Judging a Book by Its Plugins: A Static-Analysis Security Evaluation of Calibre

Hatcher, Collin
Middle Tennessee State University
Computer Science
Murfreesboro, USA
ch9x@mtmail.mtsu.edu

Rodriguez, Kevin
Middle Tennessee State University
Computer Science
Murfreesboro, USA
kar6r@mtmail.mtsu.edu

***Abstract*—This document provides a comprehensive security assessment of Calibre, an open-source e-book management software. Our assessment focuses primarily on the plugin architecture to uncover potential security vulnerabilities. We use formal threat modeling techniques like STRIDE and DREAD, and we conduct a hands-on assessment using static analysis tools like Bandit and Pylint. We identified multiple potential vulnerabilities within Calibre's plugin design and implementation. By assigning severity scores and documenting our results, we provide an evidence-based security assessment to improve Calibre's overall security. Our findings showcase the need for constant vigilance and review for stricter code hygiene, robust access controls, and ongoing reviews to ensure the long-term resilience of Calibre's plugin architecture.**

Keywords—Calibre, Security Threats, Open-source, Static Analysis, Threat Modeling, Security Assessment, Plugin Architecture

I. Background and Introduction

A. *Name and Version:*

Calibre e-book management, Version 8.1.1 (28 Mar 2025)

B. *Software Overview*

Calibre is a widely used open-source application to manage, convert, and organize e-book libraries. It has a large user community, cross-platform support, and a modular design. Calibre has gained popularity among e-book enthusiasts, researchers, and everyday readers for its library management and conversion tools. As open-source software with a large user base, security is critical to maintaining user trust and data integrity.

C. *Problem Motivation*

We chose Calibre for this open-source software security assessment due to its popularity and large user base. As a widely used tool to collect, manage, and organize personal libraries and research collections, it is a prime candidate for uncovering potential security vulnerabilities in its plugin architecture. Small or unseen vulnerabilities with Calibre can result in compromised user data, denial of service, or degraded system performance. Our research aims to reinforce user trust and ensure the integrity of Calibre.

D. *Assessment Objective*

In this security evaluation, we seek to identify potential vulnerabilities and generate an evidence-based report for issues within Calibre's plugin architecture. Using formal threat modeling and static analysis, our goal is to provide security recommendations to strengthen security, protect user data, and maintain the trust of Calibre's user base.

II. Software Description

A. *Software functions*

Calibre is a cross-platform digital library manager. It maintains a local SQLite catalog of e-book titles, authors, or other custom tags and fields. Calibre can integrate with several e-reader devices and has a built-in web server for browsing and downloading e-books. It can translate several file formats, such as EPUB, PDF, DOCX, HTML, and convert them into a target format while normalizing metadata and CSS. It supports text-to-speech and highlighting. It can retrieve and read newspapers or blogs on a schedule and send them to a connected device for viewing. Overall, Calibre offers several functions to allow users to manage, transform, organize, and enjoy e-books without leaving the app.

B. *Key Features and Architecture*

Calibre is primarily written in Python with C and C++ for speed and system interfaces. It has a modular design with subsystems for the library manager, the conversion system, the device interface, and the viewer. Additionally, its plugin system is simple and powerful, and can communicate through the modular subsystems. Plugins can be added to the plugin code directory or installed via preferences settings. Plugins can be used to introduce new GUI actions, to add new device drivers, to override metadata downloaders, or to implement other customizable features.

Furthermore, Calibre's command-line tool allows users to build, install, and parameterize plugins. These extension tools allow Calibre to be user-friendly and highly customizable while making the plugin layer a potential attacker vulnerability. Because of this, our security assessment will focus on Calibre's plugin architecture.

C. *Installation and Testing Setup*

The first step in our assessment's installation and testing phase is to create a controlled environment to test the plugin systems. We performed these tests on a Linux Operating System. We created a virtual machine (VM) using Oracle VirtualBox and installed Ubuntu 22.04 LTS as the guest operating system. This approach provides a consistent environment that mimics real-world deployment scenarios without impacting a user's main system. VirtualBox allows us to take system snapshots at various checkpoints or reload a new instance of the same operating system in case of system faults.

Once the Ubuntu VM was set up, we installed the latest version of Python (3.12) and pip (Python's package manager). We also installed Visual Studio Code (VS Code) as our primary Integrated Development Environment (IDE) for code development and script execution.

A Python virtual environment was initialized within our project directory to manage dependencies and avoid conflicts with system-wide packages. The environment was activated, and required packages such as Bandit, Pylint, Pandas, and Matplotlib were installed within it.

Next, we cloned the official GitHub repository for Calibre using:

(git clone https://github.com/kovidgoyal/calibre.git)

This command gets the most recent version of Calibre's repository. If reproducing and want to test on the same version we are working with, use the following before the GitHub link:

(git clone --branch v8.2.1 --depth 1 {Repo Link})

We worked with version 8.2.1, which was released on March 28, 2025, to ensure our security assessment aligns with the most recent stable release. The cloned repository was stored locally within the VM.

To locate plugin-related files, we created a Python script that recursively scanned the repository for Python (.py) files and searched for keywords such as "plugin," "load_plugin," and "plugin_download." This filtered approach allowed us to focus our security analysis on components most relevant to Calibre's plugin architecture.

III. Threat Model

In performing a security analysis of Calibre's plugin systems, we define attacker roles, their objectives, and how they can exploit vulnerabilities. We will then apply two formal threat modeling techniques to categorize and prioritize risks.

A. *Attacker Roles, Objectives, and Threat Vectors*

We categorize and define attackers into three roles. To exploit plugin systems, the attacker can exploit installation and update mechanisms, manipulate the files of a local user, or find vulnerabilities in plugin libraries. Each role of attackers is motivated by either the collection of personal data, privilege escalation, service disruption, or harm to Calibre's reliability and reputation.

1) Malicious Plugin Authors

This attacker creates and distributes malicious plugin code that contains vulnerabilities for exploitation. Their goal is to use these exploits for multiple reasons. They may escalate their privileges to gain control of the host user device, obtain unauthorized access to user information, or disrupt services for those using the plugin.

2) Local Attacker

This attack category is defined as someone accessing a local user's machine. They aim to tamper with installed plugins to introduce backdoors that can elevate their privileges. They may also alter existing plugin configurations to disrupt services.

3) Remote Attacker

A remote attacker aims to target the network capability of Calibre's plugin systems. Their goal is to spoof legitimate plugin downloads/uploads to inject malicious code. They also aim to identify vulnerable network endpoints for further malicious code execution or unauthorized data access.

B. *Threat Modeling Techniques*

We use two techniques to categorize and prioritize the risks of Calibre's plugin architecture. The first is STRIDE: "Spoofing, Tampering, Repudiation, Information disclosure, Elevation of privilege" to help reason and find threats in the system. The second is DREAD: "Damage, Reproducibility, Exploitability, Affected users, Discoverability" to rate the threats identified and assign prioritization. Below are two charts of the given models of possible scenarios we are looking for during our static analysis

1) Stride Scenarios

Spoofing	Fake plugins	Attacker could upload a plugin that pretends to be from a trusted source
Tampering	Malicious code in plugin that modifies internal Calibre files	Attacker writes plugin code that could affect library behavior
Repudiation	Plugins that don't work	Plugins that fail silently may cause frustration
Information disclosure	Weak or insecure parsing or encryption	Sensitive user information or library content could be leaked
Denial of service	Plugins that cloud your system with information when opened	Possibly crash calibre and prevent user from opening large libraries
Elevation of privilege	Malicous code that gives shell access	Attacker could use OS-level commands to gain higher acces to a users system.

2) Dread Assessment

Threat Scenario	D	R	E	A	D	Score (out of 25)
Malicious plugin runs arbitrary code on the host machine	5	5	5	4	4	23
Plugin downloads or installs untrusted code	4	5	4	4	4	21
Plugins access or leak private user data	5	4	3	5	3	20
Plugin vulnerabilities lead to crashes or Denial of Service	3	4	3	3	4	17
Poor error handling	2	5	2	3	3	15

IV. Methodology

Our methodology is designed to be easily reproducible and to provide an evidence-based security assessment of Calibre's plugin architecture. Using a combination of formal threat modeling and static code analysis, we provide a high-coverage inspection to identify possible vulnerabilities that can be easily reproduced.

A. *Step-by-Step Process Recap*

1) Scope:
Our security assessment uses keywords to focus on Calibre's GitHub repository plugin architecture.

2) Version:
Version 8.2.1, released March 28, 2025, is the latest release at the time of this report. We decided on this version because it accurately reflects what an average user would utilize.

3) Establish Threat Models:
We applied STRIDE and DREAD to approximate and score possible vulnerabilities.

4) Source Code and Plugin-Related Files
To ensure repeatable results, we cloned Calibre's GitHub repository and created a Python script to scan files recursively for analysis.

5) Static Analysis Execution
After identifying plugin-related files, Bandit and Pylint were used to identify and evaluate code quality, potential defects, possible security vulnerabilities, and novel issues.

B. *Environment setup*

All static analyses were performed inside an isolated Ubuntu 22.04 LTS virtual machine hosted on Oracle VirtualBox.

C. *Static analysis*

We have chosen Bandit and Pylint as our static analysis tools because they offer a wide coverage of security vulnerabilities in Python code bases. Bandit is a powerful static type checker that verifies that variables and functions within a program are used correctly. It can assist programmers in finding inconsistencies that might result in runtime errors or security flaws. We do this to ensure an attacker cannot improperly handle supplied data in an attempt to modify the program or inject malicious code. We also use Pylint, a linter, to analyze code for standard coding practices, code complexity, and insecure processes. The combination of type safety checks from Bandit and code quality checks from Pylint provides a comprehensive security analysis of the most downloaded plugins for potential security gaps.

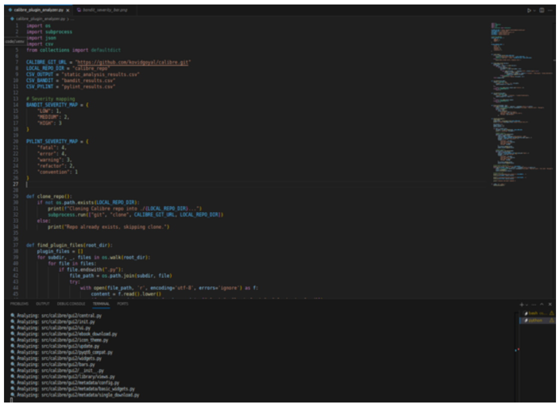
1) Bandit:
Bandit can classify vulnerabilities as low, medium, or high, and we score those classifications as 1, 2, and 3, respectively.

2) Pylint:
Using pylint, we can differentiate code errors into four sections: "convention," "refactor," "warning," and errors / fatal errors. Each specification scored 1, 2, 3, and 4, respectively.

V. Results and Findings

We created Python scripts to scrub Calibre's GitHub repository using keywords to find plugin-related files. The keywords used were "Plugin", "load_plugin", and "plugin_download." With our main analysis script, a total of 319 plugin-related files were found.

Initial scan:



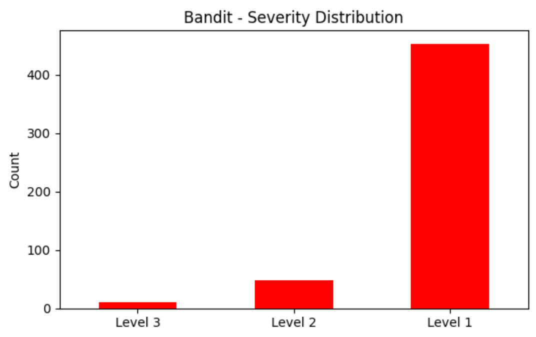
```
import os
import subprocess
import json
import csv
from collections import defaultdict

CALIBRE_GIT_URL = "https://github.com/kovidgoyal/calibre.git"
LOCAL_REPO_DIR = "calibre_repo"
CSV_OUTPUT = "static_analysis_results.csv"
CSV_BANDIT = "bandit_results.csv"
CSV_PYLINT = "pylint_results.csv"

# Severity mapping
BANDIT_SEVERITY_MAP = {
    "LOW": 1,
    "MEDIUM": 2,
    "HIGH": 3
}

PYLINT_SEVERITY_MAP = {
    "fatal": 4,
    "error": 4,
    "warning": 3,
    "refactor": 2,
    "convention": 1
}


def clone_repo():
    if not os.path.exists(LOCAL_REPO_DIR):
        print(f"Cloning Calibre repo into ./{LOCAL_REPO_DIR}...")
        subprocess.run(["git", "clone", CALIBRE_GIT_URL, LOCAL_REPO_DIR])
    else:
        print("Repo already exists, skipping clone.")


def find_plugin_files(root_dir):
    plugin_files = []
    for subdir, _, files in os.walk(root_dir):
        for file in files:
            if file.endswith(".py"):
                file_path = os.path.join(subdir, file)
                try:
                    with open(file_path, 'r', encoding='utf-8', errors='ignore') as f:
                        content = f.read().lower()
```

```
🔍 Analyzing: src/calibre/gui2/central.py
🔍 Analyzing: src/calibre/gui2/init.py
🔍 Analyzing: src/calibre/gui2/ui.py
🔍 Analyzing: src/calibre/gui2/ebook_download.py
🔍 Analyzing: src/calibre/gui2/icon_theme.py
🔍 Analyzing: src/calibre/gui2/update.py
🔍 Analyzing: src/calibre/gui2/pyqt6_compat.py
🔍 Analyzing: src/calibre/gui2/widgets.py
🔍 Analyzing: src/calibre/gui2/bars.py
🔍 Analyzing: src/calibre/gui2/__init__.py
🔍 Analyzing: src/calibre/gui2/library/views.py
🔍 Analyzing: src/calibre/gui2/metadata/config.py
🔍 Analyzing: src/calibre/gui2/metadata/basic_widgets.py
🔍 Analyzing: src/calibre/gui2/metadata/single_download.py
```

Once these files were located, each file was run through a Bandit and Pylint scan. The scan results were stored in separate CSV files, one for Pylint findings and the other for Bandit. After our initial scan, the Bandit scan yielded 462 possible vulnerabilities, and Pylint returned with 22,876 results.

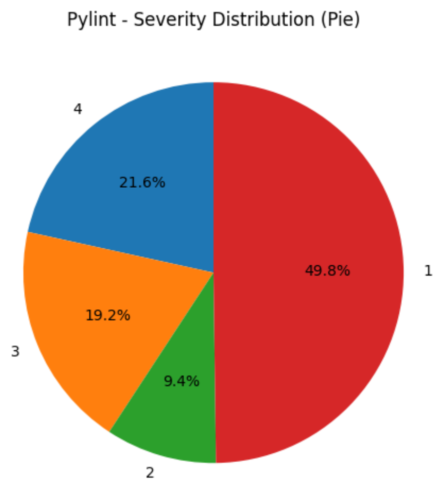
A. *Initial Analysis*

1) Bandit:
The bar chart below shows the distribution of each severity level found with Bandit.



2) Pylint:

We can see the results of the pylint scan on the plugin-related files in the pie chart below.



B. *Initial Analysis Impression:*

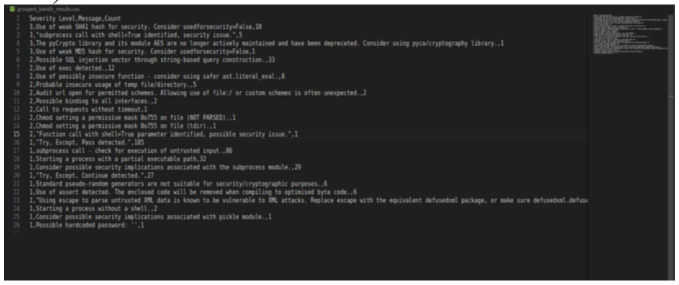
We observed a few key patterns after conducting our static code scans with Bandit and Pylint. Most findings were categorized as low severity, level 1, highlighting common issues such as poor coding practices, insecure configurations, and other minor deviations. While most of these may not pose immediate threats, their prevalence may suggest needed maintenance. We detected several medium to high severity vulnerabilities, for which we have documentation of exact errors and files, and will present them in our results. Our next plan of action was to consolidate the findings so we could better analyze what vulnerabilities and errors were being caught most frequently in these plugin-related files.

C. *Consolidated Analysis*

We noticed that in both scans there were a lot of the same errors and messages being given, and we wanted to generalize those results so we could easily classify what sections or commonly used practices within the plugin files were more critical to the programs security.

To consolidate the Bandit results, we formatted a new Python script that read in each line of the Bandit results spreadsheet and extracted the warning message. A count was kept for each message that appeared, so we could see which were more prominent in our scan. After consolidating our Bandit results, we had a spreadsheet of 25 unique vulnerabilities with their frequency and severity levels recorded.

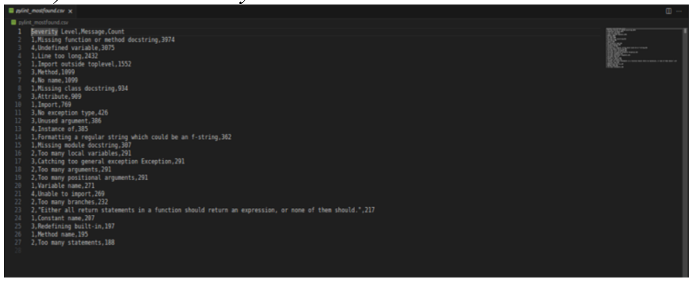
1) Consolidated Bandit Results:



```
Severity Level,Message,Count
3,Use of weak SHA1 hash for security. Consider usedforsecurity=False,10
3,"subprocess call with shell=True identified, security issue.",5
3,The pyCrypto library and its module AES are no longer actively maintained and have been deprecated. Consider using pyca/cryptography library.,1
3,Use of weak MD5 hash for security. Consider usedforsecurity=False,1
2,Possible SQL injection vector through string-based query construction.,33
2,Use of exec detected.,12
2,Use of possibly insecure function - consider using safer ast.literal_eval.,8
2,Probable insecure usage of temp file/directory.,5
2,Audit url open for permitted schemes. Allowing use of file:/ or custom schemes is often unexpected.,2
2,Possible binding to all interfaces.,2
2,Call to requests without timeout,1
2,Chmod setting a permissive mask 0o755 on file (NOT PARSED).,1
2,Chmod setting a permissive mask 0o755 on file (tdir).,1
2,"Function call with shell=True parameter identified, possible security issue.",1
1,"Try, Except, Pass detected.",185
1,subprocess call - check for execution of untrusted input.,86
1,Starting a process with a partial executable path,32
1,Consider possible security implications associated with the subprocess module.,29
1,"Try, Except, Continue detected.",27
1,Standard pseudo-random generators are not suitable for security/cryptographic purposes.,6
1,Use of assert detected. The enclosed code will be removed when compiling to optimised byte code.,6
1,"Using escape to parse untrusted XML data is known to be vulnerable to XML attacks. Replace escape with the equivalent defusedxml package, or make sure defusedxml.defuse
1,Starting a process without a shell.,2
1,Consider possible security implications associated with pickle module.,1
1,Possible hardcoded password: '',1
```

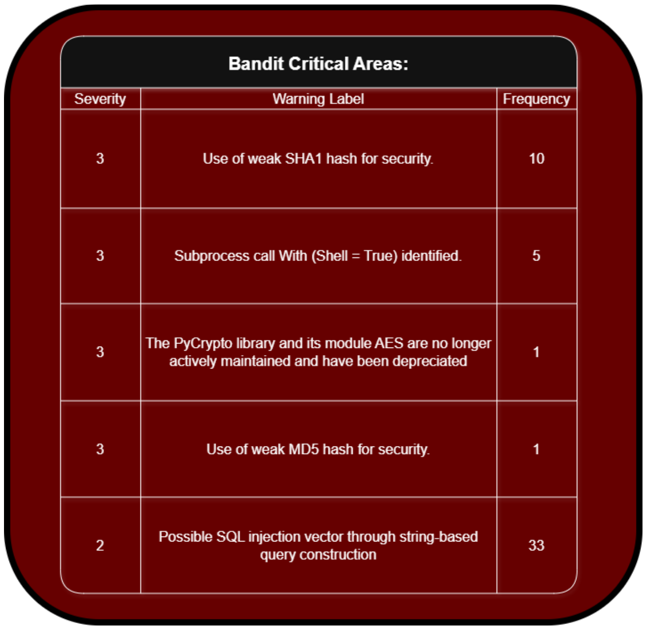
We conducted a similar process to consolidate the Pylint results. We tried consolidating the results with the same parameters as the bandit scans. Still, even the warnings under the same category had too many unique file names to consolidate properly by the full error messages. To combat this, we separated the messages from the rest of the error messages. With this method, we were still left with 203 errors and warnings from Pylint. We wanted to see the most concerning areas in these plugin files, so we consolidated that list to match the Bandit scans by placing the top 25 most frequently discovered syntactical/semantical errors Pylint found.

2) Consolidated Pylint Results:



```
Severity Level,Message,Count
1,Missing function or method docstring,3974
4,Undefined variable,3075
1,Line too long,2432
1,Import outside toplevel,1552
3,Method,1099
4,No name,1099
1,Missing class docstring,934
3,Attribute,909
1,Import,769
3,No exception type,426
3,Unused argument,386
4,Instance of,385
1,Formatting a regular string which could be an f-string,362
1,Missing module docstring,307
2,Too many local variables,291
3,Catching too general exception Exception,291
2,Too many arguments,291
2,Too many positional arguments,291
1,Variable name,271
4,Unable to import,269
2,Too many branches,232
2,"Either all return statements in a function should return an expression, or none of them should.",217
1,Constant name,207
3,Redefining built-in,197
1,Method name,195
2,Too many statements,188
```

D. *Vulnerabilities or Misconfigurations:*



Bandit Critical Areas:

Severity	Warning Label	Frequency
3	Use of weak SHA1 hash for security.	10
3	Subprocess call With (Shell = True) identified.	5
3	The PyCrypto library and its module AES are no longer actively maintained and have been depreciated	1
3	Use of weak MD5 hash for security.	1
2	Possible SQL injection vector through string-based query construction	33

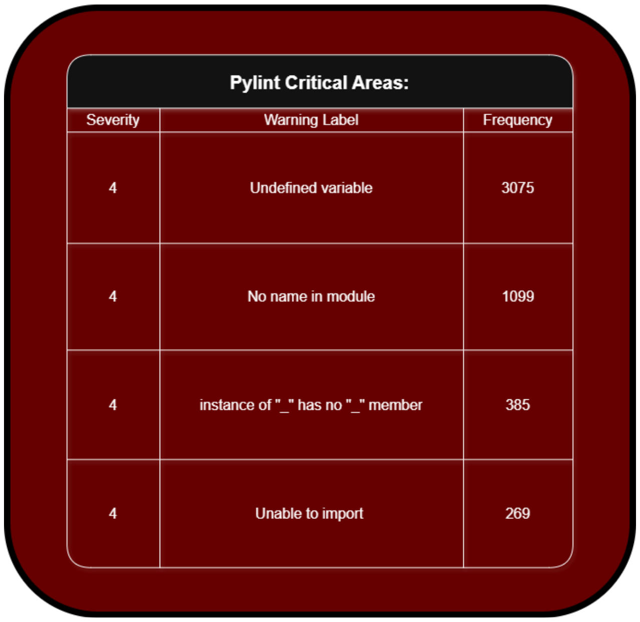
1) Vulnerabilities:

a) The warnings for weak SHA1 and MD5 are similar and used for hashing private information. Bandit flags these methods because they are considered cryptographically broken and are vulnerable to collision attacks, meaning two pieces of data can produce the same hash value. If Calibre relies on these methods for password hashing, file integrity, or verification, attackers could exploit this to steal data or bypass security mechanisms.

b) Using Shell = True in Python's subprocess module can lead to shell injection vulnerabilities if they are related to any user input fields and could be used for unwanted code execution. This can be critical in an application like Calibre because of the user-driven automation of plugin systems.

c) Using the PyCrypto library introduces inherited risk, such as exposure to unpatched vulnerabilities and cryptographic errors. This poses a large threat because this is a third-party library that Calibre cannot manage itself.

d) Using raw string concatenation or formatting to construct SQL queries creates an opening for SQL injection sites. In the case of Calibre, any component, such as plugin modules, metadata handling tools, or user-generated query interfaces that construct SQL queries via string interpolation, is vulnerable to exploitation. Attackers could manipulate the structure of a query to access or modify unauthorized data, bypass authentication, or execute destructive commands within the database context.



Pylint Critical Areas:

Severity	Warning Label	Frequency
4	Undefined variable	3075
4	No name in module	1099
4	instance of "_" has no "_" member	385
4	Unable to import	269

2) Misconfigurations:

a) Having undefined variables is a critical defect because it can result in runtime errors. These could lead to crashes during execution and cause data loss during plugin execution, file conversion, or synchronization.

b) Attempting to import nonexistent module members can lead to import or attribute errors at runtime. These errors may stem from outdated references or API depreciation. This could lead to the failure of plugin initialization and cause reduced functionality.

c) Pylint also picked up on errors when the program tries accessing nonexistent attributes on object types. In a dynamically loaded and plugin-driven architecture like Calibre's, these errors could go undetected until runtime, leading to Attribute Error exceptions.

d) Unresolved module imports indicate failed import attempts because of incorrect paths, missing packages, or broken dependencies. These errors could cause various problems, such as loading UI elements or auxiliary libraries.

3) Result Impressions:

The results of our static analysis gave us valuable insights into Calibre's security mechanisms as they pertain to its plugin system. Critical issues such as undefined variables, improper module imports, and usage of insecure cryptographic functions indicate vulnerabilities that could propagate through user-contributed plugins and custom extensions. By correlating specific high-severity Bandit and Pylint warnings with their file paths, we can isolate patterns of problematic modules and potentially vulnerable plugins. These findings highlight areas needing patching and can be used to create a prioritization model for code review. For instance, plugins that use shell execution or construct SQL queries dynamically represent high-value targets for manual inspection and extensive testing. Moreover, the clustering of issues like failed imports and attribute access errors within certain directories suggests maintainability problems. These insights can guide future efforts to harden Calibre's plugin infrastructure and develop automated safeguards against insecure third-party contributions.

VI. Discussion and Limitations

Our security assessment can only provide a nuanced picture of Calibre's overall security. While nuanced, it provides a solid foundation of the plugin's architecture strengths and weaknesses. A few issues we encountered include not performing dynamic analysis, verifying test results, and testing on only one operating system.

Time constraints remained one of our most significant limitations for this security assessment. Because of this, the dynamic analysis remained outside of our scope. A weakness of only performing static analysis is the possibility of producing false positives and negatives. With further testing, dynamic analysis, and a comprehensive analysis of the results, we could provide a truly comprehensive security assessment of Calibre.

VII. Proposed Mitigations

Following our security assessment, we propose recommendations in three categories to ensure a strong security posture.

A. Secure configuration tips

Because Calibre is an open-source software, users should adhere to the principle of least privilege. They should ensure that only administrators can use or access the app's administrative tools or functions. Additionally, to prevent unverified third-party plugins from being installed by default, all users should disable “Allow installing from untrusted sources.”

Preferences -> Plugins -> Allow installing from untrusted sources

To further ensure protection against malicious plugins or insecure code, all users should routinely verify they are current on software and security updates.

B. *Recommended code/design changes*

To protect against insufficient plugins, we recommend a few design changes to the plugin architecture. Introducing and using a permission-based system with strong authentication to monitor and restrict plugin actions, adding an official plugin repository so plugins can be properly vetted and monitored, and mandatory plugin signatures and verification to ensure only trusted plugins are loaded.

C. *Plugin/module securing suggestion*

We propose adding a “required reading” for all plugin authors. This can include guidelines of secure coding practices, a new plugin guide or rulebook, and information about unauthorized plugin functions. By implementing these suggestions, we aim to enhance overall security without compromising the customizability that contributed to Calibre’s success.

VIII. Conclusion

In this assessment, we evaluated the security of Calibre’s plugin architecture using static analysis tools and formal threat modeling techniques. Our goal was to uncover critical areas in Calibre's plugin system by finding vulnerabilities, misconfigurations, and insecure coding practices that could compromise users' privacy, system performance, or the overall reliability of the Application.

We set up our tests in a controlled Ubuntu-based environment and located and scanned 319 plugin-related files using Bandit and Pylint. Our analysis used an ideology that stemmed from our previously established STRIDE and DREAD threat models. We can use our findings to locate areas in Calibre’s plugin architecture that may threaten the integrity of the entire application.

A. *Summary of assessment results*

Our static analysis revealed over 22,000 pilot messages and 462 Bandit security warnings, highlighting both minor issues and critical security flaws. With Bandit, we exposed vulnerabilities like weak cryptographic algorithms, areas where unwanted shell command execution may be present, errors in using third-party dependencies, Frequent undefined variables and import errors that indicate weak runtime reliability, and other areas that may be Open to SQL injection.

B. *Final takeaway message*

The flexibility of Calibre through its plugin system is core to its appeal, but also represents one of its most significant security challenges. Our analysis shows that the plugin system can become a large surface area for attacks and system instability without careful management. To protect users and system information, we recommend employing a signed plugin requirement and maintaining an official plugin repository to implement a permission system to limit what plugins can access or perform. It would also be beneficial to provide developer guidelines and secure coding checklists for all plugin contributors while regularly conducting static and dynamic analysis to stay ahead of evolving threats. By adopting these recommendations, Calibre could reinforce its reputation as a trusted, customizable, and secure platform for digital library management.

IX. References

A. *Tools*

[1] Calibre e-book management 8.1.1 [28 Mar, 2025]

[2] VirtualBox 7.1.8 [15 Apr, 2025]

[3] Ubuntu LTS 22.04 [21 Apr, 2022]

[4] Python 3.12 [2 Oct, 2023]

[5] Pip 24.0 [2 Mar, 2024]

[6] Visual Studio Code 1.99 [Mar, 2025

[7] Bandit 1.8.3 [16 Feb, 2025]

[8] Pylint 3.3.6 [20 Mar, 2025]

[9] Pandas 2.2.3 [20 Sep, 2024]

[10] Matplotlib 3.10.1 [27 Feb, 2025]

[11] Numpy 2.2.4 [16 Mar, 2025]

[12] Pyparsing 3.2.3 [25 Mar, 2025]

B. *Documentation*

[13] Kovid Goyal, “calibre User Manual,” April 18, 2025. Available: https://manual.calibre-ebook.com/calibre.pdf

[14] Kovid Goyal, “Setting up a calibre development environment,” April 18, 2025. Available: https://manual.calibre-ebook.com/develop.html

[15] Kovid Goyal, “Customizing calibre,” April 18, 2025. Available: https://manual.calibre-ebook.com/customize.html

[16] Kovid Goyal, “Writing your own plugins to extend calibre’s functionality,” April 18, 2025. Available: https://manual.calibre-ebook.com/creating_plugins.html

[17] Kovid Goyal, “ebook-convert,” April 18, 2025. Available: https://manual.calibre-ebook.com/generated/en/ebook-convert.html