National College of Ireland

EDR Silencers: Prevent, Detect and Remediate

MSc Research Project
Programme Name: MSc Cyber Security

Arghadeep Lahiri
Student ID: 23403888

School of Computing
National College of Ireland

Supervisor: Kamil Mahajan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Arghadeep Lahiri
Student ID:	23403888
Programme:	MSc Cyber Security **Year:** 2025
Module:	Practicum
Supervisor:	Kamil Mahajan
Submission Due Date:	11th December 2025
Project Title:	EDR Silencers: Prevent, Detect and Remediate
Word Count: 5925	**Page Count:** 17

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Arghadeep Lahiri

Date: 11th December 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

EDR Silencer: “Prevent, Detect, Remediate”

Arghadeep Lahiri
Student ID: 23403888

Abstract:

With the abundant increase in attacks to silence EDR and security software’s using Windows firewall misconfiguration and policy misconfiguration is often exploited by malicious actors to maintains stealth nature and disable endpoint protection without triggering any detection from intrusion detection system. While most security products focus on network packets and endpoint threats, underlying firewall misconfigurations are never really validated. This project proposes the design and implementation of host based monitoring system that specifically monitors the Microsoft’s WFP to identify any misconfigured or malicious firewall block rules associated with processes.
The system interprets the Base Filtering Engine (BFE) while using the native BFE APIs to interact with the active firewall filters and extract application identities that links them to Block rules. A baseline concept-based model is implemented where a baseline image of existing WFP block filters are captured on a pre-configured windows machines and stores in a working directory. When a new firewall rule to block network connections of a process, the system compares with the baseline and flags the anomaly to the user. The information is logged in details inside a separate log files and Windows Event Viewer. Thus this model makes it a probable concept for the future.
Evaluating experimentally reveals that the system reliability identifies application-level suppression that included user level and policy level rule creation and distinguishes between baseline and new list. This approach focuses on host level misconfiguration rather than network packet level or kernel level inspection.
The output generated from underlying monitor software reveals vital information about the newly blocked application providing critical visibility into silent firewall tampering and uses WFP as a powerful telemetry source for monitoring misconfigured security rules in firewall.

Keywords: Windows Filtering Platform, Base Filtering Engine, Windows Firewall, Detection, Endpoint Security, Baseline detection model, Host based monitoring, Windows Event logging, System Hardening, Policy tampering, Defensive security Telemetry

1. Introduction:

Host based Firewall systems in modern endpoint security play an important role in controlling how applications can communicate over network. On Windows systems, this function is carried out by Windows Filtering Platform (WFP) which evaluates every network traffic and maintains a set of dynamic rules. While these rules are mandatory for enforcing policy, they are also prone to misconfiguration or deliberate manipulation by malicious actors that are looking to disable endpoint security software and go by their business.
Traditional monitoring and previous research talk on how EDR silencers infiltrate a system and disable security, but very less research on what are the factors that lead to the entry of such actors in under researched. An attacker can silently create a firewall rule to block network communications on security software services thus blinding the security of the host system. This project focuses on developing a lightweight monitoring system that detects any

unauthorized creation of rules and flags to the user by interrogating the WFP filters and comparing to a baseline image.
Thus, it aims to provide a practical host level application that enhances firewall monitoring without the need of packet inspection, kernel level checks or third party security frameworks

2. Literature Review: The literature review part compromises of four important sections: Motivation to choose this topic, evolution of EDR evasion and bypass technique, related work on WFP and Kernel manipulation, discussion on existing gaps and research contribution.

2.1 Motivation to choose the topic:

Endpoint Detection and Response (EDR) are essential central components of modern cybersecurity architecture which provides real time threat monitoring, alerting, incident detection and remediation automation. Despite being sophisticated, threat adversaries has developed techniques to bypass, disable and tamper EDR network services, thereby reducing immediate threat alerting mechanism. Existing research has showed how these malicious actors can exploit both user and kernel level weaknesses, manipulated WFP rules or configuration and driver level tampering to hamper network visibility among endpoint detection and central management consoles.
Current studies, talks a lot about how EDRs can be evaded, however comparative studies on detection and remediation of such silencing techniques is still under-researched.
During my professional time at Sophos, I handled more than 2000+ endpoint and malware issues cases, which motivated me to work on the next gen technology, and I am hoping to explore and nurture my knowledge towards securing endpoints in an enterprise.

2.2 Study of EDR Evasion and Bypass technique:

Recent studies have characterized EDR evasion techniques as multi layered offensive strategy targeting memory monitoring, kernel callbacks and behaviour heuristics. In a study by Hook chain: A new perspective for Bypassing EDR Solutions [1], the author exemplifies on how an attacker can disrupt the user-mode API calls and eventually evading EDR hooks using chained trampolines. Equally [3], empirical testing of endpoint security in relation to Advanced, persistent threats found that commercial EDRs tend not to respond when attackers use memory injections and inter process communication obfuscation in unison.
O'Reilly's Evading EDR [13], is the perfect book that demonstrates **hook unhooking**, **kernel- callback suppression** and **API redirection**. It shows EDR software's have blind spots that can be used for code injection, dll sideloading and process hollowing attempts. As demonstrated in *Fast and Furious: Outrunning Windows Kernel Notification Routines from User-Mode* [21], process monitoring can be evaded by exploiting kernel gaps. These comparative studies demonstrate the attacker ability vs response time in real time and how these gaps can be exploited to take down the backbone of endpoint security.

2.3 WFP and Kernel Manipulation:

Studies suggest that Kernel level and network layer silencer often leaves EDR software's useless as they no longer can communicate back to their central consoles. *NoFilter: Abusing the Windows Filtering Platform for Privilege Escalation* [16] and *Potential Evasion via Windows Filtering Platform* [15] clearly shows how attackers can craft and modify malicious WFP rules to silence network telemetry or reroute traffic. Microsoft's documentation [17][18] express details about filter altitude and sublayers, which are easily exploitable by malicious drivers.
Previous study where Sysmon.sys was repurposed to alert alerts or mis direct traffics,

illustrating BYOVD concept [14]. Kernel hook and filter manipulation can hide malicious activity without leaving system call traces. Pwn2Own studies re affirms that kernel exploitation remains a viable option to disable any security drivers [22]. So, looking at these papers and research, we can certainly say that WFP filter manipulation or silencing is still a menacing topic which is still under-researched.

2.4 Detection, threat hunting and defensive approach:
Defensive research has mostly focused on post incident behaviour, but profiling kernel level event can alert an early ransomware infection possibility, on the other hand adversarial machine learning [10][12] approaches focuses on data driven decisions. These efforts remain reactive, however often failing to detect configuration level tampering in real time.
Emerging work, like Security Analysis of Scan Obfuscation Techniques [11] and HookChain[1] suggest that correlation of user mode and kernel telemetry can be used, to improve visibility into self-healing architecture[20] and recent USENIX discussion [8]underscores the importance of automatic identifying and rectifying integrity violations.

2.5 Gaps Identified:
After doing extensive research on existing EDR defensive approach, the following gaps were found which are worth discussing. Firstly, there are limited level of detection configuration level interference and only few solutions can monitor BFE/WFP for unauthorized rule creation that can drop network traffic of processes. Secondly, systems often fail to identify accidental to malicious misconfiguration which leads to more false alerts. Lastly, there is a huge scope in autonomous integrity validation, that can monitor for driver health, processes making network connection.

2.6 Research contribution:
After a detailed study on the existing defensive mechanisms against EDR silencers, it is understood that there are several loopholes inside WFP rule and BFE misconfiguration, the can lead to attackers, disable endpoint EDR and go their way executing malicious activities. This research project will focus on proposing vender-oriented framework. By leveraging WFP APIs, ETW trace logs and driver altitude verification, the system will automatically identify and restore configurations that are affected by silencers.
Unlike prior work focusing on attack techniques, this project will primarily focus on monitoring WFP and BFE for process silencing attempts, simultaneously providing context aware remediation or alerting the administrator rectifying the system configuration gap, thus offering an experimental foundation for self-healing EDR architectures and maintaining integrity of work.
Bridging the offensive and defensive techniques, will provide a sustainable approach from reactive detection to proactive detection and strengthen resilience.

3. Methodology:
This section outlines the methods to design, implement and evaluate the results proposed in the research. Unlike traditional malware detecting approach, the project focuses on anomalies in the system configuration and provides best practice recommendations that may effectively disable endpoint communication from security monitoring. This project will combine interactive design science with controlled experimental validation.

3.1 Research technique:
For my research I am implementing the Design Science Research (DSR)[25] methodology

which aligns with cybersecurity system development that aims at solving real world problems effectively. The steps followed in this project are as follows:

a) Identify problem: Determining how attackers can use WFP/BFE misconfigurations and alter the system level security controls (EDR) from working effectively.
b) Requirement Definition: Establishing functional, performance and reliability requirements for a monitoring solution.
c) Design and Implementation of Artifact: Developing C++ based code that can integrate easily with native Windows WFP API's
d) Evaluation: This step covers the end results and effectiveness of the solution that will be built which includes detection performance, accuracy and fault tolerance
e) Reflection of the outcome: Assessing the effectiveness of the solution and the output it bears and possible extension to EDR's resilience.

3.2 System architecture overview:

Based on Windows architecture, the system overview is reviewed which first includes a Baseline image that will capture and store an initial WFP and sublayer configuration snapshot (BlockedAppsBaseline.txt l) for comparison with any changes. The next step will be to monitor the native Windows FWPPM API like FwpmEngineOpen0(), FwpmFilterEnum0() that periodically evaluates active filters and detect differences from the baseline expected values. Based on the detection module, rule-based logic to detect WFP(Block) operation, tracking processes that kills network communication and accordingly disable a service or delete the operation and alert the admin about the anomaly. Suspicious configurations will result in Warning or Critical alert in form of Windows toast and event viewer logs. Using C++ language, monitoring for WFP/BFE state every 5 minutes will be conducted and using Powershell and a log will be written to be fed in local log servers or ready them manually. More technical details will be elaborately discussed during the implementation stage.

3.3 Implementation Strategy:

The system will be implemented using C++ in Visual studio using Windows native libraries. It will operate under LocalSystem which will provide the elevated privileges required read and validate WFP configuration object. Upon the first execution of C++ code looking for FwpmEngineOpen0 and FwpmFilterEnum0, the system will perform a complete enumeration of the active WFP filters. For each filter the Filter KEY(GUID), Action(FWP_ACTION_BLOCK / FWP_ACTION_PERMIT), Sublayer GUID and altitude, Condition structures (FWPM_FILTER_CONDITION0) and Provider identity will be gathered, and a baseline image will be created. Core windows library like Fwpuclnt.lib will be used for WFP communication and wevtapi.lib for event log integration. Key functions include - FwpmEngineOpen0() – Establish connection to BFE; FwpmFilterEnum0()- Inspect active filters; FwpmFilterDeleteByKey0() – delete malicious filters; EnableTraceEx2() – Subscribe to WFP ETW trace for live monitoring; ReportEventW() – log alerts with severity level.

3.4 Experimental Setup:

The test will be executed in a controlled experimental setup in VMware to ensure isolation and reproducibility. The test VM Target will include the following:

- Target VM: Windows 10 or Windows 11 which will run the DRS prototype and if time persist using trial version EDR agent like Bitdefender will be evaluated.

- Also, a malicious WFP C++ WFP payload will be sent to add or modify WFP filter and simulate layering conflicts from the Target machine
- Baseline Capture: A reference snapshot of legitimate active filters will be taken before each test using ./WFPHealthMonitor.exe –create-baseline parameter

The following experience scenarios will be tested:

Scenario	Action	Expected Result
Malicious silencing	Inject DROP filter targeting EDR processes	Detect and generate alert/logs for user/admin
Accidental Misconfiguration	Admin by mistake add legitimate block rule	Detect and generate warning alert; no action taken
Normal Operation	No change to WFP/BFE	No alert; Baseline integrity intact

3.5 Tools Used:

Tools or Framework	Goal
Visual C++ (Microsoft Visual Studio)	Development of prototype to inject malicious WFP rule focusing on the ALE Layer
Windows Filtering Platform (WFP API)	Manipulation of BFE filters
PowerShell	Alerting mechanism as alert pop up on screen
Task Scheduler	Required to automate the process execution
Windows Event Viewer	Alerting administrators
Windows Event tracing (ETW)	Real time logs for rule changes
VMware workstations	Deploy windows 10 for emulating attack and developing remediating tactics
Wireshark	Monitor network level activity of filters

3.6 Risk and Ethical consideration:

All said experiments will be conducted in controlled and isolated virtual lab environment with no relation to productions environment. It does not involve handling personal or enterprise sensitive data. All scripts are digitally signed, ensuring access to only controlled environment. The experimental study solely focuses on a defensive research approach strategy and follows the regulation of ethical standards security experimentation.

4 Architecture Diagram:

WFP and BFE architecture:

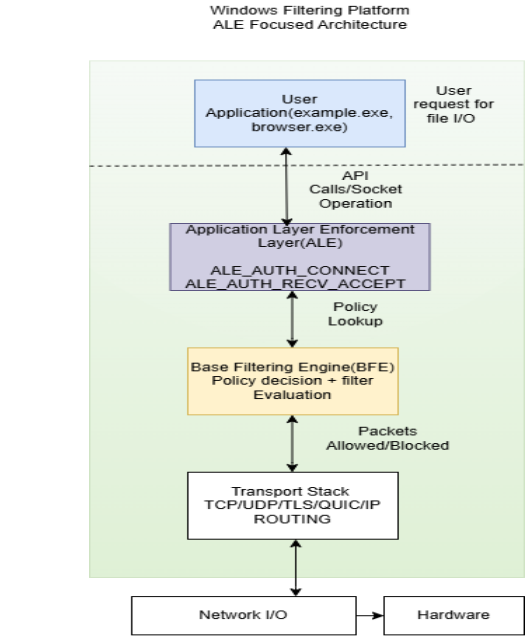

Figure 1: WFP architecture

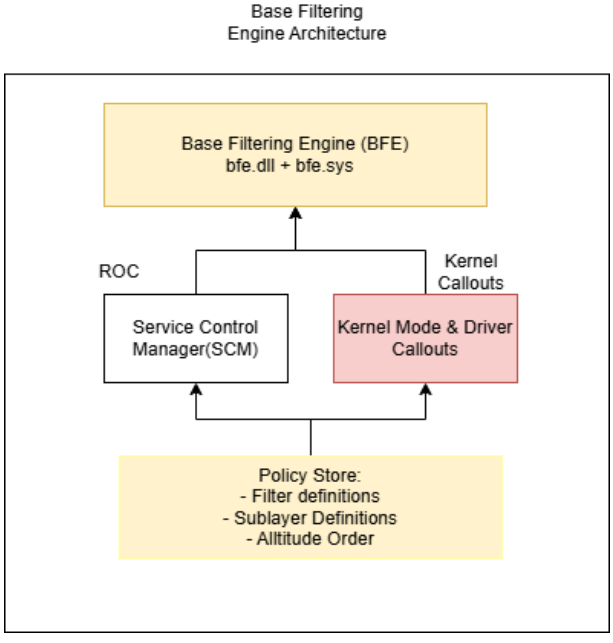

Figure 2: BFE architecture

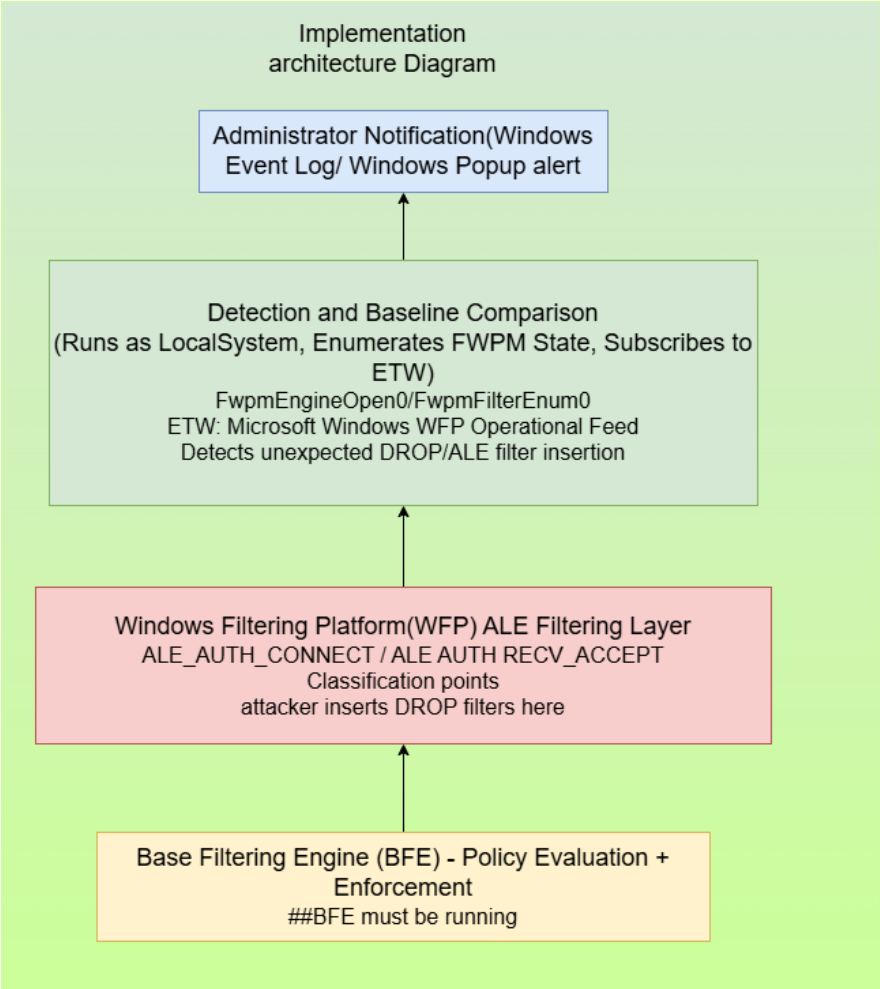

Figure 3: Implementation architecture

5. Implementation:

5.1 System overview: The entire prototype was created on a user-mode C++ application designed for inspecting firewall configuration state that is maintained by the Windows Filtering Platform (WFP) and Base Filtering Engine (BFE). The project does not rely on network packet capture or intrusive kernel drivers, instead it takes a configuration-driven detection strategy, which checks live firewall rules that determine host network behavior. The design option provides a high level of detection during the cases of logs of the traffic being disabled, encrypted, or made unreadable and it is also ideal in terms of aligning the enterprise security products with the local system policies.

5.2 Architecture components: The core components of the system is built around Microsoft's documented WFP management APIs that are available via **fwpuclnt.dll.** Utilizing **FwpmEngineOpen0** function a handle to base filtering engineer is created, after which the filter store is enumerated using WFP API call from **FwpmFilterCreateEnumHandle0** and **FwpmFilterEnum0.** Each filter representation is done internally by a **FWPM_FILTER0** structure, which includes every uniquely identified filter key, its provider and layer identifiers, evaluating the weight, conditions and enforcement actions. The project selectively processes those filters only whose action type equals **FWP_ACTION_BLOCK** which also includes an **FWPM_CONDITION_ALE_APP_ID** condition. The **ALE_APP_ID** field illustrates the

executable path in which the Block rule is relevant and then is saved in the form of a byte blob. The application then unpacks the structure into human readable NT Device path (i.e. such as, \Device\Harddisk\Volume3\Windows\System32\svchost.exe) hence binding firewall rules and applying them directly to application id.

5.3 WFP Integration and Filter Enumeration: To analyse the configuration drift, the system integrates a baseline model in the working directory, which during initial deployment, the application is run in baseline creation mode, which enumerates all current blocked application. Identifies and stores them in a text file (BlockedAppsBaseline.txt). Each line of the baseline would consist of the application path and how many block rules are associated with each of them. The program initialize a session with BFE by calling the following: FwpmEngineOpen0(NULL, RPC_C_ AUTHN_WINNT, NULL, NULL and engine); Filter are now listed in pages o 64 with: **FwpmFilterenum0(engine,enumhandle, 64, &filters, &numreturned)**; Each **FWPM_FILTER0** is interrogated to seek Action type - **FWP_ACTION** block; Presence of application identity **FWPM_condition ALE_AUTH** connect **IPforward** and lastly **Filter Metadata - Weight, Provider, Conditions**. **FPW_BYTE_BLOB** payloads are also extracted, and the executable code is then decoded into Unicode pattern where it can be better understood by humans. To make the system more resilient to rule duplication and other policy noises, the baseline purposely captures application identity as opposed to raw filter structures so that when an attacker adds a series of overlapping rules to the rule set, the presence of the executable in the rule set is counted as an event; thus, it is possible even to have an overlap between a rule set and a code segment to count as a single event.

5.4 Baseline generation and Drift detection:
In detection mode that uses baseline generation, the application loads the baseline into the system memory in the form of a set of hash where comparisons with real time against the lates firewall configurations are performed. The parameter *WFPHealthMonitor.exe --create-baseline* is used to create the baseline on an elevated prompt or powershell to give the basis. Such reasoning prior to each execution is a guarantee of new enforcement instead of the noisy alteration of the rules. The application also aggregates all current block rules by executable name and the path/directory and prints group summary with the number of block filters present in each process. This provides analytical advantage to gain insight into which binaries are aggressive policy control. One such example of baseline info would be*:\Device\HarddiskVolume3\Windows\System32\omadmclient.exe*

5.5 Logging and Event gathering:
Beyond detection, the project includes the layered logging mechanism to ensure forensic delivery and traceability. File based logging and Event tracing logging ensures structured, timestamped entries to a persistent log file (WFP_monitor.log) that is present in the working directory of the application, which categorizes messages using severity labels such as **INFO**, **ALERT** and **ERROR**. For the event tracing logs, the application writes information in Application logs in event viewer with source=WFPHealthMonitor, using the **ReportEventW** API. Critical information and errors are then transmitted into user readable form, which can later be made available for SIEM tools without any additional integration effort. This robust design ensures the system to operate as both a standalone monitor and as a telemetry source in an enterprise incident detection procedure.

5.5.1 Event Classification:

Level	Event ID	Definition
INFO	1000	Startup, enumeration, baseline loaded
ALERT	2001	New blocked application detected
ERROR	2002	Write errors, engine failure

Screenshot from event viewer

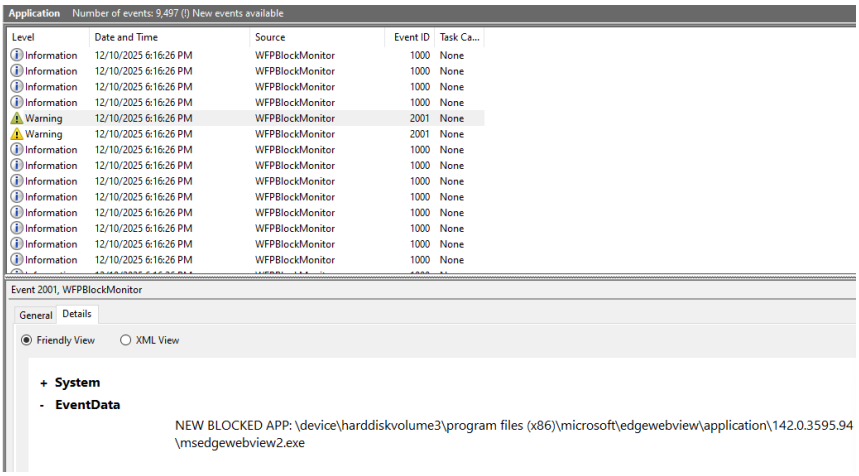

Figure 4: Event viewer logs

5.6 Forensic logging availability:

Along with the logs important information is also included such as conditional detailed logging for forensic analysis. Upon detection of new executable relative to baseline, the application automatically dumps the full internal structure of the new block filter. Information like the **filterKey, providerKey, layerKey, subLayerKey, weight, effectiveWeight, action type**, are listed with all filter conditions. This information enables analysis to understand where in the network, stack enforcement occurs, which provider is the owner of the rule, and how priority conflicts are resolved. The design specifically only outputs the details of the new filter that is detected in order to reduce log flood and maintain signal to noise ration.

5.7 Automation: To automate unattended execution, the application is designed to operate on its own using Windows Task Scheduler with elevated privileges. The Windows task scheduler documentation [26] is followed to automate the execution and produce an alert pop up if new filter is detected. The ability to turn this prototype into a deployable defensive mechanism in enterprise solutions, transforms the project into more vendor centric while helping academic researchers to contribute to EDR security.

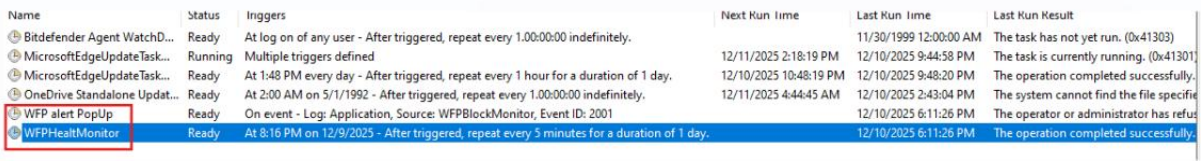

Figure 5: Task scheduled for WFP pop alert and automatic execution of application.

6. Test Results:

Several tests were conducted to validate functional correction, detecting accuracy and overall resilience under real time rule injection.

6.1 Baseline accuracy:
Initial Capture is focused on confirming that the Windows 10 virtual machine application could enumerate existing firewall rules, system policies and application blocked entries. Multiple extracted blocking rules applied to system executables such as deviceenroller.exe, omadmclient.exe, and dmcertinst.exe are captured. These files handle device management and enterprise enrolment. The results shows that the implementation catches both user-created and policy-based enforcement rules just like it should.
After the invoking the application with –create-baseline switch a baseline file was generated in the working directory that contains exactly one line per unique WFP filters attached to a process. Hence, it shows the model's application-centric nature that works as designed and does not capture redundant entries.

```
2025-12-09 21:12:25 [INFO] Logger initialized, log file: WFP_Monitor.log
2025-12-09 21:12:25 [INFO] Running in baseline creation mode.
2025-12-09 21:12:25 [INFO] Opening BFE/WFP engine...
2025-12-09 21:12:25 [INFO] Enumerated 22 BLOCK filters with ALE_APP_ID conditions.
2025-12-09 21:12:25 [INFO] Creating baseline file: BlockedAppsBaseline.txt
2025-12-09 21:12:25 [INFO] Baseline written with 7 unique blocked application paths to BlockedAppsBaseline.txt
2025-12-09 21:12:25 [INFO] === Grouped summary: applications with BLOCK rules ===
2025-12-09 21:12:25 [INFO] App Path: \device\harddiskvolume3\windows\system32\deviceenroller.exe | Block rules: 4
2025-12-09 21:12:25 [INFO] App Path: \device\harddiskvolume3\windows\system32\dmcertinst.exe | Block rules: 4
2025-12-09 21:12:25 [INFO] App Path: \device\harddiskvolume3\windows\system32\settingsynchost.exe | Block rules: 2
2025-12-09 21:12:25 [INFO] App Path: \device\harddiskvolume3\windows\system32\searchprotocolhost.exe | Block rules: 2
2025-12-09 21:12:25 [INFO] App Path: \device\harddiskvolume3\windows\system32\searchfilterhost.exe | Block rules: 4
2025-12-09 21:12:25 [INFO] App Path: \device\harddiskvolume3\windows\system32\omadmclient.exe | Block rules: 4
2025-12-09 21:12:25 [INFO] App Path: \device\harddiskvolume3\program files (x86)\incredibuild\coordinatorservice.exe | Block rules: 2
```

Figure 6: Running WFPHealthMonitor.exe –create-baseline

6.2 Manual Rule Injection test: To validate the detection log several new firewall rules were created manually via the **nesth advfirewall firewall add rule** command. On a clean system without any applications or central policy enforced, new block rules were created for processes like edge.exe, svchost.exe. Running the WFPHealthMonitor.exe immediately detects the presence of a new filters and write the information to WFP_monitor.log and event viewer along with an automatic toast notification from WFP Filter. Checking the WFP_monitor. Log will present messages like ALERT with the entry of the new application that was added to the block rule. Additionally, the output produced for each new detection presents valuable information about the filter such as showing filter identifiers, provider ownership, assigned layers, and condition values. Below are some example line from blockedappsbaseline.txt and WFP_monitor. Log

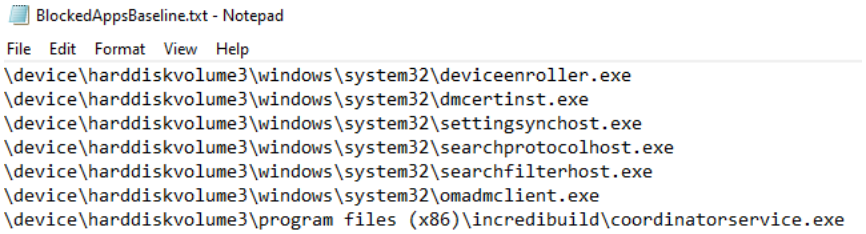

Figure 7: BlockedAppsBaseline.txt

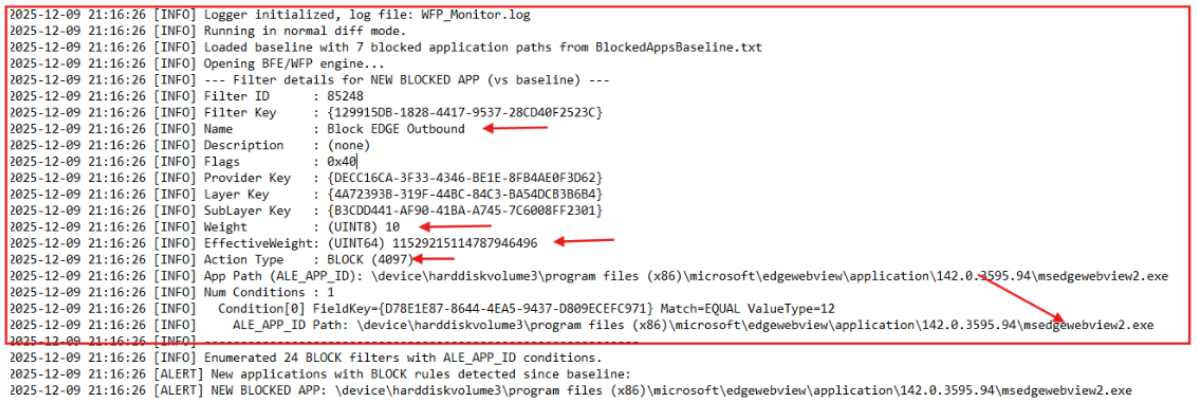

Figure 8: WFP_monitor. Log

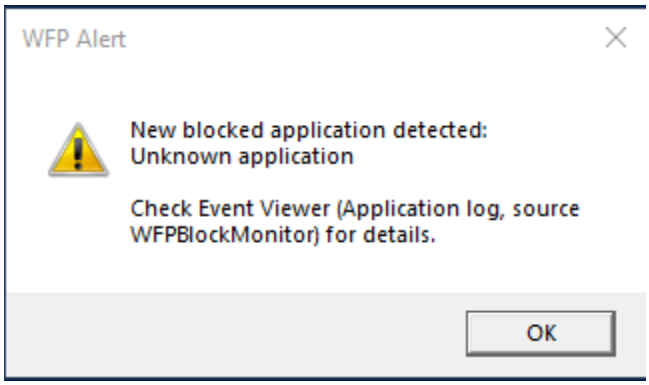

Figure 9: Alert pop up

6.3 Central Policy Observation: In situations where central control of the policies by Antivirus software or an external firewall like Bitdefender was intervening, the test results have shown a fundamental difference in behaviour. While netsh would show that the rule was created successfully, the newly added rules did not seem to be there in the WFP filters list. Hence our monitoring tool would quite rightly have shown no new output for blocked rules. Manually checking the Firewall Profile with Get-NetFirewallProfile showed that policy disabled local firewall rules (AllowLocalFirewallRules = False). Therefore, the application does actually shed some light on the true status of protection, not just what's on the paper. If it does not detect anything in these situations, it is because there is genuine policy enforcement, not because it has missed something.

6.4 Event Viewer Validation:

Testing the logging of files and the integration with the Windows Event Log: It was also performed by running the tool the usual way and, of course, the weird way. Each time the tool records a message with a date stamp in the log file, meaning you do have a history that will remain there, regardless of how often the tool is used. We also saw messages recorded in the Event Viewer with the name of the source that we designated. This is the data that will show the logging system works regardless of the console.

The Grouped Summary Testing validated the correct grouping of the rule counts for each application. The results provided accurate data for the number of block rules that were applied against the same executable, which helped identify that the logic for grouping was

operating correctly. This helped identify the stacking of the policies, particularly for the services that have been blocked by various layers of rules.

7. Evaluation:

The system was developed aimed principally at the scan for misconfiguration regarding high level Firewall restrictions. Skippable firewalls or other higher level administrative tools are bypassed by this technique. It simply examines the system from exactly the same perspective as the platform enforcement engine of the operating system. This is where a perfect match between the reported results, and the functionality of the system, is of the utmost importance. Baseline comparison offers a reliable method whereby any variation regarding configurations, and policies, is subject to scrutiny, which could only come about by unauthorized modifications of the norm within the business, or protective, environment wherein the attacker will attempt quietly to alter the host protective mechanism.

Architectural simplicity, along with high fidelity. This is one of the most important factors for the implementation s success. Operating completely in user mode, using only the supported WFP interfaces, the solution is very robust and windows version-compatible. This is not like other packet inspection tools that require kernel mode privileges higher than user level, and hence it is less risky. By combining these logs, the quality of life of the solution will also get better. By utilizing the windows logging facilities, the users are provided with long-term SIEM ingestion.

Furthermore, with the program's help, forensic depth is made available. The complete architecture of the filter can be shown to security engine. Thus, they can see not only what a rule does but also how and where it enforces policy within stack. This level of detail is crucial when you are responding to an incident but often goes away; it is abstracted out by commercial firewall management tools.

Nonetheless, it is important to bear in mind that there are some limitations of this system: it emphasizes hardware configuration more than actual implementation; does not know whether traffic has actually been cut off and so cannot always correctly notify users. Also can spot applications getting blocked at the application level, but not right now take ip-based or port-based rule makers unless there is an * in front of each of them with an executable that embodies all instructions for accomplishing whatever the maker means A further limitation is the inherent sensitivity of the baseline's hashing algorithm to the way it represents a path; if the device mappings change, even computation-executable programs which have the same cryptographic hash and therefore identical program IDs may appear in completely different parts or positions within memory. In future further enhancements could normalise paths or resolve NT device paths into canonical Win32 paths.

This is another major study of systems being managed centrally and problems: if the rules governing a local firewall are all disabled by policymakers at their control center as such conditions do not exist in any good-regulated country for instance there cannot be any leaked test rule even though the program may have been installed which means that one never appears in WFP to find exceptions and therefore none are observable.

While this may appear initially like a limit how accurate it is couldn't but underscores the appropriateness of this choice: WFP tells you what your system really does, not what overworked system admin sloppily asked for.

This solution overall does meet the original design goal, with only minor damage and largely accurate intelligence set up, and very good audit trail. It is practical, expandable and technically consistent with the way today's commercial endpoint protection products monitor host policy state. If given additional work in further runtime telemetry, hash-based rule verification, or collective baseline management, this kernel package could become a

configuration monitoring agent suitable for general development into production-ready software.

8. Conclusion:

This paper proposed the design and assessment of the configuration-based monitoring system for the Windows Filtering Platform (WFP), which is useful for the identification of the manipulation of the firewall and the divergence of the host-based policy. It is clear that the status of enforcement and the host firewall policy can also be obtained directly using the Base Filtering Engine (BFE) through the native windows API. This, in return, removes the need for specific details of the WFP that are actually running. Based on the mere observation of the firewall behavior on the policy level, rather than the behavior on the network level, it also advocated that the silent defensive downfall could actually be identified in the absence of suspicious/malicious packets.

We were also successful in implementing a model that enables the monitoring of newly blocked applications. The moral of the matter is that the level of noise is lower when monitoring the ordinances of the applications compared to monitoring the rules of the applications, and the interpretations are considerably better. This approach also asserted that Configuration Integrity Monitoring (CIM) is a very good add-on tool for monitoring endpoint security-related events. This approach is very effective when it comes to monitoring the malicious activity that intends to thwart the enforcement of security enforcement technology.

However, the new logging architecture also made the system not only functional, since it had the capability of logging or resource sharing with very minimal labor invested in the maintenance of the system, but also provided for a form of observability that is expected of systems that operate within the environment of the enterprise, where kernel drivers, or rather the APIs that do not support Linux, were not used. Additionally, the alerts that existed within the previous windows telemetry pipes were also easy to integrate for analysis using the Security Information and Event Management (SIEM) system.
This paper, on the whole, reveals that the monitoring of the policy configuration through WFP is one of the approaches that help identify the hidden changes. This project verifies the effectiveness of this technique of revealing the firewall cheating by comparing it with a minimum standard. It also emphasizes the importance of the Windows Filtering Platform, which is a remarkable resource for host-related security information.

9. Limitations:

In spite of the advantages, the system also suffers from limitations.
Firstly, at the current stage, the implementation is limited only to the application firewall policies via the condition ALE_APP_ID. The blocked rules that contain only IP addresses, ports, services, or interfaces and not the specific executable will not be detected. Even though it was a deliberate choice to emphasize the issue of application suppression attacks, it cannot trigger other firewall misconfigs.
Secondly, the architectural design features a model that is polling-based, not real-time subscription. On one side where the polling works well, it will not detect the moment when changes happen. However, the events between the different time polls could exist for a short time until it is detected, creating a small loophole for the attacker to get into the system.

Thirdly, the baseline mode that employs the use of the executable NT paths as identifiers, which brings fragility when there are changes, for example, resizing of disks or movement of installed software. Such changes bring about the inclusion of benign applications that will show up within new entries, which will, in the process, produce false positives. These characteristics points out the weakness of the baseline systems that only make use of the naming conventions and not the cryptographic identity.
Fourth, the system will always require administrative privileges when it polls WFP filters. Though technically necessary because of the restrictions imposed by the security environment on the Windows platform, it somewhat hinders the use of the tool on the least privileged environment, other than when running it as a service.
Lastly, the system reports on configuration observance, failing, however, to shed light on the effectiveness of enforcement. The proposed system cannot verify if the traffic that failed was blocked by configuration or attempted by the application during runtime. Consequently, the proposed system turns out to offer complementarity for monitoring traffic instead of replacing it.

10. Future Work:

There are several scopes that are still valid for the expansion of the system from the stage of a research prototype into that of a production-ready system.
One of the major features of the enhancement of the future will be the use of real-time ETW-based logging and monitoring using the WFP operational event providers. This will ensure that the system will be enabled to detect the changes in policies when the events take place, instead of waiting for the scheduled scan. This enhancement of the future will also provide the correlation of the firewall block rules, which will be added for specific executables when they are running.
The next important aspect is identity hardening. This version will use the executable paths only, but the next version will add features like computing the cryptographic hashes of the binaries and monitoring the applications on the basis of trusted signatures and hashes. This will completely shift the paradigm of how this tool will identify new rules.
Increased awareness of the policy could also form a new level of enhancement, where the resolution of provider identifiers for trusted Microsoft and third-party vendors. Through the strict definition of the rule origin, for example, Defender, Intune, EDR Softwars, VPN, the system will be able to identify the differences between the enforcement of the policies and the local suspicious activity.
Another significant improvement that could be made is that the integration of logs should be centralized through the external log ingestion SIEM tool pipeline. Event data could also be exported in xml or JSON format for easier readability. This allows the organization or enterprise to align endpoint detection telemetry data with user activity.
Finally, extending the analysis beyond the level of filters on the application level to also include rules on the IP layer, transport level rule, and service-based rules would enhance the functionality of the firewall and the objective. This minor aspect would ensure that the prototype is actually a real-time environment for analyzing the firewall rules rather than merely a specific tool for detecting changes.

References:

1. H. C. Junior, “HookChain: A new perspective for Bypassing EDR Solutions,” *arXiv.org*, Apr. 04, 2024. https://arxiv.org/abs/2404.16856
2. Chatzoglou, E., Karopoulos, G., Kambourakis, G. & Tsiatsikas, Z., “Bypassing antivirus detection: old-school malware, new tricks,” *arXiv preprint arXiv:2305.04149*, May 2023
3. G. Karantzas and C. Patsakis, “An Empirical Assessment of Endpoint Detection and Response Systems against Advanced Persistent Threats Attack Vectors,” *Journal of Cybersecurity and Privacy*, vol. 1, no. 3, pp. 387–421, Jul. 2021, doi: 10.3390/jcp1030021.
4. M. E. Ahmed, H. Kim, S. Camtepe, and S. Nepal, “Peeler: Profiling Kernel-Level Events to detect Ransomware,” *arXiv.org*, Jan. 29, 2021. https://arxiv.org/abs/2101.12434#:~:text=In%20this%20paper%2C%20we%20present%20a%20novel%20ransomware,generic%20characteristics%20of%20ransomware%20depicted%20at%20the%20kernel-level.
5. A. J. Rose, “ScriptBlock Smuggling: Uncovering stealthy evasion techniques in PowerShell and .NET environments,” *AFIT Scholar*. https://scholar.afit.edu/facpub/1519/
6. F. Yang, Y. Han, Y. Ding, Q. Tan, and Z. Xu, “A flexible approach for cyber threat hunting based on kernel audit records,” *Cybersecurity*, vol. 5, no. 1, Jun. 2022, doi: 10.1186/s42400-022-00111-2.
7. M. Landauer, L. Alton, M. Lindorfer, F. Skopik, M. Wurzenberger, and W. Hotwagner, “Trace of the Times: Rootkit Detection through Temporal Anomalies in Kernel Activity,” *arXiv.org*, Mar. 04, 2025. https://arxiv.org/abs/2503.02402#:~:text=The%20framework%20outlined%20in%20this%20paper%20injects%20probes,and%20uses%20statistical%20tests%20to%20detect%20time%20shifts.
8. R. Uetz *et al.*, *You Cannot Escape me: Detecting evasions of SIEM rules in enterprise networks*. 2024. [Online]. Available: https://www.usenix.org/system/files/usenixsecurity24-uetz.pdf
9. K. M. Shulika, D. S. Balagura, and Z. M. Sydorenko, “Analysis of methods for bypassing modern EDR endpoint protection systems,” *Radiotekhnika*, vol. 2, no. 217, pp. 64–68, Jun. 2024.
10. “PAD: towards principled Adversarial Malware Detection against evasion attacks,” *IEEE Journals & Magazine | IEEE Xplore*, Apr. 01, 2024. https://ieeexplore.ieee.org/document/10097506
11. Y. Sao and Sk. S. Ali, “Security analysis of scan obfuscation techniques,” *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 2842–2855, Jan. 2023, doi: 10.1109/tifs.2023.3265815.
12. D. C. D’Elia, E. Coppa, F. Palmaro, and L. Cavallaro, “On the Dissection of Evasive Malware,” *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 2750–2765, Jan. 2020, doi: 10.1109/tifs.2020.2976559.
13. M. Hand, “Evading EDR,” *O’Reilly Online Learning*. https://learning.oreilly.com/library/view/evading-edr/9781098168742/xhtml/chapter1.xhtml#h-7
14. M. Graeber and L. Christensen, “Subverting sysmon: application of a formalized security product evasion methodology,” conference-proceeding, 2018. [Online].

Available: https://specterops.io/wp-content/uploads/sites/3/2022/06/Subverting_Sysmon.pdf
15. K. Elastic, “Potential Evasion via Windows Filtering Platform,” Elastic Security — Prebuilt detection rule, 2025. Available: https://www.elastic.co/docs/reference/security/prebuilt-rules/rules/windows/defense_evasion_windows_filtering_platform
16. R. Ben Yizhak, *#NoFilter: Abusing Windows Filtering Platform for privilege escalation*. 2021. [Online]. Available: https://media.defcon.org/DEF%20CON%2031/DEF%20CON%2031%20presentations/Ron%20Ben-Yizhak%20-%20NoFilter%20Abusing%20Windows%20Filtering%20Platform%20for%20privilege%20escalation.pdf
17. EliotSeattle, “Allocated Filter Altitudes - Windows drivers,” *Microsoft Learn*. https://learn.microsoft.com/en-us/windows-hardware/drivers/ifs/allocated-altitudes
18. Stevewhims, “Windows Filtering Platform - Win32 apps,” *Microsoft Learn*. https://learn.microsoft.com/en-us/windows/win32/fwp/windows-filtering-platform-start-page
19. H. S. Mahajan, “Filtering engine model for VIMNet,” M.S. thesis, The University of Toledo, Toledo, OH, 2023.
20. https://static.googleusercontent.com//42189.pdfmedia/research.google.com/en//pubs/archive/42189.pdf
21. Ciholas P, Such JM, Marnerides AK, Green B, Zhang J, Roedig U. Fast and Furious: Outrunning Windows Kernel Notification Routines from User-Mode. Detection of Intrusions and Malware, and Vulnerability Assessment. 2020 Jun 11;12223:67–88. doi: 10.1007/978-3-030-52683-2_4. PMCID: PMC7338165.
22. T. Imbert and Synacktiv, “Windows Kernel Security: A Deep Dive into Two Exploits Demonstrated at Pwn2Own,” Jul. 2023. [Online]. Available: https://conference.hitb.org/hitbsecconf2023hkt/materials/D2T1%20-%20Windows%20Kernel%20Security%20-%20A%20Deep%20Dive%20into%20Two%20Exploits%20Demonstrated%20at%20Pwn2Own%20-%20Thomas%20Imbert.pdf
23. K. Pundeer, “Host-Based Malware Analysis,” M.S. thesis, University of Alberta, Edmonton, AB, 2020.
24. Pat_H/to/File, “Getting more out of the Windows Filtering ETW Events,” *pat_h/to/file*, Oct. 31, 2020. https://blog.tofile.dev/2020/10/31/wfp.html
25. T. Strohmann, “DSR Approach Models | Design Science Research,” Design Science Research, Oct. 17, 2024. /https://design-science-research.de/en/course/dsr-questions/vorgehensmodelle/
26. Stevewhims, “Task Scheduler for developers - Win32 apps,” Microsoft Learn. https://learn.microsoft.com/en-us/windows/win32/taskschd/task-scheduler-start-page
27. K. Parveen, “Advanced Techniques of Malware Evasion and Bypass in the Age of Antivirus,” *International Journal for Electronic Crime Investigation*, vol. 8, no. 3, Sep. 2024
28. Decoding Malicious Camouflage: Analysing Evasion Techniques in Malware Against EDR and AMSI Detection,” *ResearchGate*, 2025.

29. J. Heibrink s1040183, “Antivirus and EDR bypasses for initial access.” [Online]. Available: https://www.cs.ru.nl/bachelors-theses/2023/Jorn_Heibrink___1040183___Antivirus_and_EDR_bypasses_for_initial_access.pdf