Reverse Engineering Summary

Here is a **comprehensive, well-formatted, and exam-focused version** of your notes on the **Reverse Engineering** chapter. The content has not been changed — just improved for clarity, structure, and readability.

Reverse Engineering – Comprehensive Notes

Definition of Reverse Engineering

Reverse Engineering is the process of analyzing a subject system to identify its components and their relationships, and to create representations of the system at a higher level of abstraction.

Design Recovery in Reverse Engineering

Design Recovery is a subset of reverse engineering.

> "Design recovery is a subset of reverse engineering in which domain knowledge, external information, and deduction or fuzzy reasoning are added to the observations of the subject system."
> *— [Elliot Chikofsky and James Cross, IEEE Software 7(1):13–17, 1990]*

Purpose of Design Recovery

- To identify meaningful **higher-level abstractions**.
- These abstractions go **beyond direct system examination**.
- Helps in understanding the system design more deeply.

Why Reverse Engineering is Used

- Often **precedes re-engineering**, but can also be valuable **on its own**.
- Used to **reverse engineer the design and specifications** of a system to:
  - Serve as input for **new requirements**.
  - Support **maintenance** and **reengineering** efforts.

Reverse Engineering as a Process

Reverse engineering is not just a technique — it's a structured **process**.
It provides a **framework** to help generate valuable insights for:

- Analysts
- Stakeholders

---

1. Seeking Approval

Ethical considerations when reverse engineering software:

- Always get **approval from the software owner**.
- Even if companies are lenient, **vulnerabilities should be reported directly to the owner**, not made public.
- The **owner decides when to report** a vulnerability to the public.
- This protects systems from exploitation **before patches are released**.

Note: Exception for Malware

- No approval is needed for **malware analysis**.
- One of the goals is to **catch the malware author**.
- When in doubt, consult:
  - A **lawyer**
  - The company’s **legal department**

---

2. Static Analysis

Static Analysis involves analyzing the program **without executing it**.

Key Points:

- Examine the **binary file** directly.
- Parse bytes to extract critical information.
- Knowing the **file type** helps the analyst prepare the right tools.
- Searching for **text strings** in the binary:
  - Can reveal the **author**, **origin**, or **functionality** of the software.

---

3. Dynamic Analysis

Dynamic Analysis is the process of analyzing a program **while it is executing**.

Key Points:

- Must be done in a **safe, enclosed environment** to avoid harming production systems.
- Typically done using **virtual machines** for:
  - Isolation
  - Easy control and monitoring
- Use tools that:
  - **Monitor**
  - **Log environment actions** during execution

Let me know if you want a **flowchart, diagram**, or **summary table** for revision purposes.

Here is the **comprehensive, cleaned-up, and exam-friendly version** of your second part of the Reverse Engineering chapter notes. The formatting is improved, and sentences are made

shorter while maintaining original content and meaning. Context has been added where necessary for clarity.

Reverse Engineering – Continued Notes

Reverse Engineering as a Process

4. Low-Level Analysis

- Some critical information may be missed during **static** and **dynamic analysis**.
- **Program flow** often depends on specific **conditions**.
  - Example: A program may **create a file only if** a certain process is running.
  - Example: A program may **create a registry entry** in the `Wow6432Node` **only if** it runs on a **64-bit Windows OS**.
- Tools like **debuggers** are used in this stage to analyze **low-level behaviors**.

---

5. Reporting

- All findings during analysis must be **documented properly**.
- Reports help with **future reverse engineering** efforts.
- They also support **developers** in securing upcoming software by learning from analyzed flaws.

A Good Reverse Engineering Report Answers:

- **How** the object works.
- **When** specific behaviors are triggered.
- **Why** specific code segments are used.
- **Where** the software is intended to run.

- **What** the program does as a whole.

Key Points in Reverse Engineering

- **Goal of Re-engineering:** Improve the **system structure** to make it easier to **understand and maintain**.
- **Re-engineering process includes:**
  - **Source code translation:** Automatically convert code from one language to another.
  - **Reverse engineering**
  - **Program structure improvement**
  - **Program modularization**
  - **Data re-engineering**

Operating System Environment in Reverse Engineering

Understanding the **environment** in which the software runs is crucial:

Core System Dependencies:

- **Memory**
- **File System**
- **Registry System** (especially in Windows)

These components are heavily used by software, and understanding them helps reverse engineers interpret program behavior.

The File System

- Responsible for **storing data** directly on the disk.
- Manages how files and directories are arranged.

- Different systems offer different file handling efficiencies.

Common File Systems:

- **FAT**, **NTFS** (Windows)
- **ex2**, **ex3**, **XFS**, **APFS**

File Metadata Stored:

- File name
- File size
- Date/time stamps
- Permissions
- Directory paths

Memory (RAM) in Execution

When a Windows **executable file runs**:

1. The OS **allocates memory space**.
2. It **reads the executable** from the disk.
3. Writes it to **predefined sections** in memory.
4. Then **executes the code**.

This memory block is:

- Known as a **process block**.
- **Linked** to other process blocks.
- Every running program occupies a **separate memory process space**.

The Windows Registry System

The **Windows Registry** is a central **database** that stores:

- **System-wide configuration**
- **Application-specific settings**

Examples of Stored Information:

- **File type associations**
  - `.docx` → Microsoft Word
  - `.pdf` → Adobe Reader
- **Icons associated** with file types and folders
- **Software Settings:**
  - Uninstall configurations
  - Update URLs
  - Port assignments
- **Product IDs**
- **User and Group Profiles**
- **Printer Setup:**
  - Default printer
  - Driver details

Registry Architecture

- The registry is stored in **protected files** called **hives**.
- Even the **list of hives** is stored inside the registry.
- Data can be viewed using **Windows Registry Editor**.

Registry Editor Details:

- **Left Pane**: Shows the registry **hierarchy** (folders/keys).
- **Right Pane**: Displays **registry values** under columns such as Name, Type, and Data.

Accessing the Registry:

- Requires use of **Windows Registry APIs** for reading/writing.
- Careful handling is needed — **mistakes can corrupt system configuration**.

Let me know if you'd like:

- A **visual summary (mind map / flowchart)**
- A **flashcard set**
- A **mock test** from this chapter for revision.

Here is a **comprehensive, structured, and exam-focused version** of your final part of the **Reverse Engineering** chapter notes. Sentences are improved and shortened for clarity and better readability, while the original meaning and content are preserved. Additional context has been included where needed.

Lecture 10: Vulnerability Assessment and Reverse Engineering

Instructor: Dr. Syed Muhammad Sajjad
Department of Cyber Security, Air University, Islamabad

System Re-Engineering

Definition

- Involves **restructuring or rewriting** part or all of a legacy system **without changing its functionality**.
- Useful when **some subsystems** require **frequent maintenance**, while others do not.

Purpose

- To make systems **easier to maintain**.
- May include re-structuring and **re-documenting** the system.

When to Re-Engineer

- When **hardware or software support** is no longer available.
- When **new access methods** (e.g., mobile access, web interfaces) are required.

Advantages of Re-Engineering

- **Reduced Risk:**
  - Developing new software carries high risks—e.g., team issues, unmet specs, development delays.
- **Reduced Cost:**
  - Re-engineering is often **cheaper** than developing a new system from scratch.

Business Process Re-Engineering

- Focuses on redesigning **business workflows** to be more efficient and responsive.
- Often requires **new computer systems** to support the redesigned processes.
- Legacy software may need re-engineering to support the new workflows.

Forward Engineering vs. Re-Engineering

Process	Description
Forward Engineering	Traditional process: moves from **high-level design** to **code implementation**.

Process	Description
Re-Engineering	Begins with an existing system; focuses on **understanding and improving** it.

> “Forward engineering is the traditional process of moving from high-level abstractions and logical, implementation-independent designs to the physical implementation of a system.”
> — *[Chikofsky & Cross, IEEE Software, 1990]*

Re-Engineering Cost Factors

- **Quality** of the existing software
- **Tool support** availability
- **Extent of data conversion** needed
- **Availability of skilled staff**

Re-Engineering Approaches

From low to high cost:

1. Automated **program restructuring**
2. Automated **source code conversion**
3. **Program and data** restructuring
4. Automated restructuring with **manual changes**
5. Restructuring with **architectural changes**

Source Code Translation

- Converts code from one language/version to another (e.g., **FORTRAN → C**).
- Required due to:
    - **Hardware upgrades**

    - **Lack of staff** skilled in old languages
    - **Policy changes**
- Only feasible if an **automatic translator** is available.

The Program Translation Process

1. Identify **source code differences**
2. Design **translation instructions**
3. Automatically or manually **translate code**
4. Produce the **re-engineered system**

Reverse Engineering Overview

Definition

- **Analyzing software** to understand its design, structure, and behavior.
- Often part of the **re-engineering process** but can also support **re-specification**.

Goals

- Identify:
    - System **components**
    - Their **interrelationships**
- Create **high-level representations** (e.g., diagrams, models)

> *— [Chikofsky & Cross, 1990]*

Purpose and Analogy

- Reverse engineering is like **dissecting** a body to understand human anatomy.

- It helps understand **how something works**, **why it exists**, and **how to improve it**.
- Example: If the **Trojan Horse** had been reverse engineered before entry, the attack could have been prevented.

Reverse Engineering for Malware

- Malware is treated as the **Trojan Horse**.
- The **analyst** is the defender who inspects it.
- The **city** is the target network.

Use in Cybersecurity

- A **core skill** for security analysts.
- Every malware attack is reversed to:
  - Understand how it **enters the system**
  - Determine **persistence mechanisms**
  - Identify the **damage** it causes

Post-Attack Analysis

- The first step is **cleaning the system** of the malware.
- Analysts investigate:
  - How it got installed
  - How it became persistent
- Helps network admins:
  - Develop **policies** to mitigate future attacks
  - Block **email attachments** (e.g., JavaScript files) if needed
  - **Restructure** the network if necessary

Strengthening Defenses Through Reverse Engineering

- Attackers, after compromising a system, may already have **full network knowledge**.
- Major **network architecture changes** can help prevent repeat attacks.

Role of Education and Policy

- **User awareness** is crucial for cybersecurity.
- Educating users about:
  - **Privacy**
  - **Social engineering**
  - **Past attack scenarios**
- Leads to:
  - **Stronger security policies**
  - **Backups**
  - A culture of **continuous learning**

Let me know if you want:

- A **revision handout**
- **Practice questions**
- A **diagram summary** of this lecture

Here is a **structured, concise, and exam-ready** version of your notes on **Typical Malware Behavior & Persistence Techniques**. The formatting is improved, sentences are shortened while preserving meaning, and relevant context has been added to help you understand better.

Typical Malware Behavior and Persistence Techniques

What is Malware?

- **Malware** stands for **malicious software**.
- It refers to any software created with harmful intent.
- Once malware enters a system, it typically:
  - **Installs itself**
  - Begins to perform its **malicious actions**
- Malware often installs silently — **no user notification** is needed.
- It directly **modifies system components** to survive and execute.

Persistence

Definition:

- **Persistence** is when malware ensures it **remains active** over time.
- Malware stays resident in the system and tries to execute:
  - **Every time the system boots**, or
  - **At scheduled times**

How Malware Achieves Persistence

Common Technique:

- **Drops a copy** of itself into a system folder.
- **Creates registry entries** to automatically execute during startup.

Example:

- **GlobeImposter ransomware**
  - Path: `C:\Users\JuanIsip\AppData\Roaming\huVyja.exe`
  - Registry Entry:
    - Key: `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run`
    - Value name: `BrowserUpdateCheck`
    - Value data: executable file path
- **Important Note:** The **path matters**, not the registry value name.

Registry Keys Used for Persistence

Run Keys (System-Wide)

Trigger execution **when Windows starts**:

- `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run`
- `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce`
- `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnceEx`
- `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServices`
- `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\RunServicesOnce`
- `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run`
- `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Windows\CurrentVersion\Run`

Run Keys (User-Specific)

Trigger execution **when the current user logs in**:

- `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run`
- `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce`
- `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnceEx`
- `HKEY_CURRENT_USER\Software\Microsoft\WindowsNT\CurrentVersion\Windows\Run`

Note on "Once" Keys:

- Programs listed under `RunOnce` or similar will execute **only once**, but malware can reinsert itself repeatedly to persist.

Other Registry-Based Persistence Mechanisms

Load and Run Values

- Key: `HKEY_CURRENT_USER\Software\Microsoft\WindowsNT\CurrentVersion\Windows`
  - `Load = <file path>`
  - `Run = <file path>`

BootExecute Value

- Key: `HKEY_LOCAL_MACHINE\SYSTEM\ControlSetXXX\Control\Session Manager`
  - XXX = ControlSet001, 002, etc.
  - `BootExecute = <file path>`
  - Default: `autocheck autochk *`

Other Persistence Keys

- **Winlogon key**
- **Policy scripts keys**
- **AppInit_DLLs**
- **Services keys**
- **File associations**

Startup Folder

- **Startup location** is another method to persist:

- Path:
  `%APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup`
- Files in this folder **automatically run after user login**.

Advanced Techniques

Image File Execution Options (IFEO)

- Malware can use this key to run its own code **instead of the real application** by:
  - Setting a **debugger path** under:
    - `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options`

Browser Helper Objects (BHOs)

- Registry keys that allow malware to load with **Internet Explorer** or similar browsers.

Task Scheduler & Conditional Execution

- Malware can **schedule tasks** to run at:
  - Specific times
  - Certain system conditions
- Also possible through **cron jobs** on Unix-based systems.

Let me know if you'd like:

- A **table format** summary for quick review
- A **diagram of malware persistence methods**
- **Practice MCQs** or **exam-style questions** based on this topic

Here is your **cleaned, well-formatted, and exam-ready version** of the chapter **"Typical Malware Behaviour & Persistence Techniques"** with improved sentence structure, clear bulleting, and consistent formatting. The original content is preserved; only formatting and readability are improved.

Typical Malware Behaviour & Persistence

What is Malware?

- Malware is short for **malicious software**.
- When malware enters a system:
  - **It installs itself silently** (without notifying the user).
  - It immediately begins **modifying system components** to perform malicious activities.

Malware Behaviour

- Once inside the system, malware does two main things:
  1. **Installs itself**
  2. **Performs malicious tasks**
- It does not require user interaction or consent.
- It modifies the system to ensure **ongoing presence and execution** (persistence).

Persistence

- **Persistence** means malware remains **active in the background**, attempting to run:

- After every **system reboot**
- At specific **times of day**

- The most common method for persistence:
  - Dropping a copy of itself in a system folder
  - Creating an entry in the **Windows Registry**

Example: GlobeImposter Ransomware

- File dropped: `C:\Users\JuanIsip\AppData\Roaming\huVyja.exe`
- Registry key used for persistence: `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run`
- Registry value: `BrowserUpdateCheck`
- What matters: the **path** of the executable in the registry **(value name doesn't matter)**

Registry Keys Used for Persistence

Run Keys – Execute at Windows Startup

System-wide keys:

- `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run`
- `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce`
- `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnceEx`
- `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServices`
- `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\RunServicesOnce`
- `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run`
- `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Windows\CurrentVersion\Run`

User-specific keys:

- `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run`
- `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce`
- `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnceEx`
- `HKEY_CURRENT_USER\Software\Microsoft\WindowsNT\CurrentVersion\Windows\Run`

Note:

- Keys containing **"Once"** will execute programs only once.
- Malware can persist by repeatedly inserting its path into these keys.

Other Persistence Methods

Load and Run Values

Located under:
`HKEY_CURRENT_USER\Software\Microsoft\WindowsNT\CurrentVersion\Windows`

- `Load = <file path>`
- `Run = <file path>`

BootExecute Value

Located under:
`HKEY_LOCAL_MACHINE\SYSTEM\ControlSetXXX\Control\Session Manager`

- `BootExecute = <file path>`
- Default value: `autocheck autochk *`

Winlogon Key

- Location:
  `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\Winlogon`

  Values:

- `UserInit = <file path>`
- `Shell = <exe file path>` (default: `explorer.exe`)
- `Notify = <DLL path>` (dynamic link library triggered at logon)

- Similar path for current user:
  - `HKEY_CURRENT_USER\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\Winlogon`

Policy Scripts Keys

Trigger scripts at specific events:

- **Startup Scripts:**
  `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Group Policy\Scripts\Startup\0\N`
- **Shutdown Scripts:**
  `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Group Policy\Scripts\Shutdown\0\N`
- **Logon Scripts:**
  `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Group Policy\Scripts\Logon\0\N`
- **Logoff Scripts:**
  `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Group Policy\Scripts\Logoff\0\N`
- `Script = [file path of executable or script]`
- `N` is the index of multiple entries (starting from 0)

AppInit_DLLs Values

Located under:
`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows`

- `AppInit_DLLs = [list of DLLs]`
  (comma or space delimited)

- `LoadAppInit_DLLs = 1` → enabled
- `LoadAppInit_DLLs = 0` → disabled

Services Key

Located under:
`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\[ServiceName]`

- `ImagePath = [path to .sys or .dll file]`
  Used to load a driver or system file

Trigger values:

- `0` → SERVICE_BOOT_START (during OS load)
- `1` → SERVICE_SYSTEM_START (during OS init)
- `2` → SERVICE_AUTO_START (when service manager starts)
- `3` → SERVICE_DEMAND_START (manual start)
- `4` → SERVICE_DISABLED

File Associations

Located at:

- `HKEY_CLASSES_ROOT`
- or
  `HKEY_LOCAL_MACHINE\SOFTWARE\Classes\[file extension]\shell\open\command`

Example:

- For `.exe` files:
  `HKEY_LOCAL_MACHINE\SOFTWARE\Classes\exefile\shell\open\command`

Startup Folder (Shell Folders)

- Files placed here run after user login:
  - `%APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup`

Registry entries:

- `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders`
- `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders`
- `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders`
- `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders`

Image File Execution Options (IFEO)

Used to hijack legitimate executables via the **Debugger** field.

Location:
`HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\[ProcessName]`

- `Debugger = [executable file]`
- Only triggers if **debugging** is invoked

Browser Helper Objects (BHO)

Location:
`HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects\[CLSID]`

- CLSID refers to the registered BHO component
- Related entry:
  `HKEY_CLASSES_ROOT\CLSID\[CLSID]\InprocServer32`
  - Default value points to the **DLL file**

- The DLL is loaded whenever **Internet Explorer** is launched

---

Scheduled Tasks & Cron Jobs

- Malware can also persist through:
  - **Windows Task Scheduler**
  - **Cron jobs** (in Unix systems)
- These allow execution **based on time** or **certain system conditions**

Let me know if you want:

- A **mind map** or visual diagram
- A **PDF or DOCX** export
- A **quiz or flashcards** to test understanding