PERFORMANCE ENHANCMENT TOOL TO OPTIMIZE SYSTEM RESOURCES IN WINDOWS OS

SIVA JEGADEESH C B
2021503559
Department of Computer Technology
MIT, Anna University
Chennai, India

BABITH SARISH S
2021503009
Department of Computer Technology
MIT, Anna University
Chennai, India

RAJKUMAR M
2021503039
Department of Computer Technology
MIT, Anna University
Chennai, India

SHARVESH S P
2021503717
Department of Computer Technology
MIT, Anna University
Chennai, India

***Abstract*—Debloating software plays a crucial role in optimizing computer systems by removing unnecessary or unwanted software, services, and features. However, existing performance enhancement techniques often focus solely on debloating, neglecting other potential optimizations. To address this, a user interface debloating software was developed for Windows Operating System. This software incorporates functionalities such as auto start management, windows spying prevention, GPU flickering fix, clean task execution, deactivation of unnecessary Windows components, and more. By automatically discarding unnecessary files, the software improves system efficiency, mitigating resource consumption and reducing lag. The software's benefits encompass enhanced system performance, strengthened privacy and security, increased storage space, improved user experience, customization options, stability and compatibility enhancements, and energy efficiency. While outcomes may vary depending on the specific software and system configuration, it is crucial to exercise caution and use reputable debloating software.**

***Index Terms*—Powershell, Shell scripting, Disk Cleanup, Uninstalling bloatware, Disable Visual Effects, bloatware**

I. INTRODUCTION

Efficient system management is essential for optimal performance, and one crucial aspect is the proper utilization of system resources, memory, and storage. However, the updating of system packages often brings along unnecessary files that consume system resources and hinder performance. These "junk files" not only occupy valuable memory but also delay the execution of necessary files, resulting in lagging system performance. While existing performance enhancement techniques primarily focus on debloating, a more comprehensive approach that combines debloating with additional optimizations can yield even better results.

Debloating refers to the process of removing unnecessary or unwanted software, services, and features from a computer system. By eliminating bloatware, redundant processes, and unneeded components, system performance and efficiency can be significantly improved.

System efficiency encompasses the optimal utilization of system resources, memory, and storage to achieve enhanced performance. By minimizing resource consumption and eliminating unnecessary files, system efficiency is maximized, resulting in faster execution times and improved responsiveness.Existing performance enhancement techniques typically focus on debloating as a means to improve system performance. While effective, there is room for additional optimizations to further enhance system efficiency and responsiveness.

By combining debloating with additional optimizations, our debloating software aims to provide users with a holistic solution for system performance enhancement. The effectiveness and impact of this software will be evaluated through rigorous testing and comparison with existing approaches. The ultimate goal is to empower users with a reliable tool that enhances their computing experience by optimizing system efficiency and responsiveness.

II. EXISTING WORK

Existing work on debloating has focused on removing unnecessary software, services, and features to improve system performance and efficiency. Techniques have been developed for identifying bloatware, automating the debloating process, enhancing privacy and security, optimizing system resources, providing user customization, considering compatibility, and evaluating the impact of debloating. These efforts aim to streamline the removal of unwanted software, reduce resource consumption, and enhance user experience. These techniques collectively contribute to the debloating process, improving system efficiency, optimizing resource utilization, and enhancing overall performance.

III. DRAWBACKS

Firstly, software compatibility can be a challenge. Debloating processes may inadvertently remove essential components or services, resulting in compatibility issues and system instability. Ensuring the debloating process does not disrupt critical software is crucial. Secondly, unintended consequences can occur. Removing certain software components or features may lead to unexpected behavior, software errors, or loss of functionality that users rely on. Thorough testing and consideration are necessary to mitigate such risks. Additionally, there are potential risks associated with system modification. Mishandling during the debloating process can cause system errors or crashes. Limited vendor support is another drawback, as debloating software may not receive regular updates or support. This can lead to compatibility issues or security vulnerabilities.

IV. LITERATURE SURVEY

From [1]

This work proposes a novel feature-oriented debloating approach and builds a prototype, named XDebloat , to automate this process in a flexible manner. First, it proposes three feature location approaches to mine features in an app. XDebloat supports feature location approaches at a fine granularity. The results show that XDebloat can successfully remove components from apps or transform apps into on-demand modules within 10 minutes. For the pruning-based debloating strategy, on average, XDebloat can remove 32.1 percent code from an app. For the module-based debloating strategy, XDebloat can help developers build instant apps or app bundles automatically.While the proposed feature-oriented debloating approach and the XDebloat prototype demonstrate effectiveness in removing components and reducing code size in Android apps, the narrow focus on Android-specific issues may not generalize well to other environments or platforms.

From [2],

This work introduces sparse constant propagation that performs constant propagation only for configuration-hosting variables and show that it performs better (higher code size reductions) compared to constant propagation for all program variables. Overall, the results show that Trimmer reduces binary sizes for real-world programs with reasonable analysis times. Across 20 evaluated programs, we observe a mean binary size reduction of 22.7 percent and a maximum reduction of 62.7 percent. For 5 programs, It observes performance speedups ranging from 5 to 53 percent. Moreover, It shows that winnowing software applications can reduce the program attack surface by removing code that contains exploitable vulnerabilities. It finds that debloating using Trimmer removes CVEs in 4 applications.One drawback of the described system is that it focuses primarily on binary size reduction and performance optimizations without explicitly addressing potential trade-offs in terms of program functionality or maintainability.

From [3],

It analyzes the system to identify components that are not directly related to real-time functionality. These components may include unnecessary libraries, drivers, or background processes that can be safely removed.This can be achieved through static analysis or runtime profiling to determine which features are not actively utilized.It applies techniques like dead code elimination, function inlining, or code compression to reduce the overall code size, which can lead to faster execution and improved real-time performance.While the goal is to remove unnecessary components, libraries, or processes, it is challenging to accurately determine the full extent of their influence on the system's behavior.

From [4],

Device drivers are software components that allow the operating system to communicate with hardware devices. It introduces a technique called IRQDebloat, which aims to minimize the code size and complexity of device drivers to mitigate potential security vulnerabilities.IRQDebloat works by analyzing device drivers and identifying unnecessary or redundant code that can be safely removed. By reducing the size of the drivers, the attack surface, or the potential entry points for attackers, is diminished. This technique helps to mitigate the risk of exploitation through driver-related vulnerabilities.While the goal is to minimize code size and complexity, the process may inadvertently remove essential functionalities or dependencies that are required for proper device operation.

From [5],

Software aging refers to the degradation and deterioration of software over time, which can lead to performance issues, crashes, and other problems.The paper discusses different types of tools and methodologies that can be employed for software aging analysis. It covers both static analysis techniques, which examine the software's code without executing it, and dynamic analysis techniques, which involve running the software and monitoring its behavior.In the case of static analysis, examining the software's code without executing it may not capture certain runtime behaviors or dependencies that can contribute to software aging.

V. PROPOSED WORK

A. ¸Analysis

The original source code undergoes analysis to identify bloat, vulnerabilities, and detect unnecessary or redundant code. This analysis helps in understanding the codebase and identifying areas for improvement.

B. CVE Mapped Source Code

Once the analysis is complete, the source code is mapped with Common Vulnerabilities and Exposures (CVE) entries. This mapping allows for identifying potential security vulnerabilities present in the codebase.

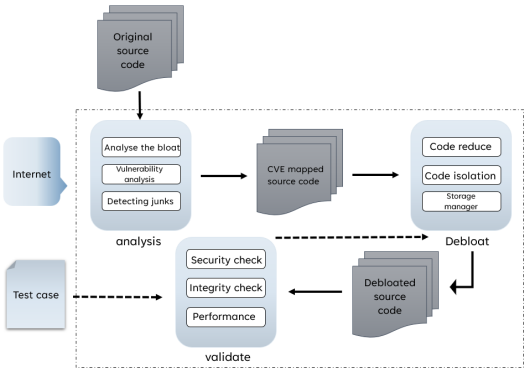

Fig. 1. DEBLOATING ARCHITECTURE

C. Debloat

In this step, the identified bloat and unnecessary code are removed or reduced. Code reduction techniques are applied to streamline the codebase and eliminate redundancies. Code isolation may be employed to separate critical functionalities from non-essential components. Additionally, a storage manager can be utilized to optimize resource usage and efficiently manage memory or storage.

D. Debloated Source Code

After the debloating process, the source code is transformed into a debloated version. This version contains the necessary code with reduced bloat, eliminated vulnerabilities, and optimized resource usage.

E. Validation

The debloated source code undergoes validation to ensure its security, integrity, and performance. Security checks are performed to identify any remaining vulnerabilities or potential risks. Integrity checks verify that the codebase is functioning as intended without any unintended modifications. Performance evaluations assess the optimized code's efficiency and effectiveness.

F. Test Case Validation

Finally, test cases are executed to validate the debloated source code. These test cases verify the correctness, functionality, and performance of the codebase, ensuring that it meets the desired requirements and objectives. %enddocument Overall, this workflow aims to enhance the security, performance, and efficiency of the codebase by removing unnecessary code, mitigating vulnerabilities, and validating the debloated version through rigorous testing.

VI. IMPLEMENTATION

A. Setup

The debloating software environment was successfully set up for the project, utilizing Windows PowerShell and shell scripting. The project was developed on a Windows operating system, leveraging the power of Windows PowerShell as a scripting language and command-line shell. Visual Studio Code served as the chosen text editor for script development, offering a versatile and user-friendly interface. Key components of the setup included the creation of a hashtable for efficient data storage, establishment of a dedicated project folder for organization, and compatibility checks for the Windows operating system version. Additionally, the list of installed software was obtained to identify bloatware, and the project path was verified to ensure smooth resource access. These setup steps collectively created a robust environment for the debloating software project, providing a solid foundation for further development and optimization.

B. Windows Services

By storing the Windows services in the "service" variable, it becomes possible to perform various operations on them programmatically. This technique will allow to set specific properties or configurations of a service, such as its startup type or recovery options. This allows for fine-tuning the behavior and settings of individual services based on specific requirements. The "service" variable enables starting or stopping services as needed i.e. the script or program can programmatically control the state of the services, either by starting them to make them active or by stopping them to disable their functionality temporarily.

C. Windows Feature

The "feature" variable holds information about all the Windows features available on the system. This includes various optional components, tools, and functionalities that can be enabled or disabled. By accessing and manipulating this variable, it becomes possible to programmatically configure the Windows features without the need to restart the system. This allows for dynamic adjustments and customizations of the Windows environment based on specific requirements.

D. Windows Capability

The "capability" variable holds information about the Windows capabilities. These are specific sets of functionalities or features that can be enabled or disabled individually. By utilizing this variable, it becomes possible to retrieve a list of available capabilities and selectively disable or enable them as needed. This offers flexibility in managing the capabilities of the Windows system and tailoring it to specific use cases.

E. Windows Tasks

The "Windows Tasks" feature facilitates the scheduling and management of tasks in the Windows operating system. It utilizes the "task" variable to store information about Windows tasks created using the Task Scheduler. This variable enables programmatic interaction with tasks, including enabling or disabling them. Disabling a task suspends its execution temporarily, while enabling it allows it to run based on its configured schedule. This feature offers automation capabilities for executing actions or scripts at specific times or conditions, providing flexibility and control over task execution within the Windows environment.

F. Windows Index

The ”WindowsIndex” feature focuses on managing environment variables and controlling the indexing functionality in Windows. A variable called ”drive” is used to track indexes and interact with Windows Management Instrumentation (WMI) objects. By utilizing the ”drive” variable, developers can access WMI objects to retrieve information and modify indexing settings. One specific functionality mentioned is setting the ”indexEnable” property to false, which disables indexing for specific drives or directories. This feature provides flexibility in managing indexing and optimizing system performance based on specific requirements.

G. Graphical User Interface

The mentioned features are part of the debloating software and are intended to be implemented in a GUI (Graphical User Interface) environment. Each feature serves a specific purpose in optimizing and customizing the Windows system. The features include setting up OO ShutUp to disable Windows features, configuring System Restore Points for system recovery, enabling the Windows Cleanup function to remove unnecessary files, implementing a Driver Cleanup function to manage outdated drivers, integrating a Fan Control function for temperature regulation, and incorporating an Auto Run function to manage startup programs. These features aim to enhance system performance, privacy, and customization options within a user-friendly GUI interface.

VII. RESULTS AND DISCUSSION

This involves evaluating various metrics to assess their effectiveness in optimizing system performance. These metrics include system startup time, execution time for common tasks, system responsiveness, resource consumption (such as CPU, memory, and disk usage), user experience, stability, and security. By measuring startup time, it is possible to determine if the debloating technique has improved the system's boot-up speed. Execution time analysis helps evaluate whether the technique has reduced the time required to perform common tasks or run applications. Assessing system responsiveness provides insights into how quickly the system responds to user interactions. Monitoring resource consumption allows for understanding the impact of debloating on resource efficiency. User experience analysis involves gathering feedback on perceived system speed, ease of use, and overall satisfaction. Stability assessment ensures that the debloating technique does not introduce system crashes or software conflicts. Lastly, security analysis examines the impact of debloating on system vulnerabilities and privacy risks.

VIII. CONCLUSION

The debloating technique offers an effective solution for optimizing system performance, enhancing privacy and security, increasing storage space, streamlining the user experience, improving stability and compatibility, and promoting energy efficiency. By removing unnecessary software, services, and features, the technique declutters the system and improves boot times, responsiveness, and multitasking. It eliminates bloatware and unwanted files, freeing up storage space and bolstering privacy and security. The technique delivers a cleaner and more organized user experience while reducing system instability and compatibility issues. Additionally, by disabling unnecessary background processes and services, it enhances energy efficiency and potentially prolongs battery life. However, careful consideration of system configurations and user requirements is necessary when applying the debloating technique. Overall, the debloating technique provides a comprehensive solution to optimize system efficiency and elevate the overall computing experience.

REFERENCES

[1] Y. Tang et al., ”XDebloat: Towards Automated Feature-Oriented App Debloating,” in IEEE Transactions on Software Engineering, vol. 48, no. 11, pp. 4501-4520, 1 Nov. 2022, doi: 10.1109/TSE.2021.3120213.

[2] Ahmad et al., ”Trimmer: An Automated System for Configuration-Based Software Debloating,” in IEEE Transactions on Software Engineering, vol. 48, no. 9, pp. 3485-3505, 1 Sept. 2022, doi: 10.1109/TSE.2021.3095716.

[3] J. Wang, A. Li, H. Li, C. Lu and N. Zhang, ”RT-TEE: Real-time System Availability for Cyber-physical Systems using ARM TrustZone,” 2022 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2022, pp. 352-369, doi: 10.1109/SP46214.2022.9833604.

[4] Z. Hu and B. Dolan- Gavitt, ”IRQDebloat: Reducing Driver Attack Surface in Embedded Devices,” 2022 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2022, pp. 1608-1622, doi: 10.1109/SP46214.2022.9833695.

[5] R. Natella and A. Andrzejak, ”SAR Handbook Chapter: Experimental Tools for Software Aging Analysis,” 2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Coimbra, Portugal, 2020, pp. 1-1, doi: 10.1109/ISSREW51248.2020.00096

[6] Q. Xie, Y. Wang and Z. Qin, ”Malware Family Classification using LSTM with Attention,” 2020 13th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI), Chengdu, China, 2020, pp. 966-970, doi: 10.1109/CISP-BMEI51763.2020.9263499.

[7] Q. Xin, M. Kim, Q. Zhang and A. Orso, ”Program Debloating via Stochastic Optimization,” 2020 IEEE/ACM 42nd International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER), Seoul, Korea (South), 2020, pp. 65-68.

[8] D. He, J. Lu and J. Xue, ”Context Debloating for Object-Sensitive Pointer Analysis,” 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), Melbourne, Australia, 2021, pp. 79-91, doi: 10.1109/ASE51524.2021.9678880.

[9] P. McDaniel, ”Bloatware Comes to the Smartphone,” in IEEE Security Privacy, vol. 10, no. 4, pp. 85-87, July-Aug. 2012, doi: 10.1109/MSP.2012.92.

[10] A. R. C, A. J and K. G. R, ”A Robust Tool To Debloat And Optimize Android Devices,” 2021 Third International Conference on Inventive Research in Computing Applications (ICIRCA), Coimbatore, India, 2021, pp. 1113-1118, doi: 10.1109/ICIRCA51532.2021.9545044.