A Tool for Learning About Outdated Software Vulnerabilities and End-of-Life*

Md Amiruzzaman†, Linh Ngo, and Ashik Ahmed Bhuiyan

West Chester University, West Chester, USA
{mamiruzzaman,lngo,ABhuiyan}@wcupa.edu

Abstract

This study presents a tool for enhancing E-Business Information Systems (EBIS) resilience by identifying outdated operating systems and installed applications on offline computers. While most EBIS vulnerability-management tools rely on network connectivity, our approach extends protection to isolated or legacy business environments through the development of a two-fold solution: a scanner app that reads and stores software information from individual computers, and a summary app that combines this data with software versions, Common Vulnerabilities and Exposures (CVE), and End-of-Life (EOL) support details from vendors. The summary app integrates EBIS data flows with machine-learning models that synthesize CVE descriptions, generating executive summaries to support managerial risk decisions.

Keywords: CVE, end-of-life, executive summary, machine learning, outdated software.

1 Introduction

As digital transformation accelerates across E-Business Information Systems, the risks of cybersecurity issues increase proportionally. Cyberattacks are a constant threat, and cybersecurity professionals work to mitigate these risks [11]. Studies show that cybersecurity risks rise as software becomes outdated [14], [19],[21]. Outdated applications no longer receive vendor support, meaning no bug fixes, patches, or updates. Outdated software can lead to serious vulnerabilities [1], leaving users and institutions unprotected. Attackers often target machines running outdated operating systems or applications, knowing that no patches will fix existing vulnerabilities [7]. Risks associated with outdated software include (a) poor performance; (b) compliance issues; (c) complete system failure; (d) bugs; and (e) increased susceptibility to cyberattacks. Organizations with constrained IT budgets, including educational institutions and small-to-medium enterprises operating EBIS platforms, often rely on outdated software [9],[2], which exposes them to cyberattacks like ransomware. Reasons for using outdated systems include preserving difficult-to-migrate data or relying on legacy software that meets specific business needs.

Typically, these outdated computers are used infrequently and kept disconnected from networks to reduce cyberattack risks. However, sensitive data transferred via USB drives to networked computers can still pose security threats. Tracking software installed on outdated systems is challenging. Existing tools like Endpoint Configuration Manager (ECM) and LAN Sweeper are network-based and can only scan connected devices. Tools like Rapid 7 and Tenable also focus on systems connected to the Local Area Network (LAN) or the Internet, making

*Proceedings of The 2025 IFIP WG 8.4 International Symposium on E-Business Information Systems Evolution (EBISION 2025), Article No. 13, December 16-18, 2025, Sapporo, Japan.

†Corresponding author

them ineffective for computers that lack connectivity. In the context of EBIS, outdated or unsupported software threatens business continuity, data integrity, and regulatory compliance. Our work contributes to the EBIS field by providing an architecture and prototype that bridge isolated legacy systems with contemporary vulnerability intelligence (CVE/EOL), thus enabling more informed e-business decision-making.

2 Background and Related Work

Literature identifies six major cybersecurity threats: (1) phishing; (2) malware; (3) man-in-the-middle attacks; (4) identity and credential theft; (5) human factors, and (6) outdated software [3],[14]. Existing works focus on reading information about outdated software, such as operating systems [13] and applications [12], and they primarily target Internet-connected systems.

Fan, Mao, Lu, and Wu [6] developed a cloud-based system called Offline Patching Scheme (OPS) to detect outdated software on cloud-connected computers and to provide patches for applications offline. Similarly, [10] employed a deep learning model to prioritize application upgrades based on vulnerabilities. This approach does not focus on collecting real-time information from outdated computers; instead, it creates a database of installed software and their vulnerabilities, helping to determine which applications need immediate attention. It is also noteworthy here that there are some other existing tools that can help in collecting installed software names and versions, such as LANsweeper and Rapid 7. However, they only work if a computer is connected to the Internet [17].

Our work addresses these limitations by targeting computers with outdated operating systems and software. The main challenge is collecting information about installed software on computers that are not connected to the Internet.

2.1 Software versioning

Every software system is identified by its name and a corresponding version number [8]. The software version helps to separate one version from another, as it indicates the changes included in a particular software package. Often, software companies develop at least two versions of the same software package, such as (a) a working version (also known as the beta version)—that indicates that it is enhanced than the previous version and not yet rigorously tested, and (b) a stable version—that is ready for all users and rigorously tested by the software quality assurance team. By all means, a software version indicates some type of improvement as well as some other important information, such as order of release, degree of changes, and date of release.

2.2 Common Vulnerabilities and Exposures

Together, the name of a software and its version help to identify the software package or application uniquely. Therefore, Common Vulnerabilities and Exposures (CVE) is tied with each software name and its associated version [15]. CVE refers to security flaws that have been identified and proven by more than one source. Each CVE is identified by its CVE ID (also known as CVE number). CVE IDs are assigned by the CVE Numbering Authority (CNA). It comprises major software vendors, such as Cisco, IBM, Oracle, Red Hat, and Microsoft, as well as research institutions. The CVE ID format includes the year when the vulnerability was discovered, such as CVE-0123-456789. So, a CVE ID can be 13 characters wide. The first three are fixed letters, the second four digits are for the year the vulnerability is accepted, and the rest are a sequence number.

2.3 Term Frequency-Inverse Document Frequency (TF-IDF)

The collected vulnerability descriptions can repeat some information and keywords. So, extracting the important keywords from each vulnerability description can help prepare a summary report for executives [18]. The TF-IDF algorithm is a good analysis tool to determine important words for a set of documents [16],[20].

$$\begin{aligned} \text{tf-idf}(t, D) &= \text{tf}(t, d) \times \text{idf}(t, D) \\ \text{tf}(t, d) &= f_{t|d} \\ \text{idf}(t, D) &= \log\left(\frac{N}{|\{d \in D : t \in d\}|}\right) \end{aligned}$$

where, $\text{tf}(t, d)$ function finds the frequency of a term (t) in a document (d), or the count of t given d. The $\text{idf}(t, D)$ function finds how many times a t appears in the total number of documents N, i.e., $N = |D|$. So, the $\text{tf-idf}(t, D)$ finds how important a term, t, in a document dataset, D.

3 Methodology

3.1 System overview

This study focuses on offline or minimally connected legacy computers that still participate in an organization's E-Business Information System (EBIS). While these computers may occasionally connect to a Local Area Network (LAN), their restricted connectivity and outdated operating systems limit the IT team's ability to collect information using contemporary Endpoint Configuration Manager (ECM) tools. As a result, the organization often lacks accurate data about installed software, versioning, and related risk exposure on these legacy nodes.

To address this gap, this work proposes a two-fold solution:

1. a *scanner app* deployed on legacy Windows and Linux hosts to collect operating system and installed software information, and

2. a *summary app* running on a modern server in the EBIS core to ingest scanner output, enrich it with Common Vulnerabilities and Exposures (CVEs) and end-of-life (EOL) information, and present host-level risk summaries.

The system architecture is organized into three interacting subsystems:

- the EBIS edge and legacy nodes, where local data collection occurs (Figure 1),

- the EBIS core network, where ingestion, enrichment, and reporting are performed (Figure 2), and

- external vulnerability and lifecycle intelligence sources, which provide CVE and EOL data (Figure 3).

3.1.1 EBIS edge: legacy nodes and scanner app

At the edge of the EBIS, the scanner app runs directly on outdated Windows and Linux machines. The key design constraint is that these hosts may be disconnected from the main EBIS network or subject to strict access controls. The scanner therefore relies only on local operating system facilities and produces a self-contained output file that can be transferred offline.

On Windows hosts (e.g., XP, Vista, 7, Server 2008, and 10), the scanner uses Java 8 and the Windows Management Instrumentation Command-line (WMIC) utility to extract:

- the operating system name and version, and
- the names and versions of installed applications or packages of interest.

The command

```
WMIC product where get Name, Version /FORMAT:CSV
```

returns a structured list of installed software and associated versions, which the scanner parses and writes to a CSV file.

On Linux hosts (e.g., Ubuntu 18.04–22.04), the scanner reads OS metadata from distribution release files under `/etc` and enumerates installed packages using distribution-specific mechanisms. In both environments, the output is standardized into records of the form:

Node, Caption, Version,

where *Node* is the computer name, *Caption* is the operating system or application name, and *Version* is its version number. The resulting file is then moved via a controlled transfer channel (e.g., USB drive or restricted file share) into the EBIS core.

This architecture is captured in Figure 1, which shows how the scanner modules interact with legacy hosts and produce scan report files for consumption by downstream components.

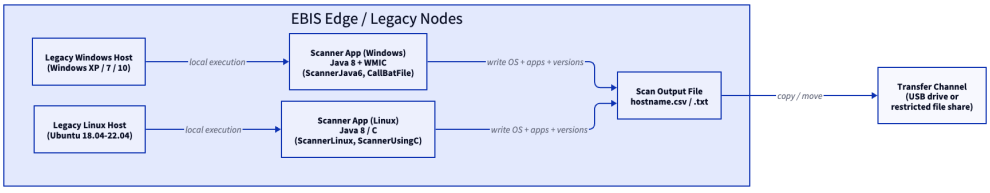

Figure 1: Scanner deployment and data collection on EBIS edge and legacy nodes. The scanner applications (Windows and Linux) run locally on each host, extract OS and software version information, and export results as flat files for secure transfer into the EBIS core network.

3.1.2 EBIS core: summary app and risk enrichment

Within the EBIS core, the summary app is implemented as a Python/Flask web application backed by a MySQL database. Authorized EBIS stakeholders (e.g., IT staff, security analysts,

and managers) access the app through a web interface to upload scanner output files, review host inventories, and inspect CVE/EOL-derived risk information. Upon upload, the summary app:

1. validates the file type and structure,
2. saves a copy in a server-side directory,
3. parses the contents into a tabular representation, and
4. inserts host–software tuples into the database with a generated Universally Unique Identifier (UUID) for each record.

The database schema is designed to support:

- a host and software inventory table (computer name, OS, installed application names, and version numbers),
- a CVE table capturing vulnerability identifiers, severity scores, and descriptions associated with specific software/version pairs, and
- an EOL table capturing end-of-life dates and support status for each software/version.

Figure 2 illustrates the core components of this subsystem, including the upload and validation controller, ETL (extract-transform-load) module, CVE and EOL integrators, TF-IDF summarizer, and the reporting interface. Together, they realize a pipeline that transforms raw scanner output into actionable EBIS risk intelligence.

3.1.3 External vulnerability and lifecycle sources

The summary app enriches local EBIS data by integrating two classes of external security intelligence (Figure 3):

- **CVE data:** Vulnerabilities associated with operating systems and installed applications are obtained from the CVE database [4], typically via publicly available NVD APIs. For each software/version pair, the system retrieves relevant CVE identifiers, severity metrics, and descriptive text.
- **EOL data:** End-of-life (EOL) information is obtained from dedicated EOL datasets [5] that map software names and versions to lifecycle milestones (e.g., end of support or extended support dates).

These integrations are encapsulated within dedicated modules in the EBIS core, preserving a clean separation between internal asset data and external intelligence feeds.

3.2 Data source

This work collects data from multiple sources:

- **Computer Details**: The computer name, operating system, and installed application names are sourced from the outdated computer(s).
- **Vulnerabilities**: Vulnerabilities associated with the operating system and its version, as well as installed applications, are obtained from the CVE database [4].
- **End-of-Life (EOL) Information**: EOL information for outdated software is sourced from the EOL database [5].

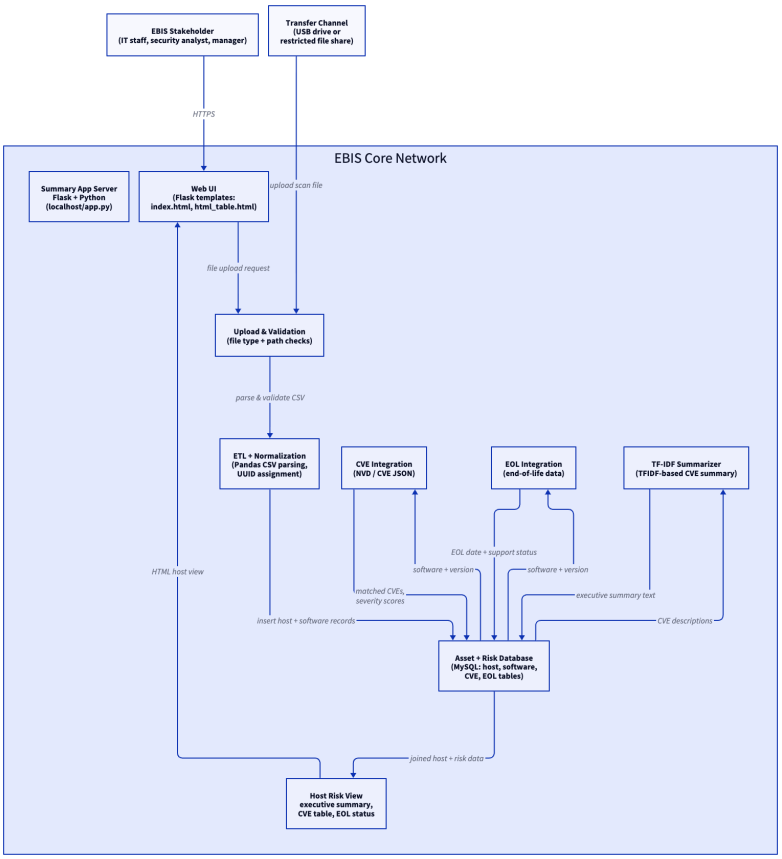

Figure 2: Summary application architecture within the EBIS core network. Uploaded scan files are processed through validation, ETL, CVE enrichment, EOL mapping, and TF-IDF summarization before being presented through the web interface.

3.3 Requirements

The design and implementation of the scanner and summary apps are guided by the following requirements, which also inform unit tests, integration tests, and the evaluation of the overall tool:

1. **Local inventory capture:** The tool must save a report of installed software and operating system information from legacy Windows and Linux hosts, without requiring network-based management agents.
2. **Database exportability:** The scan report should be exportable to a relational database to support persistent storage, querying, and integration with existing EBIS asset-management systems.
3. **Distinct software and lifecycle tracking:** The database must represent distinct soft-

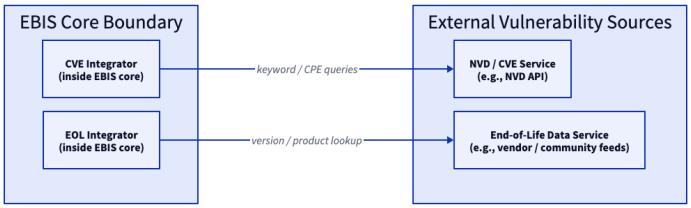

Figure 3: Integration between EBIS core vulnerability integrators and external intelligence sources, including NVD/CVE services and end-of-life data feeds. These services supply the system with real-time vulnerability and lifecycle information.

ware and corresponding versions, and associate them with end-of-life (EOL) information obtained from the EOL database [5].

4. **Vulnerability enrichment:** The tool should collect vulnerability information from the CVE database [4] for each relevant software/version pair and store it alongside local asset records.

5. **Text-based risk summarization:** The tool should analyze vulnerability information, focusing on a small set of recent CVE descriptions per host, and produce an executive summary using text analysis (TF-IDF).

3.4 System workflows and algorithms

To make the system behavior reproducible and implementation-agnostic, this subsection presents high-level algorithms corresponding to the EBIS edge, core, and summarization workflows.

Algorithm 1: Scanner app workflow (Windows and Linux)

Input: Legacy host H (Windows or Linux)
Output: Scan report file R containing OS and installed software with versions

1. Obtain the computer name by executing the host command (e.g., `hostname`) and store it as *hostname*.
2. Construct the output filename R as `$hostname.csv`.
3. If H is a Windows host:
   (a) Execute a WMIC query to obtain the OS caption and version.
   (b) Execute a WMIC query to obtain installed product names and versions of interest.
   (c) Write a header row `(Node, Caption, Version)` to R.
   (d) Append rows for the OS and each installed product, using *hostname* as the node name.

4. Else if H is a Linux host:
   (a) Parse distribution release files under `/etc` to determine OS name and version.
   (b) Enumerate installed packages using the distribution's package manager or configuration files.
   (c) Write a header row `(Node, Caption, Version)` to R.
   (d) Append a row for the OS and rows for each installed package, using *hostname* as the node name.
5. If a previous file with the same name exists, remove it to avoid stale data.
6. Return the completed file R for transfer to the EBIS core.

Algorithm 2: Summary app ingestion and enrichment

Input: Scan report file R uploaded via the web interface
Output: Persisted host and software records with CVE and EOL enrichment, and a host-level risk view

1. Receive the uploaded file R from an authenticated user.
2. Verify that the file extension is supported (e.g., `.csv` or `.txt`); reject otherwise.
3. Save a copy of R in a server-side upload directory.
4. Parse R into a tabular structure and normalize column names to `(Node, Caption, Version)`.
5. For each row in the table:
   (a) Sanitize the strings to prevent SQL injection and enforce length constraints.
   (b) Generate a UUID for the record.
   (c) Insert the tuple (UUID, node name, software name, version) into the host–software table.
6. Extract the distinct set of software/version pairs from the database.
7. For each distinct software/version pair:
   (a) Construct a query key based on the software name and version (e.g., keyword or CPE-style string).
   (b) Query the CVE service to retrieve relevant vulnerabilities.
   (c) Insert the returned CVE identifiers, severity scores, and descriptions into the CVE table, linked to the software/version pair.
8. Load EOL data from the external EOL dataset.
9. For each software record in the host–software table:
   (a) Find the best-matching EOL entry for the software name and version.
   (b) If a match is found, update the record with EOL date and support status.

10. For each host node:
    (a) Collect the CVE descriptions associated with that host.
    (b) Apply Algorithm 3 to compute an executive summary of the host's vulnerability profile.
    (c) Store the resulting summary text in a host-summary table.
11. When a user requests details for a host, join the host–software, CVE, EOL, and summary tables and render the combined view in the web interface.

Algorithm 3: TF-IDF-based executive summary of CVE descriptions

Input: A set of CVE descriptions $D = \{d_1, d_2, \ldots, d_N\}$ for a given host
Output: A concise executive summary S

1. For each document $d_i \in D$:
    (a) Remove stop words, articles, and punctuation.
    (b) Split d_i into sentences and collect the cleaned sentences into a list.
2. Let *sentences* be the list of all cleaned sentences across all documents.
3. For each sentence s in *sentences*:
    (a) Tokenize s into words.
    (b) For each word w in s, compute the term frequency $\mathrm{tf}(w, s)$ as the count of w in s divided by the total number of words in s.
4. For each distinct word w appearing in any sentence:
    (a) Let n_w be the number of sentences in which w appears.
    (b) Compute the inverse document frequency $\mathrm{idf}(w) = \log\left(\frac{|sentences|}{1+n_w}\right)$.
5. For each sentence s in *sentences*:
    (a) Initialize a score $\mathrm{score}(s) = 0$.
    (b) For each word w in s, update $\mathrm{score}(s) \leftarrow \mathrm{score}(s) + \mathrm{tf}(w, s) \times \mathrm{idf}(w)$.
6. Compute the average sentence score

$$\text{threshold} = \frac{1}{|sentences|} \sum_{s \in sentences} \mathrm{score}(s).$$

7. Select all sentences whose score is greater than or equal to *threshold*, preserving their original order.
8. Concatenate the selected sentences to form the executive summary S.

This algorithmic description refines and structures the TF-IDF-based summarization process introduced earlier, making explicit how multiple CVE descriptions are transformed into a short, human-readable summary for each host.

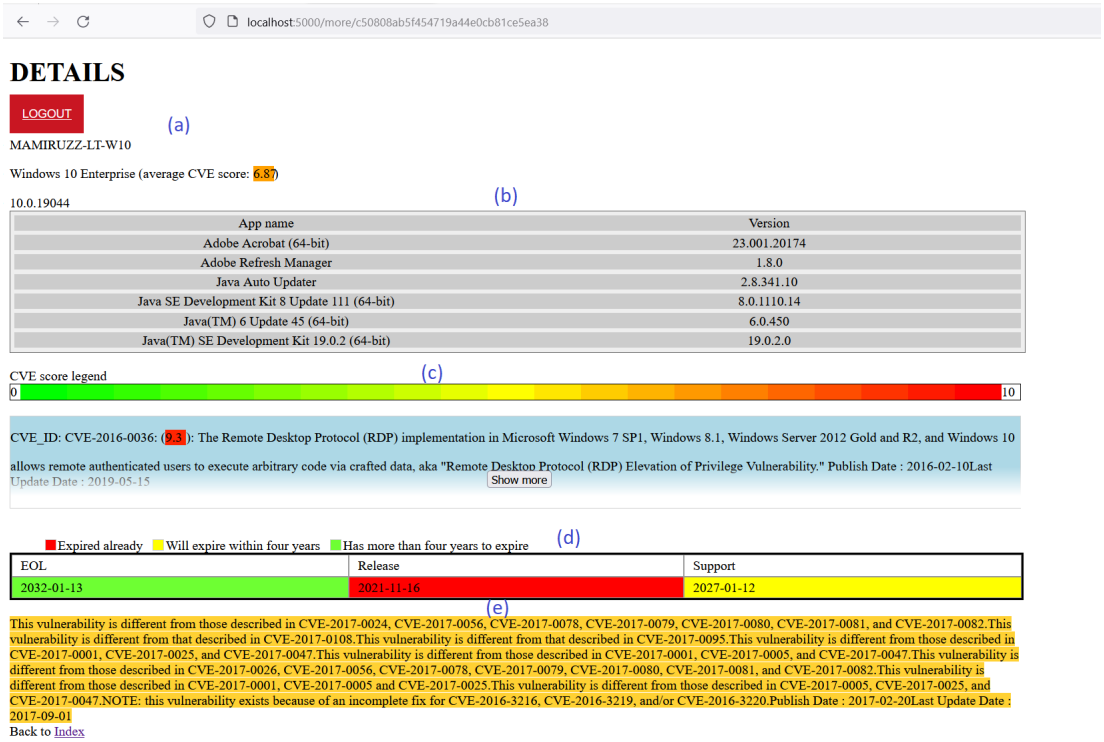

Figure 4: Implemented interface of the summary app displaying detailed information for a scanned machine.

4 Implementation

Following the EBIS architecture and workflows described in Section 3, the system implements two interoperable software modules: the scanner app deployed at the EBIS edge and the summary app deployed in the EBIS core. These modules operationalize Algorithms 1–3 and collectively support automated asset inventory, vulnerability enrichment, and executive-level risk summarization.

4.1 Scanner app

The scanner app is implemented in Java 8 to ensure compatibility with legacy Windows and Linux systems (XP, Vista, 7, Server 2008, and older Linux distributions). Its behavior follows Algorithm 1 and the architecture in Figure 1. The design goal is to collect reliable system metadata using only local operating system facilities, without requiring modern management agents or network connectivity.

Windows implementation. The scanner executes WMIC commands to extract OS metadata and installed software:

```
WMIC product where get Name, Version /FORMAT:CSV
```

Outputs are parsed and normalized into `(Node, Caption, Version)` tuples, then exported as a CSV report. This approach avoids registry parsing and ensures stable behavior across multiple Windows releases.

Linux implementation. On Linux hosts, the scanner reads distribution release files under `/etc` to identify OS name and version, and enumerates installed packages using distribution-specific mechanisms. Each entry is normalized using the same schema as Windows, ensuring platform-independent handling in the summary app.

A sample output is shown in Table 1. This file forms the core artifact consumed by the summary app.

Table 1: Sample output from scanned computer: (left side) name of the installed packages and (right side) version corresponding to each installed package/application.

Node	Caption	Version
AMIR_LT_W7	Microsoft Windows 7 Ultimate	6.1.7601
AMIR_LT_W7	Java 8 Update 202 (64-bit)	8.0.2020.8
AMIR_LT_W7	Microsoft .NET Framework 4.7.2	4.7.03062

4.2 Summary app

The summary app is implemented as a Python/Flask web application running on a modern EBIS core server. It follows the workflow of Algorithm 2 and the architecture in Figure 2. Its responsibilities include ingesting scanner files, performing CVE/EOL enrichment, storing normalized asset data, and producing host-level risk summaries.

Data ingestion and validation. Upon upload, the system verifies file type, stores a server-side copy, and parses the contents using Python's CSV utilities and Pandas. Each tuple is validated for proper formatting and SQL-sanitized before being inserted into the MySQL back-end. UUIDs are assigned to ensure each record is uniquely identifiable.

Database storage. The database schema contains three primary tables:

- a host–software inventory table (scanner output),
- a CVE table (vulnerability enrichment), and
- an EOL table (lifecycle status).

These tables collectively support complex queries for risk inspection, filtering, and historical analysis.

CVE and EOL integration. Following Algorithm 2, the summary app queries external CVE sources and matches software/version pairs to EOL datasets [5]. Retrieved CVE identifiers, severity scores, and descriptive texts are stored in the CVE table. Lifecycle metadata (e.g., supported, deprecated, EOL) is attached to each software entry.

TF-IDF executive summary. For each machine, the summary app extracts CVE descriptions and applies Algorithm 3 to produce a concise risk narrative. This summarization enables IT managers to understand host-level risk posture without reading lengthy CVE descriptions. The resulting summary is stored alongside other host metadata.

4.2.1 Interface design

The Flask-based interface presents the list of scanned computers and allows authorized users to access per-machine detail pages (Figure 4). Each detail view includes:

- the operating system and its average CVSS score,
- a list of installed applications and versions,
- matched CVEs with severity color-coding,
- EOL status indicators, and
- the TF-IDF-based executive summary.

Buttons for "Show more" allow users to expand long lists of vulnerabilities or EOL entries. Access control ensures only authorized personnel can view or delete scanned entries.

5 Evaluation

The developed tool is the first of its kind. There is no existing tool that does the same thing. Therefore, comparison of this tool's performance with existing apps is not feasible. However, to ensure its success, accuracy, and performance, a series of unit-tests and integration tests were performed.

5.1 Unit testing

While developing the tool, each functional unit was divided into single methods. This helped to focus on individual tasks more carefully and also test them individually. A set of unit tests were written to check the functionality of each method. Both success cases and failure cases were tested to ensure the performance of the methods (see Tables 2 and 3). This was done for both the scanner app and the summary app.

Table 2: Unit test for scanner app

Unit test	Status
Read computer name	☑
Read Operating system name and version	☑
Read installed application names and version	☑
Save the output to a file for the summary app	☑
Delete previous file when the scanner app is ran again	☑

5.2 Integration testing

The developed tool was evaluated by testing the tool on different operating systems. For example, the scanner app (see Sec 4.1 for more details) was tested on several older operating systems than Windows 10 (see Table 4 for a complete list). This provided an opportunity to understand how the tool can capture information from outdated computers.

Table 3: Unit test for summary app

Unit test	Status
Check supported file type	☑
Save uploaded scanner file	☑
Read installed application names and version	☑
Insert information read from the file	☑
Delete previously saved records	☑
Read CVE information	☑
Read EOL information	☑
Get executive summary from texts	☑

Table 4: Completed simulation on different OS, version, and environment

OSs	Version	Environment
Windows 10	10.0.19045	Real
Windows 10	10.0.19044	Real
Windows 7 SP1	6.1.7601	VM
Windows Vista	6.0.6000	Real
Windows XP	5.1.2600	Both real and VM
Ubuntu	22.4	VM
Ubuntu	20.4	VM
Ubuntu	18.4	VM

Then, captured information was imported to get the summarized information (see Sec. 4.2). This helped to evaluate the success of the tool for different functions and features.

While the tool was tested on several older operating systems, several questions were posed to evaluate its complete integration. Questions were: (a) what operating system is the test computer running on? (the name of the OS was the answer) (b) How old the operating system is to the current date? (version number and EOL data helped to get answer) (c) What are the packages or applications running on the test computer? (list of installed applications and their version numbers were the answer)(d) What are the vulnerabilities related to those packages or applications? (list of CVE description and severity score were the answer)(e) When is the end-of-life of the operating system for the test computer? (EOL data was the answer)

Also, each step, including the ones mentioned in Sec. 4.2 were tested using Python-based integration tests to evaluate their expected behavior (see Tables 5).

Table 5: Integration test for summary app

Check login based on user database	☑
Insert the scanned information to app database	☑
List all the scanned computers from the app database	☑
Provide details for a specific scanned computer	☑
Download CVE database and save as a json file	☑
Get the executive summary from CVE descriptions	☑
Perform summary based on specific query of scanned computer	☑

6 Conclusion

This work advances E-Business Information Systems security by enabling accurate identification of OS and application metadata on legacy or offline computers, facilitating proactive risk assessment and lifecycle management. These computers are not connected to the Internet, which makes utilizing existing modern online tools challenging. Furthermore, the collected information is complex and requires significant effort to analyze. Within the EBIS domain, the proposed architecture demonstrates how machine-learning-based summarization and standardized vulnerability data (CVE/EOL) can be operationalized into decision dashboards for business stakeholders.

This work approaches the above problem using a two-fold solution. One to collect information from an outdated computer and another to combine and present the information. The project leveraged the CVE, EOL databases, and machine learning techniques to summarize vulnerability information to prepare an executive summary. Future work will extend interoperability with enterprise resource planning (ERP) modules and automate patch-planning workflows, reinforcing the EBIS ecosystem's overall resilience.

References

[1] Alfadel, M., Costa, D.E., Shihab, E.: Empirical analysis of security vulnerabilities in python packages. Empirical Software Engineering **28**(3), 59 (2023)

[2] Betz, J., Betz, T., Fent, F., Geisslinger, M., Heilmeier, A., Hermansdorfer, L., Herrmann, T., Huch, S., Karle, P., Lienkamp, M., et al.: Tum autonomous motorsport: An autonomous racing software for the indy autonomous challenge. Journal of Field Robotics **40**(4), 783–809 (2023)

[3] Bojjagani, S., Sastry, V., Chen, C.M., Kumari, S., Khan, M.K.: Systematic survey of mobile payments, protocols, and security infrastructure. Journal of Ambient Intelligence and Humanized Computing **14**(1), 609–654 (2023)

[4] CVE Details: The ultimate security vulnerability datasource. https://www.cvedetails.com/vulnerability-list/ (2023), accessed: 2023-5-26

[5] endoflife.date: End-of-life (eol) and support information. https://endoflife.date/ (2023), accessed: 2023-5-26

[6] Fan, K., Mao, D., Lu, Z., Wu, J.: Ops: Offline patching scheme for the images management in a secure cloud environment. In: 2013 IEEE International Conference on Services Computing. pp. 587–594. IEEE (2013)

[7] Gonzalez-Barahona, J.M., Sherwood, P., Robles, G., Izquierdo, D.: Technical lag in software compilations: Measuring how outdated a software deployment is. In: Open Source Systems: Towards Robust Practices: 13th IFIP WG 2.13 International Conference, OSS 2017, Buenos Aires, Argentina, May 22-23, 2017, Proceedings 13. pp. 182–192. Springer International Publishing (2017)

[8] Gupta, D., Jalote, P., Barua, G.: A formal framework for on-line software version change. IEEE Transactions on Software engineering **22**(2), 120–131 (1996)

[9] Harrell, C.R., Patton, M., Chen, H., Samtani, S.: Vulnerability assessment, remediation, and automated reporting: Case studies of higher education institutions. In: 2018 IEEE International Conference on Intelligence and Security Informatics (ISI). pp. 148–153. IEEE (2018)

[10] Hore, S., Shah, A., Bastian, N.D.: Deep vulman: A deep reinforcement learning-enabled cyber vulnerability management framework. Expert Systems with Applications **221**, 119734 (2023). https://doi.org/https://doi.org/10.1016/j.eswa.2023.119734, https://www.sciencedirect.com/science/article/pii/S095741742300235X

[11] Lee, I.: Cybersecurity: Risk management framework and investment cost analysis. Business Horizons **64**(5), 659–671 (2021)

[12] Narayana Samy, G., Ahmad, R., Ismail, Z.: Security threats categories in healthcare information systems. Health informatics journal **16**(3), 201–209 (2010)

[13] Nunkesser, R., Thorn, J.: Possibilities and limitations of mobile applications for controlling. In: The Digitalization of Management Accounting: Use Cases from Theory and Practice, pp. 243–261. Springer (2023)

[14] Park, C., Kontovas, C., Yang, Z., Chang, C.H.: A bn driven fmea approach to assess maritime cybersecurity risks. Ocean & Coastal Management **235**, 106480 (2023)

[15] Pham, V., Dang, T.: Cvexplorer: Multidimensional visualization for common vulnerabilities and exposures. In: 2018 IEEE International Conference on Big Data (Big Data). pp. 1296–1301. IEEE (2018)

[16] Qorib, M., Oladunni, T., Denis, M., Ososanya, E., Cotae, P.: Covid-19 vaccine hesitancy: Text mining, sentiment analysis and machine learning on covid-19 vaccination twitter dataset. Expert Systems with Applications **212**, 118715 (2023)

[17] Sami, M., Bhatti, S., Baloch, J., Shah, S.: How to keep your systems records safe. International Journal of Information Technology **11**, 287–293 (2019)

[18] Simon, V., Rabin, N., Gal, H.C.B.: Utilizing data driven methods to identify gender bias in linkedin profiles. Information Processing & Management **60**(5), 103423 (2023)

[19] Vasek, M., Moore, T.: Identifying risk factors for webserver compromise. In: Financial Cryptography and Data Security: 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers 18. pp. 326–345. Springer (2014)

[20] Yahav, I., Shehory, O., Schwartz, D.: Comments mining with tf-idf: the inherent bias and its removal. IEEE Transactions on Knowledge and Data Engineering **31**(3), 437–450 (2018)

[21] Zerouali, A., Cosentino, V., Mens, T., Robles, G., Gonzalez-Barahona, J.M.: On the impact of outdated and vulnerable javascript packages in docker images. In: 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER). pp. 619–623. IEEE (2019)