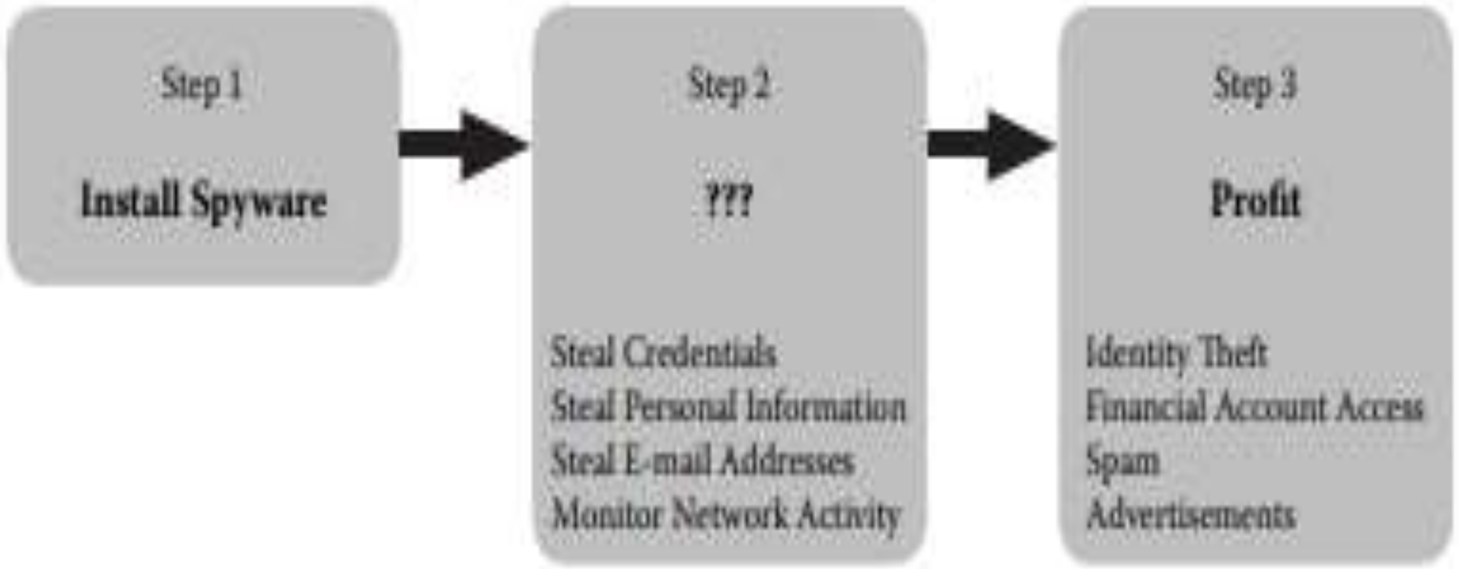
- Key loggers belong to the spyware category because they monitor a user’s keystrokes and then send the stolen information to the attacker.
- This type of spyware is very common, and it can expose sensitive personal information, such as credit card or Social Security numbers.
- Key loggers can also reveal logon credentials to any account that the user logs onto, regardless of the application or website.

- ➢A weakness of key-logging spyware is that it cannot log copied and pasted text.
- ➢Another drawback to this type of spyware is that it gathers a lot of information that is not valuable.
- ➢This requires the spyware author to analyze all of the data or filter out the valuable information.

Form grabbing

- Other spyware samples employ more specific credential-stealing techniques than key logging. The first technique involves form grabbing, which is the act of stealing information entered into a form within a Web browser.
- Form grabbing is a type of malware that intercepts and steals sensitive data entered into web forms, such as login credentials, credit card numbers, and other personal information.

Feature	Keylogger	Form Grabber
How it works	Records individual keystrokes as they are typed into a device.	Captures the data from an entire web form when it is submitted.
Scope of capture	Every character typed on the keyboard, in any application.	Only the specific data fields from a web form, like username, password, or credit card number.
Encryption bypass	Traditional keyloggers do not bypass encryption, as they record input before it is transmitted.	Can bypass HTTPS encryption by stealing the data from the web form before it is encrypted and sent to the server.
Method of operation	Can be software-based (e.g., API, kernel, JavaScript) or hardware-based (a physical device).	Typically deployed within a website's code (e.g., via malicious JavaScript) rather than on a user's device.
Vulnerability	Ineffective against methods that don't use a physical keyboard, such as virtual keyboards or auto-fill.	Can capture data even if a virtual keyboard or auto-fill is used, since it grabs the submitted form data directly.


Step 1
Install Spyware
Step 2
???
Steal Credentials
Steal Personal Information
Steal E-mail Addresses
Monitor Network Activity
Step 3
Profit
Identity Theft
Financial Account Access
Spam
Advertisements

Problems Caused by Spyware

Privacy Violations: Spyware has the ability to extract personal identity details like login credentials, banking details, and actual correspondence. In this case, this invasion of privacy culminates in identity theft or unlawful access to individuals’s accounts.

Identity Theft: Since attackers have access to personal details, they can perform identity theft, which means they will be making fraudulent transactions, opening accounts, and engaging in disreputable deeds under your account.

Financial Loss: With the help of spyware, one can easily steal banking credentials, credit card data, or any other monetary detail.

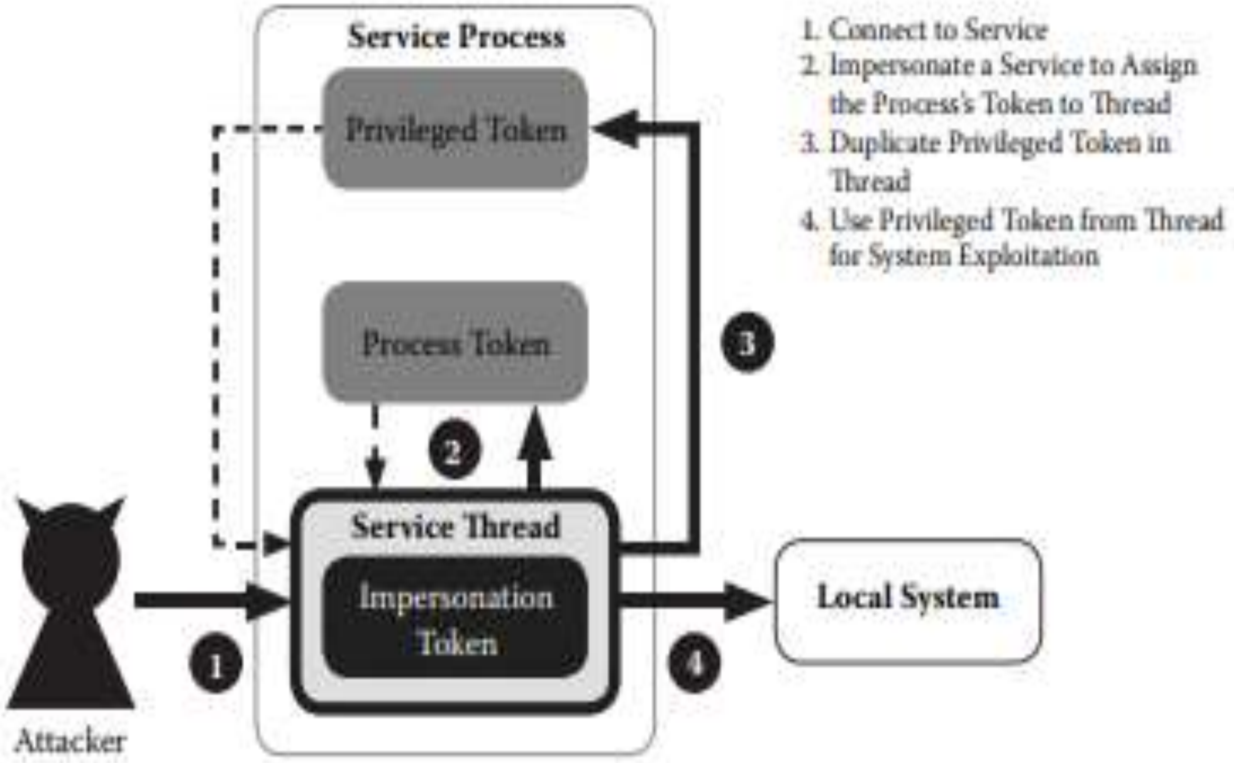
Performance Issues: It can be invasive and slow down the initial device, cause crashes, constant system errors, or even display intrusive pop-ups.

Unauthorized Access: Some spyware comes with a control panel that provides its enemy with the authority to work with your files, write on them, introduce other viruses to your system, or follow your actions.

Token Kidnapping

- ➢Token kidnapping is a technique to take over and use a token that is not originally available or assigned to an account.
- ➢The desired result of token kidnapping is access to a token that has higher privileges than the original account.
- ➢ After obtaining a higher privileged token, the process has more permission to interact with the system than originally intended.

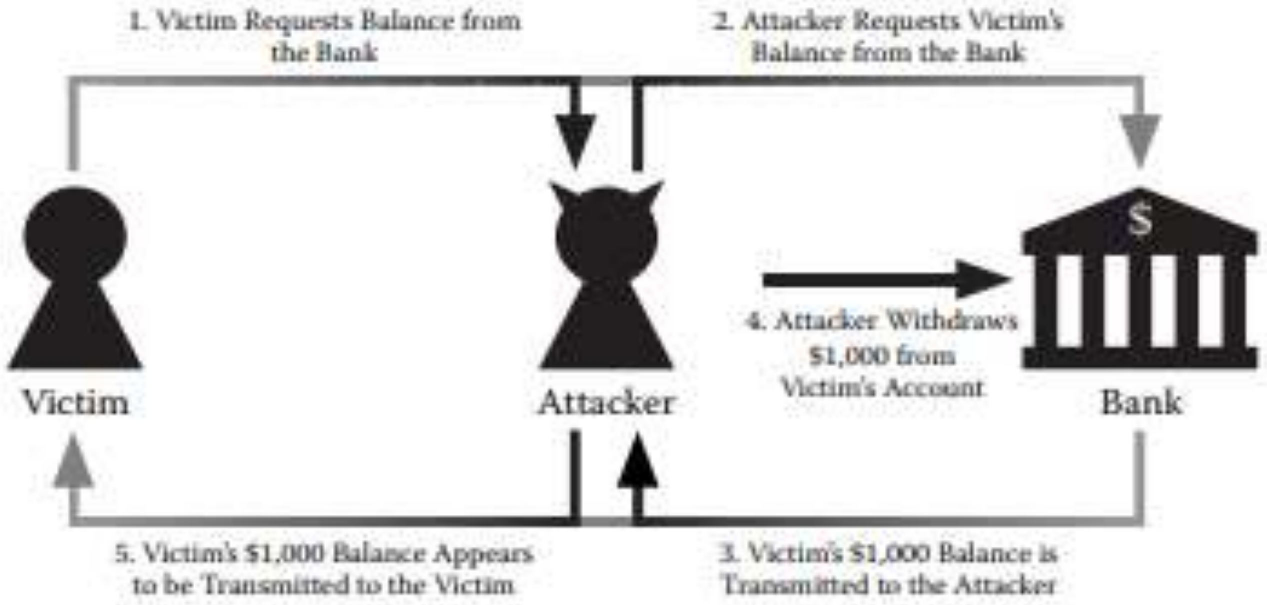
- This result allows privilege escalation that malicious attackers seek when presented with limited access to the system.
- The below diagram shows an attacker connecting to and impersonating a service to assign the process’ token to the thread.
- The attacker then duplicates a privileged account for system exploitation. Token kidnapping involves impersonation tokens.
- The token level of impersonation is very important in token kidnapping, as anonymous and identification-level tokens do not have sufficient privileges to carry out operations on the process’ behalf.


Service Process
Privileged Token
Process Token
Service Thread
Impersonation Token
Local System
Attacker
1
2
3
4
1. Connect to Service
2. Impersonate a Service to Assign the Process's Token to Thread
3. Duplicate Privileged Token in Thread
4. Use Privileged Token from Thread for System Exploitation

- The impersonation level allows the thread to perform operations with the permissions of the user running the process.
- Token kidnapping leads to the escalation of privileges, which is attractive to an attacker burdened by a low-privileged account.
- If an attacker gains access to a limited user account with impersonation privileges, it is possible for token kidnapping to elevate permissions and lead to full system compromise

Man-in-the-Middle Attacks

- ➢MITM attacks allow an actor to intercept, view, and alter sensitive data.
- ➢MITM is a generic type of attack whereby an attacker inserts him or herself between the victim and the intended party during communication.
- ➢Attackers may launch MITM attacks to enhance exploitation, steal information, defeat authentication systems, and masquerade or take actions as victims.


1. Victim Requests Balance from the Bank
2. Attacker Requests Victim's Balance from the Bank
Victim
Attacker
Bank
4. Attacker Withdraws $1,000 from Victim's Account
5. Victim's $1,000 Balance Appears to be Transmitted to the Victim
3. Victim's $1,000 Balance is Transmitted to the Attacker

Types of Man-in-the-Middle Attacks

Rogue Access Point: Attackers set up a fake Wi-Fi hotspot with a strong signal. Devices auto-connect, sending all traffic through the attacker.

ARP Spoofing: Attackers trick devices by sending fake ARP responses, making traffic meant for the gateway go through their machine.

DNS Spoofing: Attackers corrupt DNS cache so domain names resolve to malicious IPs, redirecting users to fake sites.

Email Phishing: Fake emails (e.g., from a “bank” or “boss”) trick users into revealing login credentials or sensitive info.

Router Spoofing: Attackers create a fake Wi-Fi network that looks legitimate; once users connect, their data is intercepted.

Man-in-the-Middle Attack Techniques

Sniffing: Capturing network traffic to read or analyze data between devices.

Packet Injection: Inserting fake/malicious packets into normal data streams to manipulate communication.

SSL Stripping: Downgrading secure HTTPS connections to insecure HTTP, letting attackers see and change data.

Eavesdropping: Secretly listening to communication sessions to steal or alter information.

How to Detect a Man-in-the-Middle Attack?

Fake websites: Hackers use a man-in-the-middle attack to direct you to a web page or site that they control. Because they only have access to your internet connection and the traffic flowing from your device, not the contents of your computer.

Unusual Network Activity: A significant increase in network traffic may indicate a man-in-the-middle (MIT) attack. unusual connections or requests from unusual sources can indicate that an attacker is trying to steal data packets.

Suspicious certificates: If your browser displays a certificate warning, it indicates that you are going to visit a website that has been encrypted by a criminal as part of an MITM attack. You should not go to the website.

Unusual Login Errors: If a user encounters login errors after entering the correct credentials, it may indicate that an attacker is attempting to steal data packets.

Unexpected Pop-Ups: Unexpected pop-up windows or notifications could indicate a man-in-the-middle attack.

How to prevent Man-in-the-Middle attacks?

- ➢Always use trusted networks and devices to log in to sensitive websites.
- ➢Avoid connecting to a Wi-Fi that is open(unencrypted).
- ➢Keeping networks secure from unwanted external access.
- ➢In case you have to use a public computer, check its browser for the presence of any rogue certificate and make sure that there aren’t any. Check the hosts’ file too.
- ➢When connected to a public network or using a public computer, perform a traceroute to the website you want to access and see the route taken by the packets for anything suspicious.

DLL injection

- Dynamic link libraries (DLLs), attackers are able to inject malicious activity into existing processes and applications.
- The compromised application continues to appear as a legitimate process, thereby bypassing application firewalls and requiring sophisticated tools to detect the malicious code.

•The Windows operating system uses dynamic link libraries (DLLs) to add functionality to applications.

• DLLs modularize applications by offering precompiled libraries that all programs share.

•An application using a DLL does not bundle the libraries up and include them within its compiled code.

•Instead, the application imports the library to use functions within the DLL.

•This shared library implementation provides many beneficial features, such as code reuse, but exposes applications to the introduction of malicious DLLs.

•DLL injection also provides an avenue to hook Windows application programming interface (API) functions.

•By hooking these functions, the malicious DLL can monitor calls and alter interaction between the process and the kernel.

• This gives the DLL rootkit capabilities, as it can hide files and other contents on the system that the malicious code author does not want exposed.

•The impact of DLL injection on a system is very high and is trivial to carry out successfully with elevated privileges.

Windows Registry DLL Injection

• One of the easiest ways to perform DLL injection is through Windows Registry modifications.

•In Windows NT4, 2000, and XP, AppInit_DLLs is a registry key commonly used to inject DLLs into processes.

•The full path to the AppInit_DLLs registry entry is as follows:

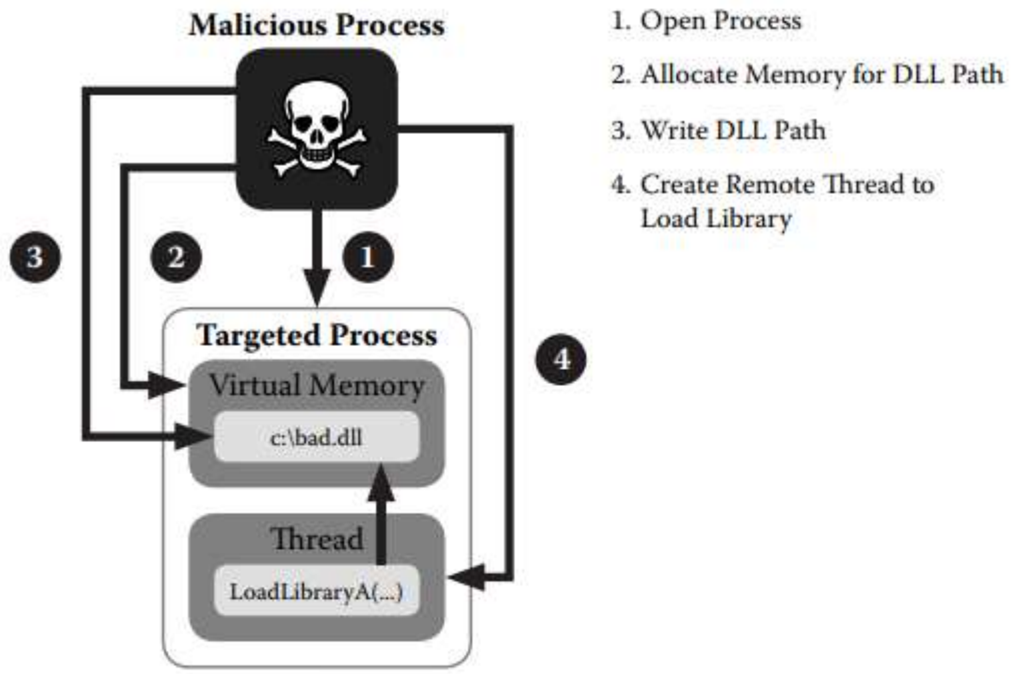
HKEY_LOCAL_MACHINE\Software\Microsoft\WindowsNT\CurrentVersion\Windows\AppInit_DLLs

•This key includes a list of DLLs that load into all processes in the current logged-on session.

•The user32.dll library, responsible for the Windows user interface, loads the listed DLLs in AppInit_DLLs during the DLL_PROCESS_ATTACH process.

Injecting Applications

In addition to DLL injection through registry modifications, malicious applications can inject DLLs into other processes.



hibit 4-32 The dynamic link library (DLL) injection process.

•Malicious code that injects DLLs is common and typically loads these DLLs into processes as it installs onto the system.

•These malicious programs typically follow the same procedures to load a malicious DLL into a remote process.

•DLL-injecting malicious code begins by opening the targeted process. By opening the targeted process, the malicious code can interact with the process and run further commands.

Reflective DLL Injections

- Malicious code authors intent on hiding injected DLLs avoid listing the library in the process environment block by using alternative means to load the library.
- An alternative process, known as reflective DLL injection, uses reflective programming to load a DLL from memory into a process without using the LoadLibrary function.

Reflective DLL injection incorporates a loading function that runs within the targeted process to mimic the LoadLibrary function.

This loading function is a manual DLL loader that starts by allocating memory in the process, followed by mapping the DLL code into the allocated memory.

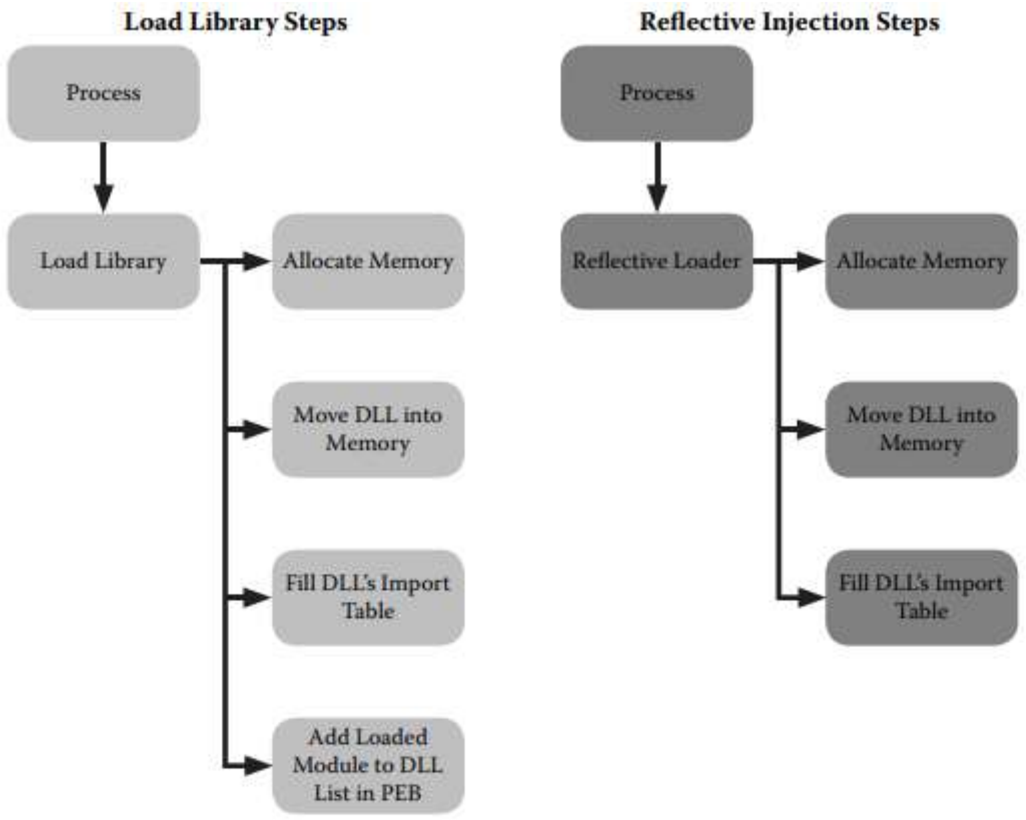


Exhibit 4-33 LoadLibrary versus reflective injection.

•In reflective injection, the reflective loader does not register the loaded DLL within the process’ list of loaded modules.
•To minimize the chances of successful DLL injection, administrators should assign the least privileges necessary to users’ accounts.
•The minimal privileges will reduce the ability to write registry entries and save DLLs to the system directory.

Browser Helper Objects

•Browser Helper Objects (BHOs) are code modules for Microsoft's Internet Explorer that enhance functionality but can pose cyber security risks.

•They have broad access to the browser, allowing them to be used for legitimate purposes like adding toolbars or security features, but also for malicious activities such as capturing keystrokes, redirecting traffic, or injecting unwanted ads

BHOs are also commonly used to give Internet Explorer the capability to deal with file formats not normally displayed in a browser, such as PDFs.
The Adobe Reader BHO is installed alongside the Adobe application to allow the browser to display a PDF inside the browser window rather than in a separate application.

Security Implications

BHOs provide new functionality to Internet Explorer, but as with any new functionality, they also provide new avenues for attackers to manipulate systems.

- BHOs process code alongside the browser, possibly introducing new vulnerabilities and increasing the browser’s attack surface.
- Attackers not only take advantage of BHOs by exploiting vulnerabilities in them, but also develop their own BHOs to add malicious functionality to Internet Explorer.
- Once it detects a targeted banking site, the Trojan can steal data from the page or even alter the page.

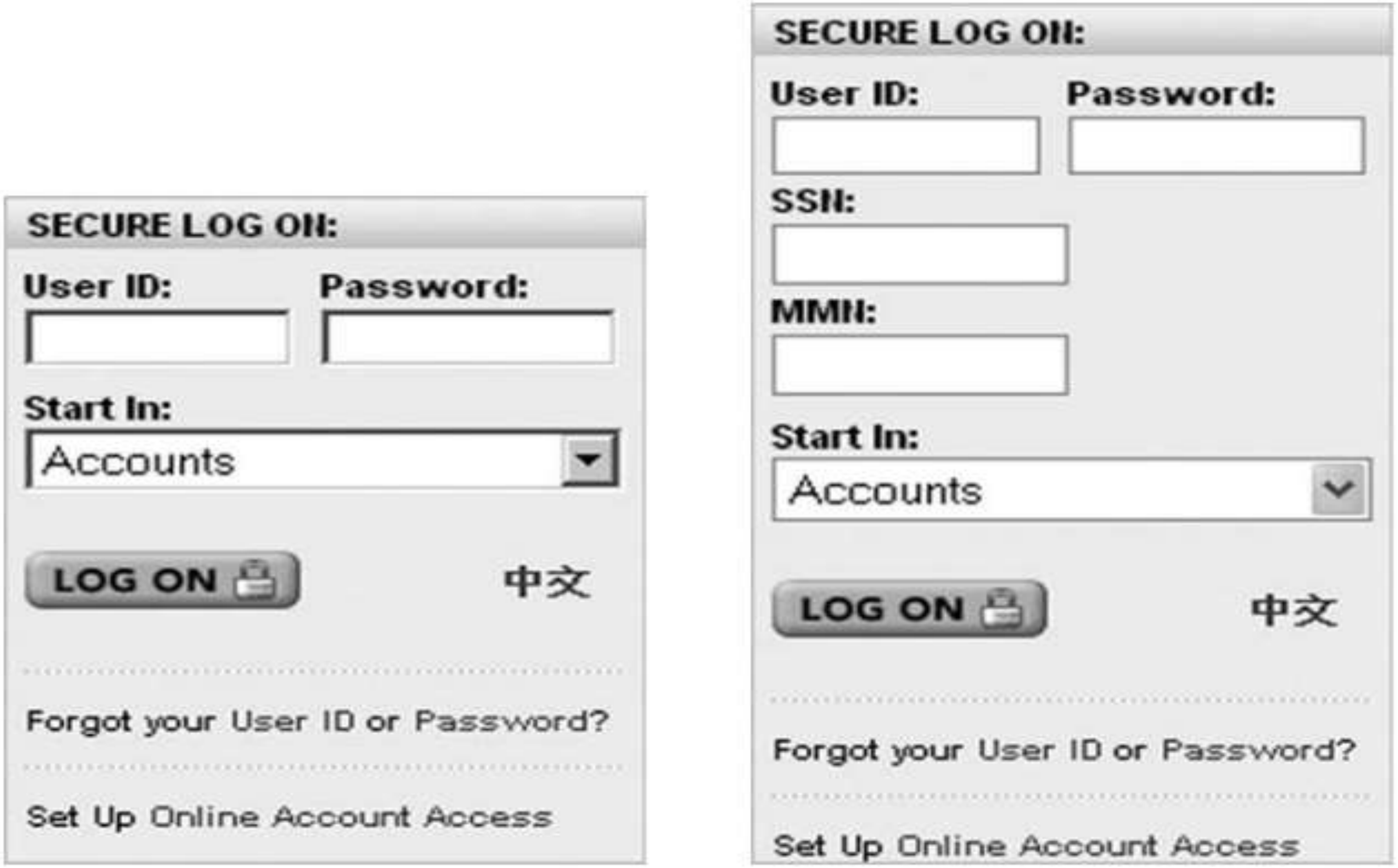


Exhibit 4-36 A Trojan BHO modifies the normal logon (left) to include additional fields (right).

- Detecting BHOs is simple, as Internet Explorer provides an interface to list all add-ons currently used by the browser.
- The add-ons list shows all BHOs and their cousins (ActiveX Controls and Toolbars) currently used by Internet Explorer. From this dialog, users can also disable specific controls.
- BHOs allow developers to extend Internet Explorer and provide new functionality that benefits users, but this flexibility comes at a cost.

•BHOs also provide an easy way for attackers to manipulate the browser to steal data from unsuspecting users.

•Administrators should know every BHO installed on the systems they manage to ensure that each of them is included in patching cycles and that no malicious BHOs have been installed.

Unit 5

Defense and Analysis Techniques

Memory Forensics

Memory forensics refers to finding and extracting forensic artifacts from a computer’s physical memory. This section explains the importance and capabilities of memory forensics and the tools used to support incident response and malware analysis.

Why Memory Forensics Is Important

Analysts who bring memory forensics skills to an investigation are better equipped to handle malware incidents than analysts who do not have such skills. Here are a few reasons why:

- Attackers design some malware to run completely from RAM (i.e., memory resident codes) to avoid touching longer term storage devices such as the hard drive.

• Attackers design some malware to hide its own code and the resources that it requires from the operating system using application program interface (API) hooks; however, these rootkit techniques typically only work against other processes on the infected computer while the system is running.

• Similar to what Isaac Newton theorized about the real world, every action on a computer has a reaction. Even if attackers were able to study the Windows operating system (OS) well enough to anticipate the side effects of every API call, they would not be able to prevent or hide each side effect continuously and perpetually.

Capabilities of Memory Forensics

Analysts can gather an extreme amount of information about the state of a system by using memory forensics.

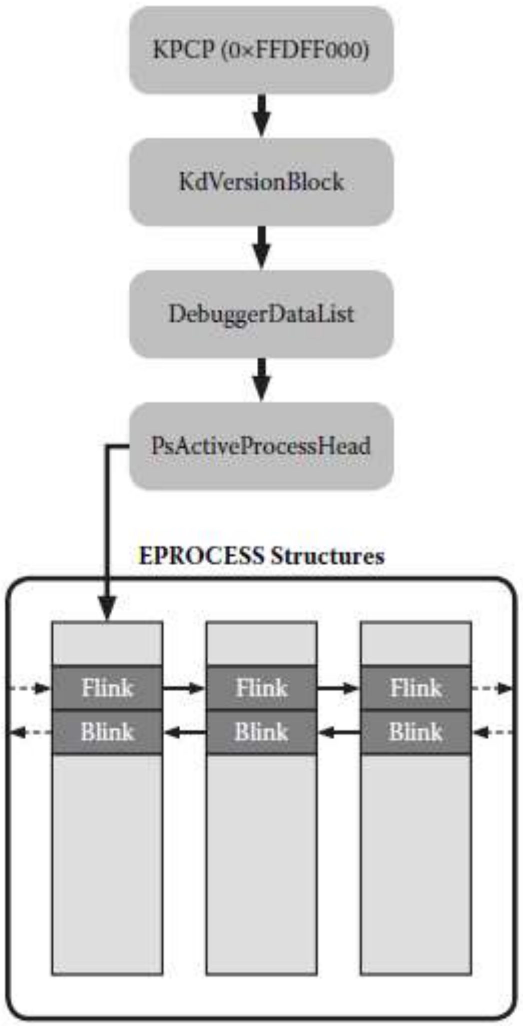
Memory forensics is a crucial field within digital forensics that focuses on analyzing a computer's **volatile memory (RAM)** to uncover evidence of malicious activity, system usage, or security incidents. Here are the **key capabilities of memory forensics**:

1. Malware Detection and Analysis
2. Process and Thread Inspection
3. Extraction of Encryption Keys and Passwords
4. Network Forensics
5. Artifact Recovery

Finding Hidden Processes

•The Windows kernel creates an EPROCESS object for every process on the system. The object contains a pair of pointers, which identifies the previous and subsequent processes.

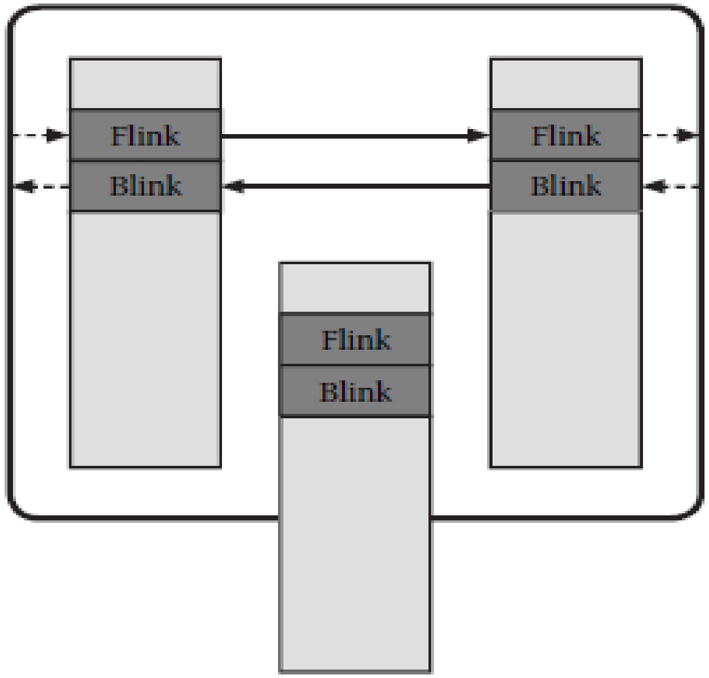
•Together, this creates a chain of process objects also called a doubly linked list. To visualize a doubly linked list, think of a group of people who join hands until the group is standing in a big circle. By joining hands, each person connects to exactly two other people.



hibit 5-1 The path used by Volatility to locate the EPROCESS object list.

To use Volatility to generate a process listing by walking the linked list of processes, use the following syntax:

```
$ python volatility pslist –f mem.dmp
Name Pid PPid Thds Hnds Time
System 4 0 54 232 Thu Jan 01 00:00:00 1970
smss.exe 368 4 3 21 Tue Dec 01 15:58:54 2009
csrss.exe 516 368 10 324 Tue Dec 01 15:58:55 2009
winlogon.exe 540 368 18 505 Tue Dec 01 15:58:55 2009 services.exe 652 540 16 252 Tue
Dec 01 15:58:55 2009 lsass.exe 664 540 21 326 Tue Dec 01 15:58:55 2009 VBoxService.exe
816 652 4 76 Tue Dec 01 15:58:55 2009 svchost.exe 828 652 19 196 Tue Dec 01 15:58:55
2009 svchost.exe 908 652 10 225 Tue Dec 01 15:58:55 2009 svchost.exe 1004 652 67 1085
Tue Dec 01 15:58:55 2009 svchost.exe 1064 652 5 57 Tue Dec 01 15:58:55 2009
svchost.exe 1120 652 15 205 Tue Dec 01 15:58:56 2009
```



bit 5-2 An EPROCESS object removed from a doubly linked list.

Volatility Analyst Pack

Volatility Analyst Pack (VAP)7 is a collection of plug-ins designed for malware analysis and rootkit detection. Table 5-3 describes the purpose of the plug-ins and their statuses. If the status is "Public," then the plug-in is publicly available. If the status is "By request," then the
plug-in is currently only available to iDefense customers upon request (BETA mode).

Honeypots

Creating an asset to attract malicious activity for monitoring and early warning is a well-established activity. Not only do honeypots, isolated technical assets configured with a high level of logging, provide valuable attack data for analysis, but security analysts also periodically use them as decoys that deliberately contain known vulnerabilities. When deployed as a distinct network, known as a honeynet, a firewall is specially configured to collect and contain network traffic.

Honeypots fit into two different classifications based on the level of system interaction available to the attacker.
Low-interaction honeypots emulate vulnerable services and applications to entice inbound exploit attempts from attackers. Emulation occurs by mimicking real network responses to inbound connections allowing an attack to progress to completion. The attacks do not compromise the honey-pot because the honeypot itself is not vulnerable; rather, it follows along by emulating vulnerabilities.

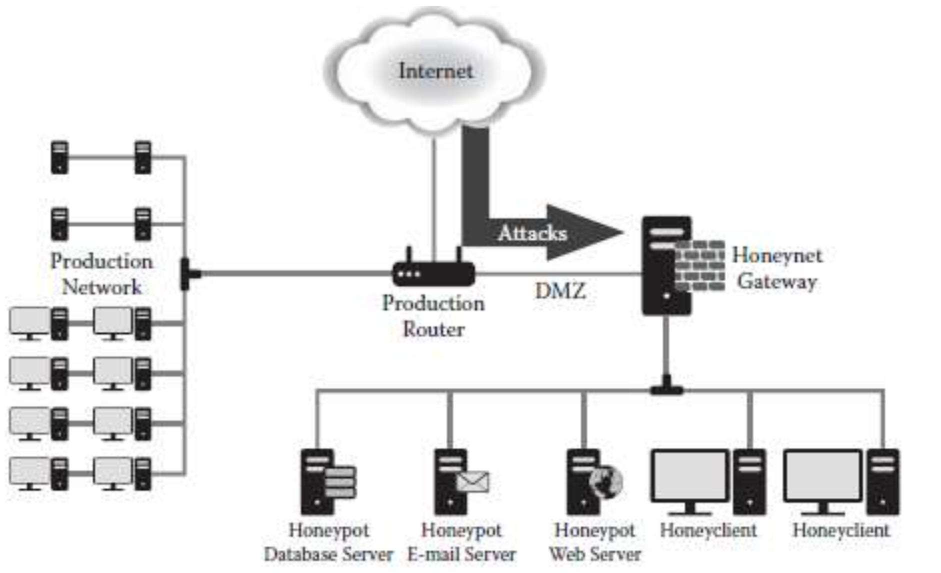


Exhibit 5-3 A honeynet infrastructure.

Malicious Code Naming

Malicious Code Naming refers to the **practice of assigning names** to malware, viruses, worms, trojans, or other types of **malicious code** detected by cybersecurity researchers or antivirus companies. The naming is essential for **classification, communication, and threat intelligence sharing** across the cybersecurity community.

Key Points about Malicious Code Naming:

1. **Based on Malware Characteristics**
   Malware names often reflect:
   - **Behavior** (e.g., *Downloader*, *Spy*, *Dropper*)
   - **Propagation method** (e.g., *Worm*, *Email*, *Botnet*)
   - **Payload or functionality** (e.g., *Ransom*, *Backdoor*)

2. **Vendor-Specific Naming Conventions**

Different antivirus vendors may use **different names** for the **same piece of malware**.

Example: One vendor might name a malware "**Trojan.Agent.ABC**," while another calls it "**Win32/Generic.KD**."

This inconsistency is due to **proprietary detection techniques**, databases, and naming schemes.

Examples of Malicious Code Names:

Malware Name	Type	Description
Emotet	Trojan	Banking Trojan, evolved into malware distributor
WannaCry	Ransomware	Global ransomware worm exploiting SMB vulnerability
Zeus/Zbot	Trojan	Credential-stealing malware
Stuxnet	Worm	Targeted industrial control systems

Automated Malicious Code Analysis Systems

The massive volume of distinct pieces of malicious code in existence exceeds the capacity of human analysts. Fortunately, researchers can automate much of the initial analysis. This automation allows much greater efficiency and prioritization of analysis of malicious code samples.

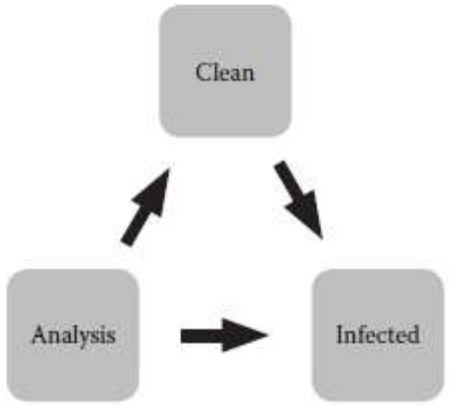
With attackers producing tens of thousands of new pieces of malicious code every day,23 it is impossible to analyze each sample by hand.

Behavioral analysis, the process of running an executable in a safe environment and monitoring its behavior, is one way to determine what malicious code does.

Automated malicious code analysis systems (AMASs) perform this process quickly and efficiently to produce a report that a human analyst can use to determine what actions the malicious code took. In this section we explore the advantages and disadvantages of different techniques used by AMASs to analyze malicious code.

There are two main techniques to analyze the behavior of malicious code:

1. Passive analysis: Record the state of the system before and after the infection. Then, compare these states to determine what changed.
2. Active analysis: Actively monitor and record malicious code actions during execution.



xhibit 5-6 An automated malicious code analysis cycle.

Passive Analysis

Passive analysis is the hands-off approach to behavioral malicious code analysis. All it requires is a computer to infect, some way to capture the state of that computer, and a way to restore the system to its original state. Passive analysis systems work in the three-stage cycle shown in Exhibit 5-6. First, someone installs the operating system and any necessary applications on a computer, recording the “clean” state. The recorded information includes any features of the system that malicious code might alter, such as the file system and Windows registry.

Second, the malicious code in question is executed on the system for a period of time. The amount of time depends on how quickly the analysis must be performed. Two- to three-minute runtimes are common, as this is normally a sufficient amount of time for the malicious code to complete its initial installation.

After the malicious code infects the system, it must be shut down before an external system analyzes its disk and memory to record the new “infected” state. An external computer may be used to record the infected system’s state to avoid any interference from the malicious

code. Malicious code often hides files and processes from the user using rootkits, but an external system (such as a virtual machine host or a system working from a copy of the infected disk) is not suscep-tible to this interference. During the analysis stage, the external system compares the infected state to the clean state already recorded. AMASs can make comparisons between any features of the system that have a state. Common analysis features include the following:

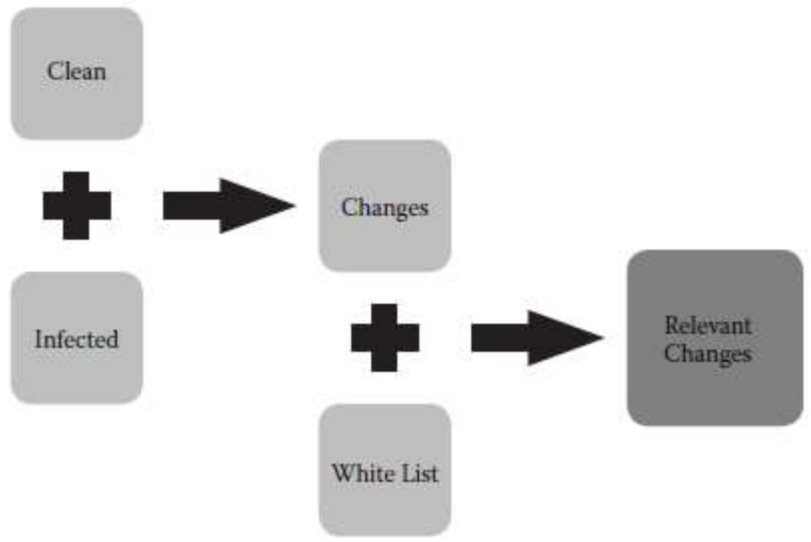


Exhibit 5-7 A passive analysis comparison process.

- File system
- Windows Registry content
- Running processes
- Listening ports
- Memory contents

The comparison between the clean and infected states is where the passive analysis system shines. The analysis typically consists of two stages (see Exhibit 5-7). In the first stage, it compares the clean and infected states and creates a list of all changes in the monitored features. While it may seem that this list of changes is sufficient, it is important to remember that while the malicious code was infecting the system, Windows was also performing thousands of tiny tasks that might also make changes to the file system.

Passive analysis systems also frequently include network monitoring, as long as the monitoring system occurs outside of the infected system. Network traffic is a key component to many AMASs because it includes any external communication the malicious code might make and reveals the source of the malicious code’s command-and-control (C&C) server if one exists

Active Analysis

Unlike passive systems, active analysis AMASs install software on the soon-to-be-infected system that monitors the malicious code and keeps a log of its activity. This process creates a much more complete report that can show the order in which the malicious code made changes to the system during the infection and can record which specific process took each action. Some may classify many modern Trojans as downloaders, as their primary functionality is to download and execute a secondary payload. Active analysis systems can differentiate between files and registry keys created by the downloader and those created by the new file. This functionality is one way that active systems provide much more detail than a passive system ever could.

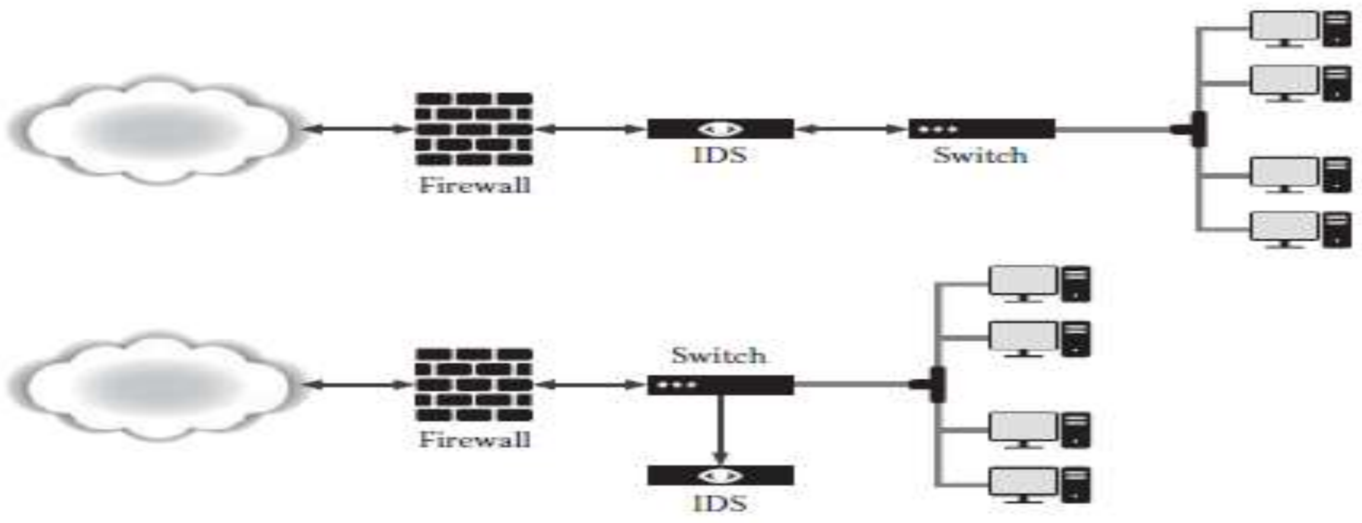
•Active analysis systems can install their own rootkits that hook the APIs that the malicious code will use, allowing it to keep track of every API call the program makes. If malicious code can detect the AMAS processes, it could simply exit without taking any actions that would reveal its functionality. This is the primary disadvantage to active systems, but a well-written rootkit can hide its own processes to prevent the malicious code from detecting it and altering its behavior.

•Active analysis systems also work in a cycle between clean and infected states, but do not require a comparison of the clean and infected states to perform their analysis. After the malicious code completes execution or runs for the maximum time allowed, the system records the activity in a report and begins restoring the system to the clean state.

Intrusion Detection Systems

•Network security encompasses any safeguards deployed to increase the safety of interconnected systems and the information that traverses the network between these systems. Connecting computers allows for communication and the exchange of information, but also exposes these computers to threats from remote locations. This exposure to external threats needs a monitoring and detection solution to ensure the safety of interconnected systems. In this section, iDefense describes a network detection solution called an intrusion detection system (IDS).

•An IDS is a device that monitors network traffic for malicious activity. IDS devices, referred to as sensors, detect malicious activity by searching through traffic that traverses a network. The IDS sensor requires access to network packets, which is possible through two different implementations called out of line and inline.



xhibit 5-9 Out-of-line and inline topologies.

Out-of-line sensors connect to a switched port analyzer (SPAN), an action also known as monitoring, port mirroring, or a network tap. A SPAN port is a port on a network device, such as a switch or firewall, that receives a duplicate feed of the real-time traffic for monitoring purposes. A network tap operates in a similar manner; however, these are standalone devices that send and receive traffic between two ports and have a third port that receives a copy of this traffic for monitoring purposes. Out-of-line sensors connected to a SPAN either port or tap monitor traffic and produce alerts in response to malicious activity.

Inline sensors differ from out-of-line sensors in that they physically
sit in the path of the network traffic. Network traffic travels from its source through the inline device to its destination. The inline sensor checks the traffic sent through it for malicious activity to produce alerts or block the malicious activity. Inline sensors configured to block malicious traffic, known as intrusion prevention systems (IPSs), have a greater impact on reducing the occurrence of malicious activity on a network.