Vulnerability Scanning and Applicable Patch Management

Product Design Document

Kapil Rajesh Kavitha - 2021101028

Harshavardhan P - 2021111003

Introduction

Unpatched software is one of the primary reasons for security breaches. In many cases, identified vulnerabilities are not being patched by users, making their systems an easy target for malicious attackers.

These breaches can be mitigated by implementing an effective vulnerability management process, which includes regularly scanning the IT assets on a system for vulnerabilities, and letting the user know about the identified vulnerabilities and their patches. In this document, we aim to describe a utility that performs this task.

System Overview

Provide a general description and functionality of the software system. Describe some potential real-life scenarios of people actually using your product. How will they use it and what problem would it solve?

The system is a Python-based CLI application which:

1. Scans the assets on a user's system (such as applications and packages)
2. Sends the list of scanned assets to an **Express** server in the backend
3. Using the server, checks the National Vulnerability Database (NVD) for vulnerabilities corresponding to these assets
4. Lists the vulnerabilities and allows the user to select which ones he/she would like to fix
5. Performs the following **hierarchical update search** operation to display patch/updates to user:
    a. Checks the UpdateStar repository for the vulnerable application.
    b. Checks the SourceForge repository for the vulnerable application.
    c. Returns the Google Search page with the search results for the query *"Patch for <application> <version>"*

The process of asset detection and vulnerability identification is triggered whenever the application is launched by the user. The user is given a numbered list of vulnerabilities identified, and has the option to select the vulnerabilities to be updated or patched.

The system is intended for use by any user who is comfortable navigating a command-line interface, regardless of their experience with patch and vulnerability management.

Some use cases would include:

- An IT security professional using this system during regular maintenance checks, to identify vulnerabilities in installed business applications and apply patches to critical systems.
- A home user who is concerned about data breaches, and periodically launches the system to check for vulnerabilities present in their system.

Implementation Info

Currently, the system only supports operations on Windows-based systems, but the **modularity** provided in the application ensures that the system can be smoothly extended to other operating systems in the future.

The hierarchical update search operation can be modified depending on the operating system and the update search mechanism which would be required for that particular OS.

Design Overview

Architecture and Component Interaction

The system has three major components: an asset scanner, a user interface through a Python CLI tool, and a centralized server with access to the NVD database. The main components explained:

1. **Asset Scanner**:
    - Implemented in **Python**
    - Scans the user's device to gather a list of installed applications along with their versions.
        - In Windows, the list of installed applications is obtained using **Powershell** commands such as **Get-Package** and **Get-WmiObject.**
    - **Functionality**:
        - **System Scan Command**: Users can initiate an on-demand scan from the CLI, which will allow the system to discover assets, gather pertinent information, and send that information to the **NVD Server** so that it can fetch matching vulnerabilities.

2. **Python CLI Interface**:
    - **Objective**: This is a terminal command line application by which the final user interacts with the application instead of a graphical user interface. Users tend to start scans; view threats detected and select patches through a set of commands and options.
    - **Key Functionalities**:
        - **Scan Command**: The command initiates the scanning of your assets, prompting the **Asset Scanner** to identify all installed software and packages.
        - **Display Vulnerabilities**: The CLI displays detected vulnerabilities after scanning. Each vulnerability is shown in a list format, which includes:
            - Severity (via CVSS scores)
            - Description and prerequisites for patching
            - Any dependencies
        - **Identify Vulnerabilities to Patch**: Users can browse vulnerabilities in the list using commands specific to select specific, individual ones that shall be patched.
3. **NVD Server**:
    - **Role**: A backend server that acts like a repository of vulnerability data, where detected assets are matched with the vulnerabilities of the National Vulnerability Database.
    - **Components**:
        - **Vulnerability Database**: All details for each vulnerability, such as affected versions, CVSS scores, and recommended fixes are included in this.
        - **Query Interface**: This is an API that processes requests from the CLI to fetch vulnerability data for assets identified by the Asset Scanner.
        - **Automatic Daily Updates Provision**: The process is automation-based and keeps on synchronizing the database with the latest data that exists in NVD, thus making the system up-to-date regarding the latest vulnerability information.

Data Flow

1. The **Asset Scanner** can relay a list of detected assets to the **CLI** when the scan is complete.
2. The **CLI** makes a query to the **NVD Server** to retrieve information on vulnerabilities for each of the assets.
3. The **NVD Server** responds back with the information pertaining to vulnerability that is then printed out on the CLI and is viewed by the users.
4. The user must choose the vulnerable applications for which they would like to apply the patches.
5. The CLI then directs the user to the page(s) where the patch can be downloaded.

- **NVD Server to Vulnerability Database**: Intra-node database query to send back data regarding the presence of the existence of vulnerabilities, updated daily from the NVD.
- **Asset Scanner to CLI**: Realtime communication/data sharing where the CLI shows the asset information scanned by the **Asset Scanner**.

System interfaces

User Interface

Explain how the user will be able to use your system to complete all the expected features. Add screenshots if possible. If this is not applicable to your project, you may omit it.

Upon running the application, the first step is the detection of the OS present in the user's system.

```
C:\Users\Kapil\Desktop\Y4S1\SQE\project>python src/main.py
Detected Windows OS
```

After this, the assets present in the user's system are scanned in the background. Once this is done, the application then begins.

```
Searching for vulnerable packages...
```

In the backend server:

```
Analyzing Update for x64-based Windows Systems (KB5001716) 8.94.
0.0...
Analyzing Python 3.11.3 Tcl/Tk Support (64-bit) 3.11.3150.0...
Analyzing Dell SupportAssist OS Recovery Plugin for Dell 5.5.9.1
8923...
Analyzing vs_minshellsharedmsi 17.5.33310...
Analyzing Webex 43.10.0.28042...
Analyzing Dell SupportAssist OS Recovery Plugin for Dell 5.5.9.1
8923...
Analyzing vs_filehandler_x86 17.5.33310...
Analyzing Python 3.11.3 Documentation (64-bit) 3.11.3150.0...
Analyzing Microsoft OneDrive 24.211.1020.0001...
Analyzing WinFsp 2023 2.0.23075...
Analyzing NVIDIA CUDA Nsight NVTX 11.7 11.7...
Analyzing Microsoft .NET Host FX Resolver - 6.0.28 (x64) 48.112.
10439...
Analyzing Microsoft .NET Framework Cumulative Intellisense 4.8.0
9037...
Analyzing NVIDIA CUDA Visual Studio Integration 11.7 11.7...
Analyzing WinRAR 6.21 (64-bit) 6.21.0...
Analyzing VS Script Debugging Common 17.0.118.0...
```

Results after all assets are processed:

```
7. Cisco Webex Meetings (43.7.0) - Severity: 10
8. Discord (1.0.9012) - Severity: 0
9. MongoDB Compass (1.44.6) - Severity: 9.8
10. Slack (4.41.97) - Severity: 7.8
11. MySQL Server 8.0 (8.0.36) - Severity: 9.8
12. Dell SupportAssist (4.0.3.61632) - Severity: 8.8
13. MSYS2 (20230318) - Severity: 7.8
14. Node.js (22.8.0) - Severity: 0
15. Brave (131.1.73.89) - Severity: 0
16. Google Chrome (131.0.6778.69) - Severity: 9.8
17. MySQL Shell (8.0.37) - Severity: 2.5
18. IntelliJ IDEA 2023.3.3 (233.14015.106) - Severity: 7.5
19. Microsoft Edge (131.0.2903.51) - Severity: 9.8
20. Archi 5.3.0 (5.3.0) - Severity: 0
21. qBittorrent (4.5.2) - Severity: 9.8
22. Steam (2.10.91.91) - Severity: 0
23. Webex (43.10.0.28042) - Severity: 0
24. MongoDB Tools 100 (100.9.4) - Severity: 6.5
25. MySQL Server 8.0 (8.0.36) - Severity: 9.8
26. Dell SupportAssist (4.0.3.61632) - Severity: 8.8
27. MySQL Shell (8.0.37) - Severity: 2.5
Select packages to search for updates. If you want to select mul
tiple packages, separate them by commas. Enter 0 to exit: 21, 23
```

```
Select packages to search for updates. If you want to select mul
tiple packages, separate them by commas. Enter 0 to exit: 21, 23

Searching for updates for package: qBittorrent
Checking UpdateStar for updates for: qBittorrent
Searching UpdateStar for updates for: qBittorrent
200
Results found.
Performing fuzzy search on UpdateStar results...
Best match found on UpdateStar: https://qbittorrent.updatestar.c
om
Searching for updates for package: Webex
Checking UpdateStar for updates for: Webex
Searching UpdateStar for updates for: Webex
200
Results found.
Performing fuzzy search on UpdateStar results...
Best match found on UpdateStar: https://cisco-webex-meetings.updatestar.com
Opening link: https://qbittorrent.updatestar.com
Opening link: https://cisco-webex-meetings.updatestar.com
Select packages to search for updates. If you want to select multiple packages, separate them by commas. Enter 0 t
o exit: 0

C:\Users\Kapil\Desktop\Y4S1\SQE\project>
```

Hierarchical search is demonstrated in the last picture.

Model

Abstract classes

Class Name	Class State	Class Behaviour
OSInterface	**Responsibility**: Serves as a base class for OS-specific implementations, maintains: a placeholder set of installed packages (installed_pkgs)	`__init__`: Initializes the `installed_pkgs` set. `scan_pkgs` (abstract): Method to scan for installed packages. `exec_command` (abstract): Method to execute a system command. `translate_cmd` (abstract): Method to translate commands to OS-specific formats.
UpdateSearcher	**Responsibility:** an abstract class for searching updates for packages, which also defines behaviors for subclasses.	`search_for_updates` (abstract): Method to search for updates for a given list of `VulnerablePackage` instances. `open_links` (abstract): Method to open provided update links in a web browser.

.

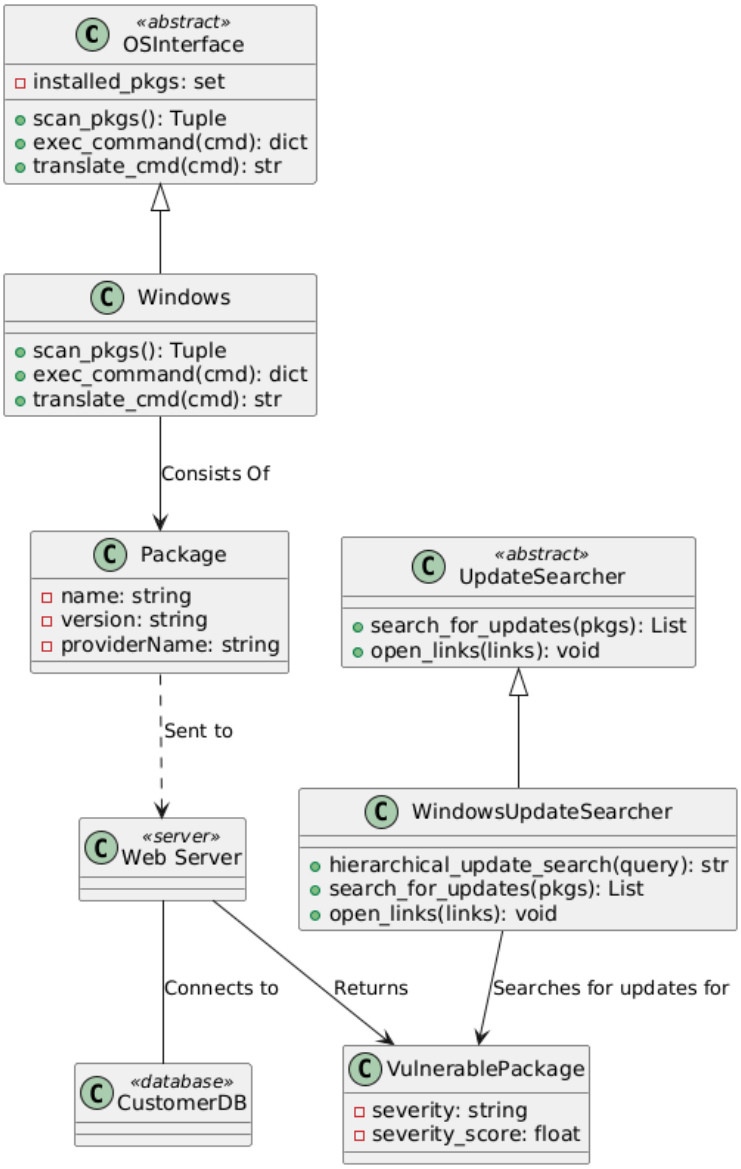
Other Classes

Class Name	Class State	Class Behaviour
Package	**Responsibility**: Maintains information about a package a. `name`**:** The package name b. `version`**:** The package version c. `providerName`**:** The package provider name	`__init__`: Initializes an instance of the class with the provided values as attributes.
VulnerablePackage	**Responsibility:** maintains information about a vulnerable package	`__init__`: Initializes the attributes with provided values.

	a. `name`: The package name. b. `version`: The package version. c. `providerName`: The package provider name. d. `severity`: The severity level of the vulnerability. e. `severity_score`: A numerical score representing the vulnerability's severity.	
Windows (a subclass of **OSInterface**)	**Responsibility:** Maintains a set of installed packages (`installed_pkgs`), which includes package names, versions, and providers detected on a Windows system.	**`__init__`**: Initializes the `installed_pkgs` set. **`scan_pkgs`**: Scans for installed packages using PowerShell commands (`Get-Package` and `Get-WmiObject`), parses the results, and populates the `installed_pkgs`. **`exec_command`**: Executes PowerShell commands and returns the command's result, including output and status. **`translate_cmd`**: Provides translation or customization of commands for Windows. **`parse_get_package`**: Parses package data from `Get-Package` output. **`parse_get_package_wmi`**: Parses package data from `Get-WmiObject` output.
WindowsUpdateSearcher (a subclass of **UpdateSearcher**)	**Responsibility:** Maintains settings and methods for searching and retrieving update links for vulnerable packages. Fuzzy matching is performed using the vulnerable application name and the existing applications in the repository. a. `threshold`: An integer that defines	`__init__(self, threshold: int = 70)`: Initializes the `WindowsUpdateSearcher` instance with a fuzzy match threshold (default is 70). `hierarchical_update_search(self, query: str) -> str`: Searches for updates for a given query using a hierarchy of sources: 1. UpdateStar: Scrapes results and performs fuzzy matching to find

	the relevance threshold for fuzzy matching during searches.	relevant links. 2. SourceForge: Extracts project links and applies fuzzy matching for relevance. 3. Google Search (Fallback): Constructs and returns a Google search URL if no relevant matches are found in the above sources. 4. Returns the best match URL or a Google search link as a fallback. `search_for_updates(self, packages: List[VulnerablePackage]) -> List[str]`: Iterates over a list of `VulnerablePackage` instances. Uses `hierarchical_update_search` to find update links for each package. Returns a list of URLs pointing to potential updates. `open_links(self, links: List[str])`: Opens the provided list of URLs in a web browser. Includes a short delay (`time.sleep(1)`) between opening each link to ensure smooth user experience.

Class Diagram

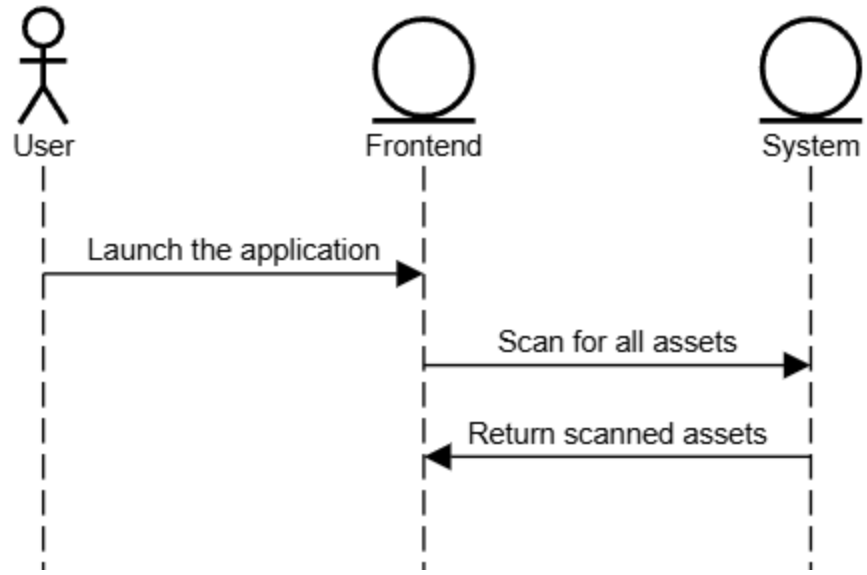


Sequence Diagram(s) for Use Cases

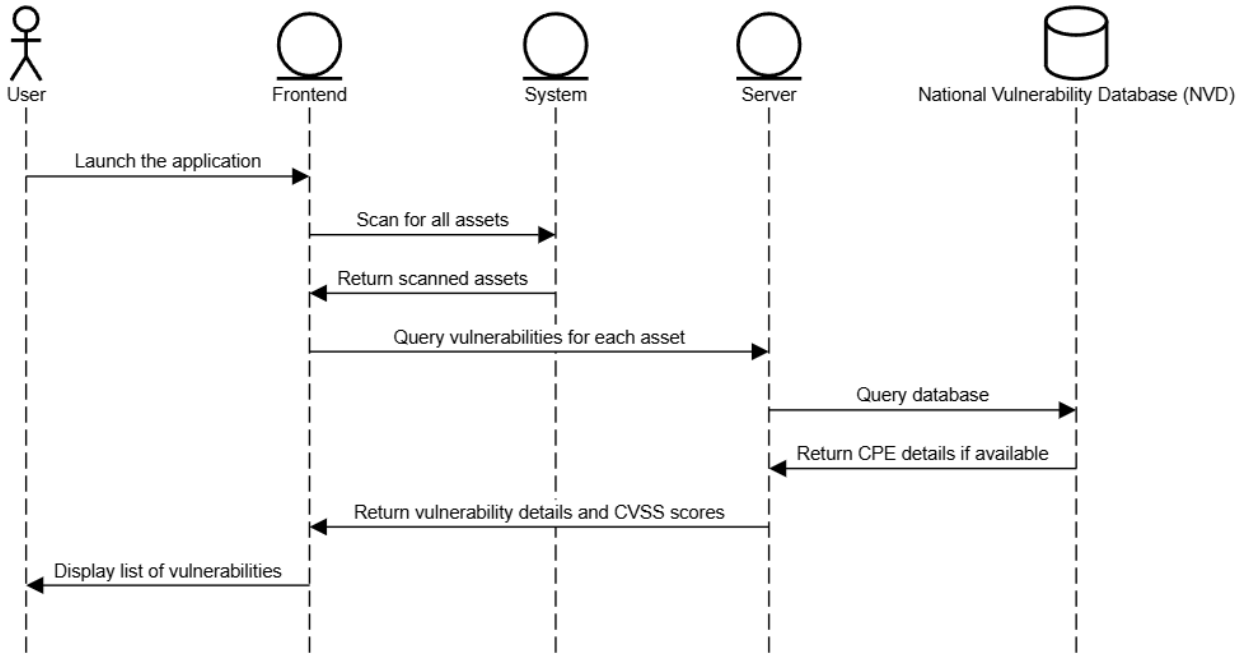
UML sequence diagram for the agreed significant use cases (about 4 major use cases).

*While you may reference an external file here, most instructors **strongly** prefer embedded images, or, failing that, external files in generic formats such as JPEG, PNG or PDF.*

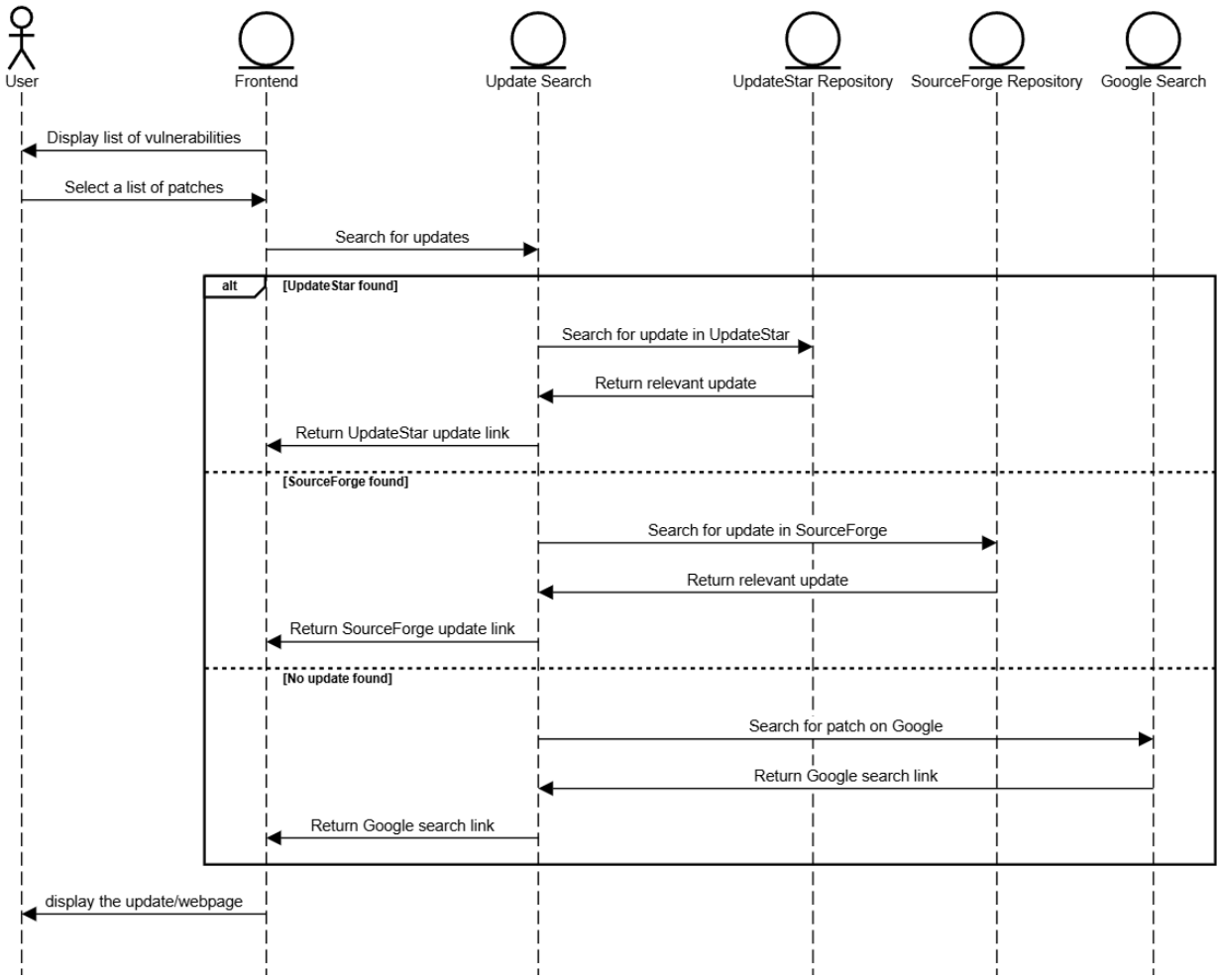
1. **Asset Detection**



2. **Vulnerability Identification and Notification**



3. **Patch selection and link/page retrieval**



Design Rationale

Why CLI Interface?

The transition into a CLI is very light and flexible, requiring no graphical resources or a web server for its installation. Designing a CLI is suitable for **end users**, who, like IT administrators and technically skilled users, feel comfortable in command-line environments. Installation and usage on different operating systems are simplified, thus fostering scalability and flexibility across different environments.

Security and Privacy

By their nature, CLI-based applications minimize potential attack vectors against web interfaces, because this application is not exposed to the network to as large an extent as usual. All

communication between CLI and **NVD Server** may be handled by HTTPS; vulnerability data will be locally stored only for a short time before being deleted when the CLI session is terminated.

Update Mechanism

It is essential to maintain the latest version of the **Vulnerability Database** in this system. The auto-update mechanism at the **NVD Server** assists in lessening the maintenance task, so the users are scanning against the latest possible data without any kind of intervention from the user's end.

User-Centric Design Choices

The CLI is structured in such a clear manner that commands like scan, list-vulnerabilities, and patch-select are intuitive, as follows:

- The process of the **Scan Command** is made easy and in no way needs actual user input. The whole procedure goes in well with that of IT security workflows.
- The **Display Vulnerabilities** feature provides detailed exposure of the vulnerabilities. Then, it will be allowed to analyse the risk created by the vulnerability according to its severity and dependencies.
- The **Patch Selection** process is interactive order wise and very simple that users can confirm or amend their choices before applying those patches.

Scalability and Maintenance

This design is scalable due to the modular nature of the CLI and its interaction with the **NVD Server**. If the database of vulnerabilities grows in size, there is no negative impact on the CLI experience since the **NVD Server** can handle it. The command-line environment also requires very few resources so is easy to maintain and operate even on resource-constrained systems.

Patch Retrieval and Application Challenges

The patch retrieval and application process poses significant challenges due to **the inconsistent and varied mechanisms used by different software vendors and application types**. Currently, the utility **directs users to official web pages or sites** for patch downloads, rather than handling the patch downloads directly. This approach is due to the **lack of a standardized patch-hosting method** across vendors—some offer centralized download portals, while others rely on individual product pages. Attempting to programmatically determine patch locations across these disparate systems has proven complex and unreliable, making manual redirection the most practical solution.

Furthermore, applying patches consistently across diverse applications presents additional complexity. Different applications have varying update requirements: some CLI tools or system

utilities support direct command-line patch installations, while others necessitate manual downloads or in-app updates that require user interaction. In-app updates, in particular, often require the program to be opened and sometimes involve authentication, adding further complexity to automated patch application. This variation in patching methods complicates the creation of a single automated solution, as it must accommodate multiple update paths and user-specific settings.

To address these challenges, the patch management feature focuses on providing users with guidance and appropriate links to official pages for each patch, allowing them to initiate updates manually. This approach strikes a balance between usability and practicality, especially for applications that require in-app interaction or unique update procedures. Where possible, we aim to offer flexibility by supporting command-line patch scripts for CLI-compatible applications while guiding users on in-app updates for others, though this approach involves more user interaction and maintenance.