المدرسة الوطنية للعلوم التطبيقية - مراكش

ECOLE NATIONALE DES SCIENCES APPLIQUEES - MARRAKECH

MARRAKECH
جامعة القاضي عياض
UNIVERSITÉ CADI AYYAD

THESIS OF THE END-OF-STUDY PROJECT
TO OBTAIN THE STATE ENGINEERING DIPLOMA
Major : Cyber Defense and Embedded Telecommunication Systems

Bypassing Application Whitelisting : Hardening AppLocker on Windows

---

Authored by :
Miss Hajar ED-DARRAJY

Supervised by :
Mr. Marouane BENCHIEKH
Pr. Said BENKIRANE

Defended on June 18, 2025, before the jury composed of :

Pr. Said BENKIRANE	EST Essaouira	- President
Pr. Aziz DAROUICHI	FST Marrakech	- Evaluator
Pr. Anas HATIM	ENSA Marrakech	- Examiner

Academic Year : 2024/2025

Dedication

I would like to express my deepest gratitude to the individuals who have supported and accompanied me throughout this journey :

To my beloved parents, no words are enough to thank you for your endless sacrifices, your patience, and your unconditional love. Your support has been the foundation of every step I've taken, and I dedicate this work to you as a symbol of my deepest respect and gratitude.

To my siblings from my eldest brother to my youngest sister, thank you for always believing in me and standing by my side. Your encouragement and presence have meant more than you can imagine.

To my friends Imane Chaik and Imane Benchakroun thank you for the laughs, the late-night study sessions, the teamwork, and the support through both good and challenging times.

To all my professors and teachers who have guided me along this academic path, I extend my sincere thanks. Your knowledge and dedication have shaped not only my education but also my mindset and curiosity. A special thanks goes to all the professors of the GCDSTE branch at the National School of Applied Sciences of Marrakech.

Acknowledgment

First and foremost, I would like to thank Allah for giving me the strength and patience to complete this work.

I would like to sincerely thank LMPS Group for welcoming me during my internship and giving me the opportunity to learn in a professional and stimulating environment. A special thank you to the Red Team, especially Imane and Soulaymane, for their support, kindness, and guidance throughout this experience.

My thanks also go to my fellow interns, Hajar and Loubna, for their collaboration, encouragement, and the good atmosphere we shared during these months.

I am very grateful to Marouane Benchikh, my company supervisor, for his availability and trust during the internship, and to Mostapha Bahadou, who suggested the topic and helped guide this project.

My deepest appreciation goes to Professor Said Benkiran, my academic supervisor from EST Essaouira, for his attentive guidance, encouragement, and follow-up during all phases of the project.

A special word of thanks to Professor Anas Aboulkalam for creating the GCDSTE branch and giving us, students, the chance to follow such a rich and forward-thinking academic path.

Finally, I would like to thank the members of the jury for their time and for reviewing this work.

Abstract

This report presents the work conducted during my final-year internship within the Red Team of LMPS Group, a cybersecurity-oriented company, as part of my end-of-studies project to obtain the state engineering degree in Cyber Defense and Embedded Telecommunication Systems at the National School of Applied Sciences of Marrakech.

The project focuses on evaluating the security limitations of AppLocker, a native Windows feature used for application whitelisting, and on developing an automated PowerShell-based tool designed to detect and remediate known bypass techniques. Although AppLocker is widely used in enterprise environments, it remains vulnerable to several evasion strategies involving trusted system binaries, scripting engines, and misconfigured or writable directories.

To address these weaknesses, we designed and implemented a PowerShell tool capable of scanning Windows systems for AppLocker misconfigurations and applying hardened rule profiles. These profiles target common bypass vectors such as LOLBAS abuse, writable paths, and alternate data streams (ADS). The tool also supports policy backup, rollback, and the selection of enforcement modes, offering flexibility and practical control to system administrators.

This project illustrates how offensive security perspectives can contribute to the development of effective defensive measures, and highlights the need for tailored AppLocker configurations to strengthen application control in real-world environments.

Keywords : Applocker, Application Whitelisting, Rules, Bypass Techniques, Windows Security, PowerShell Automation, LOLBAS, Hardening.

Résumé

Ce rapport présente le travail réalisé durant mon stage de fin d'études au sein de la Red Team du groupe LMPS, une entreprise orientée cybersécurité, dans le cadre de mon projet de fin d'études pour l'obtention du diplôme d'ingénieur d'État en Cybersécurité et Systèmes Télécoms Embarqués à l'École Nationale des Sciences Appliquées de Marrakech.

Le projet se concentre sur l'évaluation des limitations de sécurité d'AppLocker, une fonctionnalité native de Windows utilisée pour la mise en liste blanche des applications, ainsi que sur le développement d'un outil automatisé basé sur PowerShell, conçu pour détecter et corriger les techniques de contournement connues. Bien qu'AppLocker soit largement utilisé dans les environnements d'entreprise, il reste vulnérable à plusieurs stratégies d'évasion impliquant des binaires système de confiance, des moteurs de script, et des répertoires mal configurés ou accessibles en écriture.

Pour répondre à ces faiblesses, nous avons conçu et mis en œuvre un outil PowerShell capable d'analyser les systèmes Windows à la recherche de mauvaises configurations AppLocker et d'appliquer des profils de règles renforcés. Ces profils ciblent des vecteurs de contournement courants tels que l'abus de LOLBAS, les chemins en écriture, et les flux de données alternatifs (ADS). L'outil prend également en charge la sauvegarde des politiques, le retour arrière, ainsi que la sélection des modes d'application, offrant flexibilité et contrôle aux administrateurs système.

Ce projet illustre comment une perspective en sécurité offensive peut contribuer au développement de mesures défensives efficaces, et souligne la nécessité de configurations AppLocker personnalisées pour renforcer le contrôle des applications dans des environnements réels.

Mots-clés : AppLocker, liste blanche d'applications, règles, techniques de contournement, sécurité Windows, automatisation PowerShell, LOLBAS, durcissement.

ملخص

يعرض هذا التقرير العمل المنجز خلال فترة تدريبي النهائي ضمن فريق "الفريق الأحمر" بشركة إل.إم.بي.إس ، وهي شركة متخصصة في مجال الأمن السيبراني، وذلك في إطار مشروع نهاية الدراسة للحصول على دبلوم مهندس الدولة في "الأمن السيبراني والأنظمة المدمجة للاتصالات" بالمدرسة الوطنية للعلوم التطبيقية بمراكش.

يركز المشروع على تقييم القيود الأمنية لأداة "آب لوكر" ، وهي ميزة مدمجة في نظام التشغيل ويندوز تُستخدم لتقييد تشغيل التطبيقات (تبييض التطبيقات)، بالإضافة إلى تطوير أداة مؤتمتة باستخدام "باور شيل" تهدف إلى الكشف عن تقنيات التحايل المعروفة ومعالجتها. ورغم أن "آب لوكر" يُستخدم على نطاق واسع داخل المؤسسات، إلا أنه يظل عرضة لعدة أساليب تجاوز تستغل برمجيات نظام موثوقة (مثل أدوات لولباس)، أو محركات السكربت، أو مسارات قابلة للكتابة وغير مؤمنة.

لمعالجة هذه الثغرات، قمنا بتصميم وتطوير أداة تعتمد على "باور شيل" تقوم بفحص إعدادات "آب لوكر" في أنظمة ويندوز، وتطبيق ملفات قواعد مشددة. تستهدف هذه القواعد تقنيات شائعة مثل استغلال أدوات لولباس، المسارات القابلة للكتابة، وتيارات البيانات البديلة. كما تتيح الأداة أيضًا وظائف نسخ احتياطي للسياسات، وإمكانية الاسترجاع، واختيار أوضاع التنفيذ المختلفة، مما يمنح مسؤولي الأنظمة مرونة وتحكمًا متقدمًا.

يُبرز هذا المشروع كيف يمكن للرؤية الهجومية في مجال الأمن السيبراني أن تُسهم في تطوير حلول دفاعية فعّالة، كما يؤكد على أهمية إعداد مخصص ومشدد لـ "آب لوكر" لتعزيز الرقابة على تشغيل التطبيقات في البيئات الواقعية.

كلمات مفتاحية : آب لوكر، تجاوز السياسات، الدفاع السيبراني، استغلال البرمجيات الموثوقة، قواعد أمنية، تقنيات التحايل، أمن ويندوز، باور شيل، لولباس، تعزيز الحماية.

Contents

List of Figures

List of Tables

Acronymes

ADS *Alternate Data Stream*

CLI *Command Line Interface*

DLL *Dynamic-Link Library*

FIM *File Integrity Monitoring*

GPO *Group Policy Object*

GRC *Governance, Risk, and Compliance*

ID *Identifier*

ISO *International Organization for Standardization*

LOLBAS *Living Off the Land Binaries and Scripts*

MSI *Microsoft Installer*

PAM *Privileged Access Management*

PASSI *Prestataire d'Audit de la Sécurité des Systèmes d'Information*

PCI DSS *Payment Card Industry Data Security Standard*

PCI QSA *Payment Card Industry Qualified Security Assessor*

PE *Portable Executable*

RT *Windows Runtime*

SCADA *Supervisory Control and Data Acquisition*

SIEM *Security Information and Event Management*

SOC *Security Operations Center*

UWP *Universal Windows Platform*

VSTO *Visual Studio Tools for Office*

WDAC *Windows Defender Application Control*

XML *Extensible Markup Language*

General Introduction

In recent years, the increasing complexity of cyber threats has pushed organizations to reinforce the security of their digital infrastructures. Among the many strategies used to protect systems and data, application control has gained significant importance, particularly in environments where endpoint protection is critical. Application whitelisting, which consists of explicitly defining which applications are allowed to run on a system, is a preventive measure that offers a strong barrier against unauthorized software execution and many forms of malware, including fileless attacks.

Microsoft Windows, one of the most widely used operating systems in enterprise environments, includes a native feature for application whitelisting known as **AppLocker**. Introduced to help organizations enforce software control policies, AppLocker enables administrators to define rules that restrict the execution of applications based on criteria such as file path, publisher, or hash. While conceptually powerful, the real-world effectiveness of AppLocker often depends on how well it is configured and maintained.

Despite being included in enterprise Windows versions and offering clear security benefits, AppLocker is frequently underutilized or misconfigured. In many cases, it is deployed in audit mode only, or implemented using default policies that fail to address known attack techniques. Moreover, various bypass strategies have been discovered over time, allowing attackers to execute code even in environments where AppLocker is active. These techniques take advantage of system components that are commonly trusted or of directories that are improperly secured, exposing a clear gap between AppLocker's theoretical protection and its practical enforcement.

This situation raises a fundamental question about the reliability of application whitelisting as a standalone defense mechanism in Windows environments. It highlights the need for a deeper understanding of AppLocker's internal workings, its limitations, and the conditions under which it can be circumvented. By exploring this problematic, we place ourselves in the broader context of system hardening and post-exploitation defense ,two essential aspects of modern cybersecurity.

To guide the reader through this subject, the report is structured into several chapters. It begins with an overview of the host company and the context in which the project was carried out. The following chapter presents the state of the art, focusing on the theoretical foundations of AppLocker, application whitelisting, and common bypass techniques. Then, a chapter is dedicated to practical case studies demonstrating how AppLocker can

be evaded under certain conditions. This is followed by a chapter that covers the project's implementation approach, including the design choices, development phases, and technical elements. Finally, the last chapter concludes the report and outlines potential perspectives for future work.

Chapter 1

General Context of the Project

Introduction

This chapter provides a general overview of the project, beginning with the presentation of the host organization, LMPS Group, a Moroccan leader in cybersecurity services. It then explores the motivation and challenges behind the project, introduces the proposed solution, and outlines the project's objectives, limitations, and planning process. The goal is to set the stage for a deeper technical exploration of AppLocker's security landscape and the development of a custom hardening tool based on PowerShell automation.

1.1 Host Organization

1.1.1 Presentation of LMPS Group

Figure 1.1: LMPS Group

LMPS Group is a Moroccan company specializing in cybersecurity, founded in 2007 by Karim Hamdaoui and based in Casablanca Nearshore Park. The company employs between 51 and 200 people and has earned several key industry certifications, including PCI QSA from the PCI Security Standards Council for PCI DSS, as well as recognition as a Cyber Security Service Provider by SWIFT.

LMPS is also ISO 27001 and ISO 9001 certified, demonstrating a strong commitment to information security and quality management. It operates its own advanced cybersecurity monitoring center "Cyber SOC" and is a member of the FIRST community. The company also holds the PASSI qualification, further reinforcing its credibility in the field.

With years of experience and a skilled team, LMPS Group plays an active role in securing information systems and managing cyber risks. Its services range from security audits and consulting to cybersecurity engineering and training, helping clients build resilience in an increasingly digital world.

1.1.2 Organizational Chart

The organizational structure of **LMPS Group** is built around a clear hierarchy. At the top sits the President, followed by the General Manager, and then the Partner. The company is divided into several departments, including Human Resources, which handles staff management, and Sales, which manages client relations.

LMPS also has specialized teams:

- **White Team** : focused on audits and compliance
- **Blue Team** : responsible for cybersecurity defense
- **Red Team** : dedicated to offensive security (ethical hacking)
- **Technology Team** : handles integration, maintenance, and security solutions

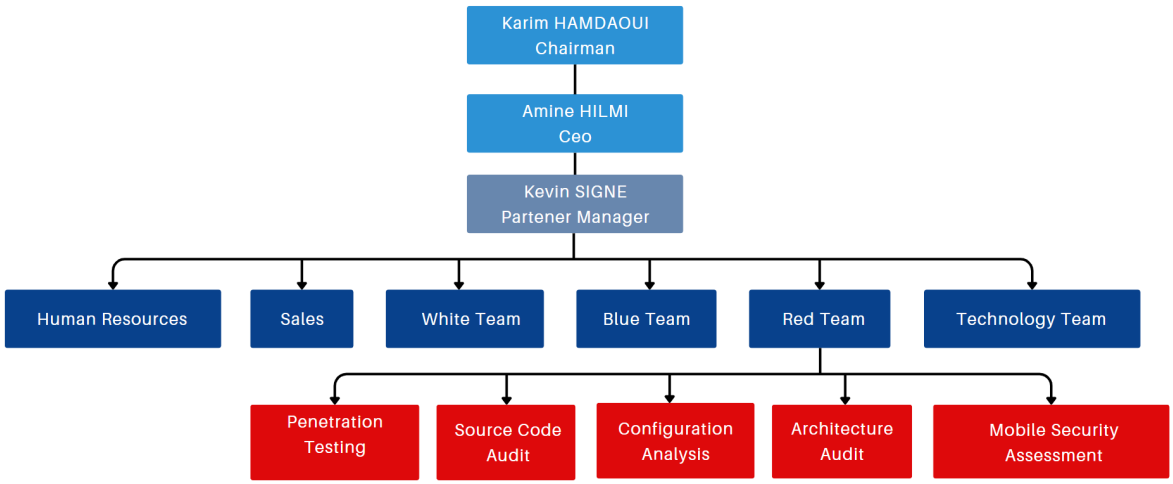

Figure 1.2: LMPS Company Structure

Each department and team plays a crucial role in ensuring the smooth operation of the company, enabling efficient management and a tailored response to various challenges and client needs. This hierarchical structure also supports clear communication and coordination across teams, fostering collaboration and operational efficiency.

1.1.3 Services Offered by LMPS

Figure 1.3: LMPS Services

LMPS Group offers a wide range of services, particularly in the **Advisory field**, which focuses on helping clients continuously manage business risks, improve the effectiveness of their risk management processes, and gradually reduce associated costs. This service area includes compliance audits, technical audits, cybersecurity and forensics, as well as support for achieving and maintaining regulatory compliance.

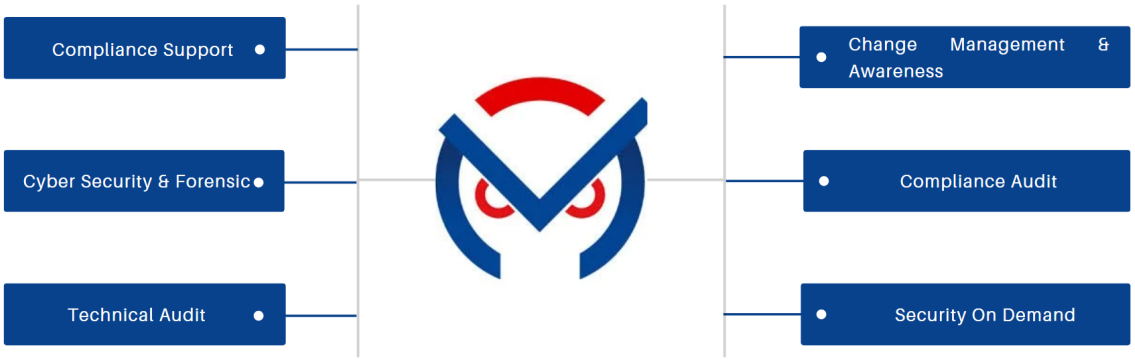

Figure 1.4: LMPS Group's Advisory Business Line Services

LMPS Group also offers a service line known as **Defensive** Services, dedicated to overseeing and managing the information security of its clients. Through its Advanced Security Operations Center, LMPS monitors security components in real time, collects security events, analyzes them, detects anomalies, and initiates appropriate responses when alerts are triggered.

This service is powered by machine learning technologies, enabling the implementation of an expert anomaly detection system based on intelligent algorithms. This approach enhances threat detection capabilities and supports proactive incident response.

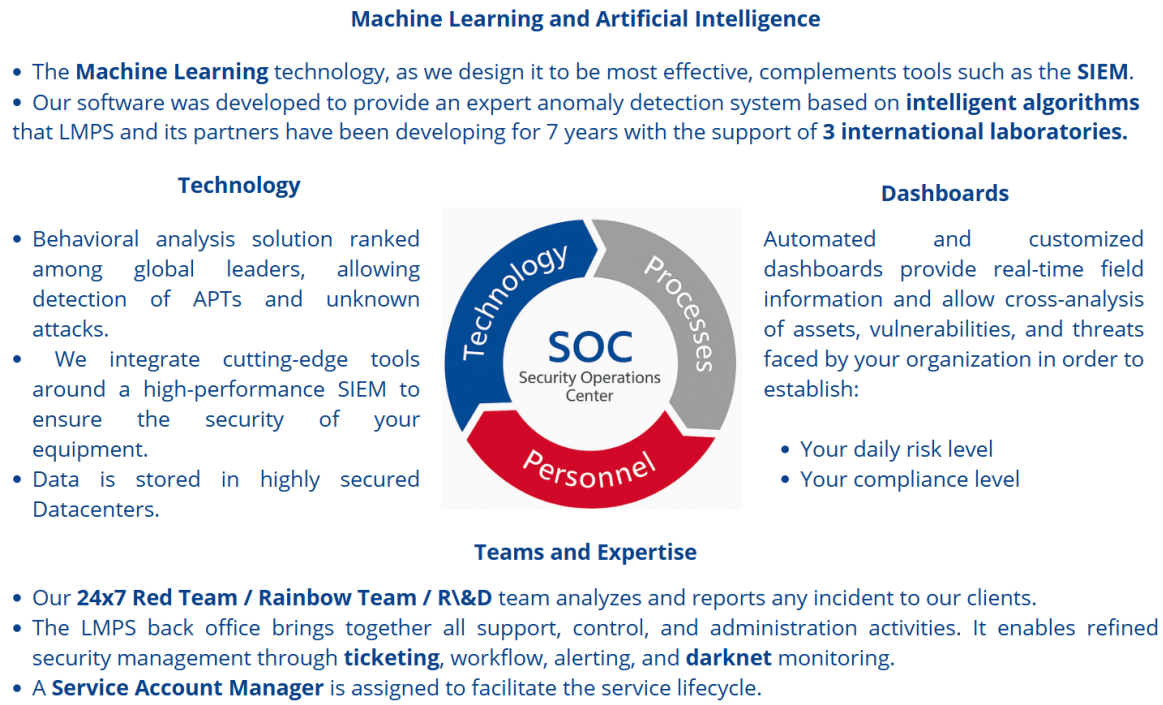

Figure 1.5: Overview of the CyberSOC of LMPS Group

The Cyber Academy Business Line at LMPS Group offers top-tier training programs in cybersecurity, enabling hundreds of engineers and apprentices each year to develop deep expertise and earn some of the most prestigious certifications in the industry. The training catalog includes courses in IT service management and project management, information systems security management, governance, risk and compliance, as well as cybersecurity and digital forensics.

In addition, **the Trust Technology** Business Line focuses on deploying information security systems and helping clients integrate technology solutions that enhance their risk management, data protection, and compliance environments.

LMPS Group delivers three main types of solutions:

- **Operational Security Solutions**
- **SCADA Security Solutions**
- **GRC Solutions for governance, risk, and compliance**

Figure 1.6: Overview of LMPS Group's Trust Technology Business Line

1.1.4 Company Teams

- **The Blue Team** is responsible for monitoring information security using the SOC to ensure tailored protection against cyber incidents. The main functions of this team include monitoring, detection, alerting, and reporting. It is composed of analysts and experts in handling security incidents, incident and crisis management procedures, and technologies such as *SIEM*, *THE HIVE*, *GRAYLOG* and *GRAFANA*.

- **The Red Team**, on the other hand, is responsible for conducting penetration tests to simulate the behavior of a malicious actor attempting to compromise the information system. It identifies vulnerabilities and prepares a report for the client, explaining the various vulnerabilities and the solutions to implement to fix them. It also performs technical audits such as source code audits, network and security equipment configuration audits, and system architecture audits.

- **The White Team** supports clients in compliance projects with information security standards and frameworks. It provides guidance for continuous protection by adopting a flexible, efficient, and defensible approach to data security. The main activities of this team include risk management, support for compliance with security standards and frameworks, compliance with cybersecurity laws and regulations, data classification, support in implementing security and business continuity management systems, and organizational and physical audits.

- **The Technology Team** aims to integrate various solutions to ensure security management, access control, malware protection, and other objectives. Among the solutions delivered by the team are:

- *PAM (Privileged Access Management)* : to authorize and monitor all privileged access to critical systems.
- *FIM (File Integrity Monitoring)* : to closely monitor sensitive files and trigger alerts in case of access, copying, downloading, or modification.

1.1.5 Red Team Missions

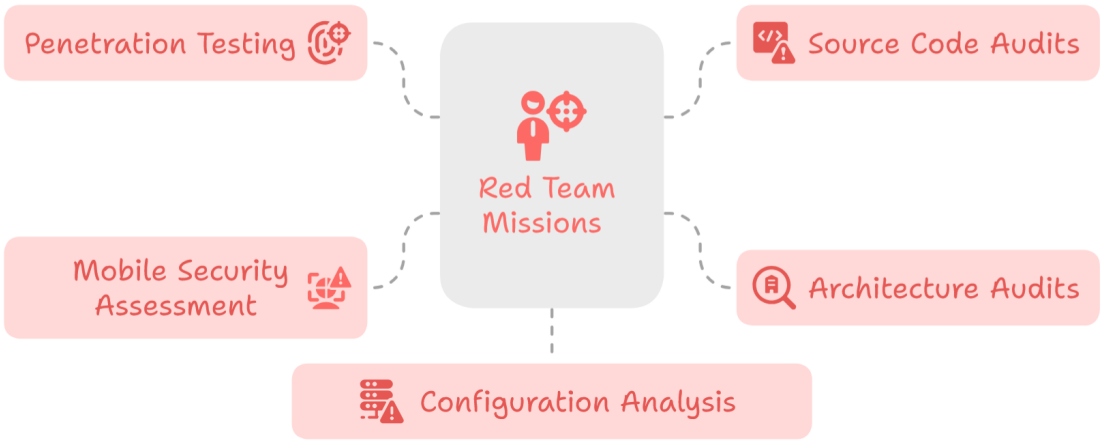

Figure 1.7: Red Team Missions

The Red Team of LMPS Group is dedicated to offensive security. Its role is to simulate real attack scenarios in order to test how well an organization can defend itself. These simulations help companies detect security weaknesses before malicious actors can take advantage of them.

The team provides different types of **penetration tests** (Black Box, Grey Box, and White Box), depending on how much access or information is given. These tests help find technical or logical flaws in networks, systems, or applications.

In addition to penetration testing, the team also performs:

- **Source code reviews**, which consist of analyzing the application's code line by line to detect security bugs, logic flaws, or dangerous coding practices (e.g., hardcoded credentials or lack of input validation).
- **Architecture audits**, where the team studies how the system is designed ,how components communicate, how data flows between services, and where potential security gaps might exist at the structural level.
- **Configuration reviews**, which focus on the setup of systems (e.g., servers, firewalls, Active Directory, etc.) to make sure they are hardened, properly isolated,

and not exposing unnecessary services or ports.

The Red Team also conducts **mobile security assessments**, where mobile apps are tested for vulnerabilities such as insecure data storage, poor encryption, or unsafe communication with external services.

Each mission results in a technical report that lists the discovered vulnerabilities, evaluates their level of risk, and proposes practical solutions to correct them. This approach helps clients improve their security posture and understand where they are most vulnerable.

1.2 General Context

Application whitelisting is a key layer in endpoint protection strategies, particularly in Windows environments. AppLocker, Microsoft's native application control feature, allows administrators to define policies that restrict the execution of unauthorized files based on parameters such as path, publisher, or file hash. Despite its potential, AppLocker is not immune to bypasses.

Over the years, multiple bypass techniques have been discovered, often leveraging LOLBAS, misconfigured permissions, writable directories, PowerShell abuse, or alternate data streams. These methods highlight gaps in default configurations and expose how AppLocker can be circumvented by adversaries.

This project focuses on analyzing these bypasses in depth and developing remediation rules to block them. The ultimate objective is to enhance AppLocker's effectiveness by implementing custom hardened policies, and to support this process through the creation of an automated PowerShell-based tool capable of scanning, applying, and reverting AppLocker configurations.

1.2.1 Problem Statement

Default AppLocker policies often leave gaps that can be exploited by attackers. Many organizations deploy AppLocker without adapting it to the evolving threat landscape or known bypass vectors. As a result, security is often overestimated, and attackers can still achieve code execution using well-documented techniques.

This project addresses the lack of visibility and remediation mechanisms around these bypasses by proposing a systematic approach to detect, block, and prevent them.

1.2.2 Proposed Solution

To address the limitations of AppLocker rules , this project introduces an automated PowerShell tool designed to strengthen application control on Windows. The tool was developed after researching and understanding the most common and well-documented AppLocker bypass techniques.

Its main goal is to apply defensive rules that prevent these bypasses and help secure the system in a practical and adaptable way. The tool offers three predefined profiles(rules):

Basic Profile – Recommended for standard users, applying essential protections without disrupting usability.

Hardened Profile – A stricter setup designed for administrators or high-privilege environments, blocking a wider range of bypass techniques.

Custom Profile – Allows users to select specific rule sets based on their needs.

The tool also supports scanning for AppLocker weaknesses, as well as backup, rollback, and policy enforcement mode selection ,providing both flexibility and control over the AppLocker configuration process.

1.2.3 Existing Solutions

Several open-source tools are available to assist with AppLocker configuration and deployment, but most remain limited in scope. AaronLocker provides policy templates and scripts to guide rule creation, but it lacks flexibility and does not adapt to specific attack scenarios. Applocker-Hardening automates the application of static policies, without offering any real analysis or bypass detection. AppLockerGen focuses on generating XML rules through a user-friendly interface but offers no support for hardening or automation.

The following table provides a benchmark comparison of the existing tools based on their ability to manage, harden, and automate AppLocker configurations.

	AaronLocker	Applocker-Hardening
Maintainer	**Microsoft**	**SimeonOnSecurity**
Type	PowerShell scripts	PowerShell scripts3
Hardened Rule Application	✓	✗
Scan for Weaknesses	✗	✗
Bypass Mitigation	✗	✗
Rule Customization	Manual	Manual
Rollback Support	✗	✗
Policy Export	✓	✓
Ease of Use	Medium	Low
Security Hardening Focus	Partial	Partial

Figure 1.8: Comparison of AppLocker Hardening Tools

These solutions are helpful in certain contexts, but none offer a complete framework that

automates detection, hardening, rollback, and customized policy application in response to real-world AppLocker bypasses. This project aims to fill that gap by providing a tool that brings all these capabilities together in a single, adaptable solution.

1.2.4 Project objectives

The main objective of this project is to improve the security of Windows systems by addressing the weaknesses of AppLocker. To achieve this, the project first focuses on understanding how AppLocker operates and how attackers are able to bypass its rules using well-known techniques. These bypasses are then reproduced and analyzed to gain practical insights into their mechanisms. Based on this knowledge, a set of custom hardening rules is designed to specifically mitigate the identified techniques. Finally, the project involves the development of a PowerShell-based automation tool capable of scanning the system for misconfigurations, applying secure policies, rolling back to previous configurations if needed, and generating detailed security reports.

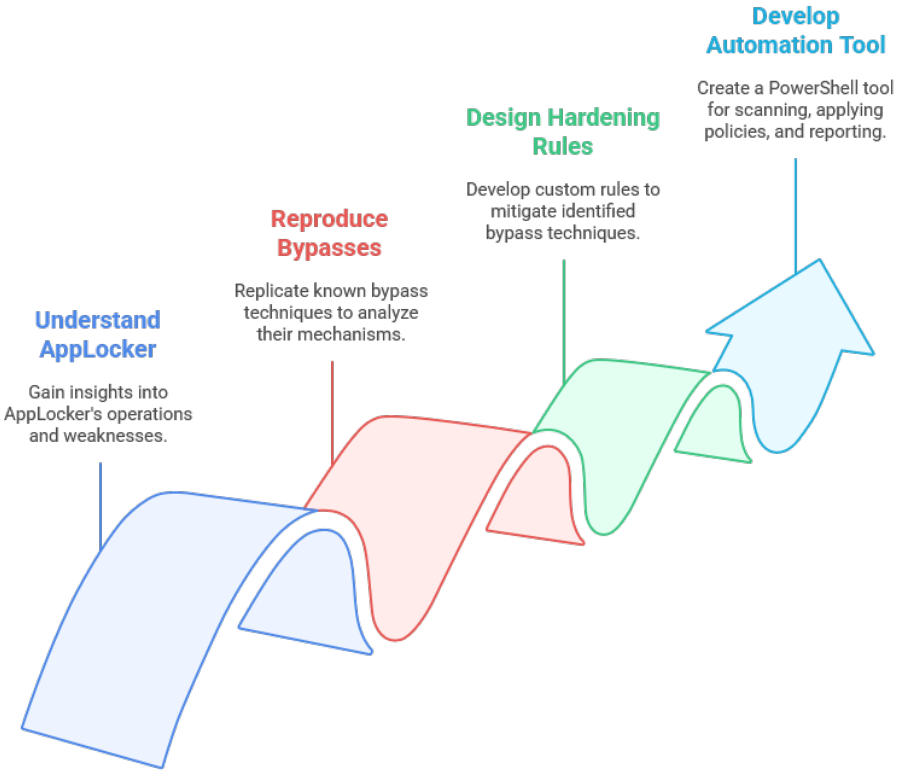

Figure 1.9: Project Objectives Line

1.2.5 Project Limitations

This project is limited in scope and focuses on specific environments and use cases. The main limitations are:

1. It targets only local Windows systems that support AppLocker, such as Windows 10/11 Enterprise and Windows Server editions.

2. It does not include enterprise-wide deployments using Group Policy or Active Directory-based AppLocker management.

3. It does not cover third-party application control solutions, such as Windows Defender Application Control (WDAC) or VMware Carbon Black.

4. It does not address kernel-level attacks, advanced persistence methods, or any techniques beyond the detection and enforcement capabilities of AppLocker.

1.3 Project management and planning

In order to effectively meet the project's objectives, careful planning of tasks and accurate time estimation were essential. Project planning plays a vital role in project management and is regarded as one of the most critical preparatory phases.

To facilitate this process and ensure clear progress tracking, we used the ClickUp platform to structure and manage the workflow. Before initiating the development, the project was broken down into specific tasks and subtasks. These are presented in Figure 1.10, along with their assigned start dates and estimated durations.

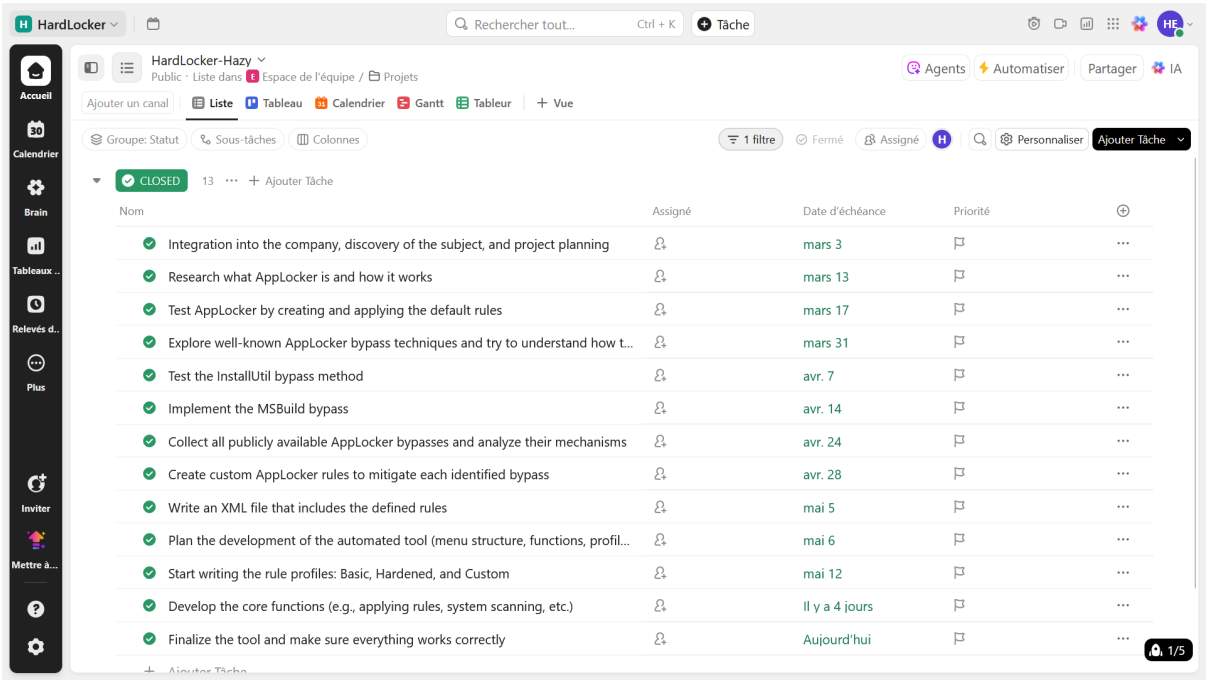

Figure 1.10: Project Task Scheduling With ClickUp

To complement the structured task management offered by ClickUp, We also used Obsidian as a personal knowledge and organization tool throughout the project. Obsidian allowed me to maintain a detailed digital lab notebook where We recorded daily progress, documented technical research, and drafted test results. Leveraging its Markdown-based system, We organized to-do lists, linked related notes, and scheduled tasks visually using calendar plugins. This setup helped me stay focused, maintain consistency in my work,

and ensure that no critical step or insight was overlooked during development.

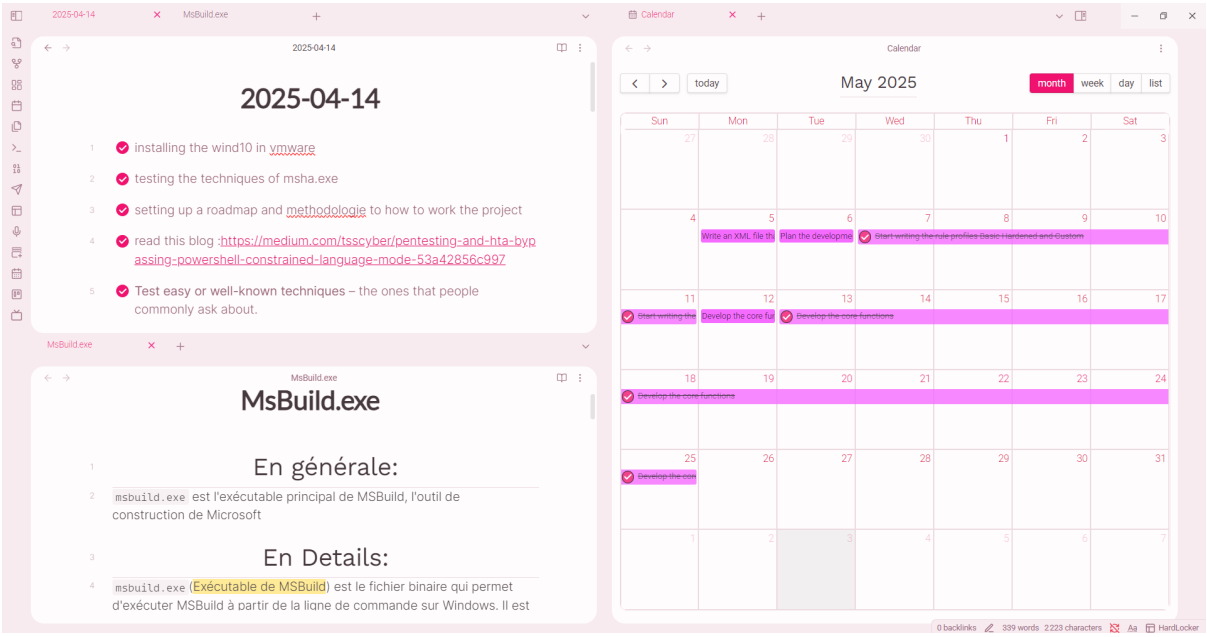

Figure 1.11: Task scheduling and note organization with Obsidian

1.3.1 Gantt Chart

A Gantt chart was used to break down the project into structured tasks and phases, allowing a visual overview of the entire timeline. It highlights task durations, start and end dates, and dependencies between activities, which made it easier to plan, track progress, and ensure timely completion.

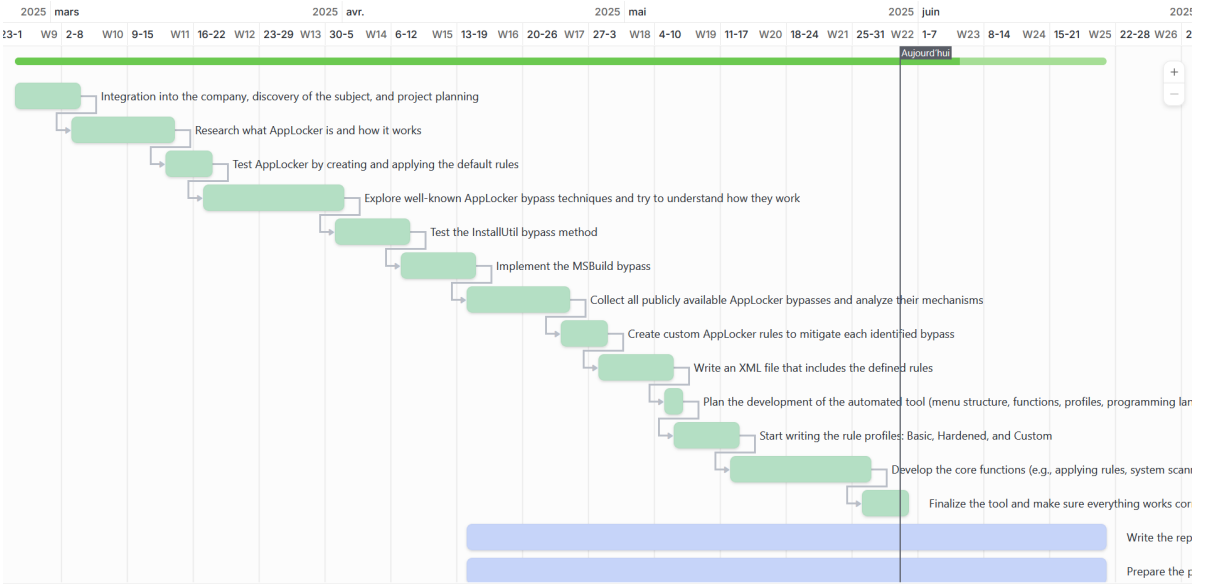

Figure 1.12: Gantt Chart of The Project

Conclusion

In summary, this chapter provided a global view of the project's background, including the presentation of LMPS Group and the motivation behind the work. It also outlined the goals, limitations, and planning approach adopted for the successful completion of the project.

Chapter 2

State of Art

Introduction

This chapter presents the theoretical foundations of the project by exploring the concept of application whitelisting and the role of AppLocker in Windows environments. It highlights the motivations behind the project, provides an overview of AppLocker's functionality, supported platforms, rule types, and enforcement modes, and reviews known bypass techniques and existing tools.

2.1 Genesis of The Project and Motivations

In today's increasingly complex digital environments, securing endpoints has become a critical concern for organizations aiming to defend against post-exploitation and lateral movement techniques. Among the native defenses available in the Windows operating system, AppLocker offers a built-in application whitelisting mechanism that allows administrators to control which applications and scripts are permitted to run. By defining execution policies based on file attributes such as path, publisher, or hash, AppLocker aims to reduce the risk of unauthorized code execution, often used by attackers to maintain persistence or escalate privileges.

However, despite its intended purpose, AppLocker is frequently misunderstood, misconfigured, or underutilized. Numerous real-world studies and offensive security research have demonstrated that AppLocker can be bypassed using widely available techniques. These include leveraging trusted system binaries (known as LOLBAS), executing payloads from writable directories, exploiting PowerShell or legacy components, and using memory-based injection techniques such as Reflective PE Injection. As a result, attackers can often evade AppLocker protections even when it is technically active.

This project was initiated in response to the growing gap between the theoretical capabilities of AppLocker and its practical effectiveness in real-world environments. While AppLocker is widely available on enterprise systems, its protections are often weakened by poor configuration and a lack of awareness of known bypass techniques. The increasing sophistication of attackers has made it clear that relying on default or minimally configured application control policies is no longer sufficient.

The motivation behind this work is to explore and understand the most common AppLocker bypass strategies, assess their impact, and contribute to the development of practical defenses. By studying these weaknesses and evaluating the limitations of existing solutions, this project seeks to promote a more security-conscious use of AppLocker and reinforce its role as a meaningful component in Windows endpoint protection.

2.2 Application Whitelisting

Application whitelisting is a preventive security mechanism that allows only explicitly authorized programs to run on a system, blocking the execution of all other software by default. This approach contrasts with traditional blacklisting methods, which attempt to block known malicious applications but allow anything not explicitly flagged. By adopting

a default-deny strategy, application whitelisting significantly reduces the risk of malware execution, unauthorized software installation, and abuse of trusted environments.

This technique is particularly effective in mitigating modern threats such as fileless malware, zero-day attacks, and the misuse of legitimate system tools. By allowing only approved software to execute, it prevents attackers from introducing unauthorized code—even if the code has not been previously identified as malicious. This makes whitelisting especially valuable in critical environments where system integrity and operational continuity are essential.

Whitelisting policies typically rely on various identification methods, including file path, file hash, and digital signatures. These attributes help ensure that only the correct versions of trusted applications are allowed to run, while unauthorized or tampered files are blocked. In managed environments, such as enterprise networks or critical infrastructure systems, application whitelisting provides strong control over the software ecosystem and limits user-initiated changes.

Beyond its role in threat prevention, application whitelisting also enhances visibility and control over system activity. It supports software inventory management and helps enforce compliance with organizational policies regarding approved software usage. However, implementing and maintaining an effective whitelist requires careful planning, ongoing monitoring, and the ability to adapt to legitimate software changes, such as updates or patches.

When deployed correctly, application whitelisting serves as a powerful line of defense, complementing traditional security solutions and providing a robust foundation for protecting endpoints against both known and emerging threats.

2.3 Introduction to AppLocker

2.3.1 Applocker Overview

AppLocker is a native Windows security feature that provides application control through rule-based enforcement. Its primary purpose is to restrict the execution of unauthorized applications, scripts, and installers based on predefined conditions. By allowing administrators to define which executables are permitted to run on a system, AppLocker helps limit exposure to potentially harmful or unauthorized software, thereby reinforcing endpoint security.

AppLocker operates through a series of rules that target specific file types, such as executable files, scripts, DLLs, Windows installers, and packaged applications. These rules can be based on several criteria, including file path, file hash, or digital signature (publisher). Once configured, AppLocker enforces these rules at runtime, either in audit or enforced mode, depending on the desired level of control.

This mechanism is particularly useful in organizational environments where maintaining control over the software ecosystem is essential for minimizing attack surfaces and

ensuring compliance with internal security policies.

2.3.2 Historical Background

AppLocker was introduced with Windows 7 and Windows Server 2008 R2 as an evolution of the older Software Restriction Policies (SRP). It was designed to offer more flexibility and granularity in application control. Unlike SRP, which was limited and less user-friendly, AppLocker introduced a modern rule management system and support for newer file types, such as packaged apps and scripts.

2.3.3 Use Cases in Enterprise Environments

In enterprise contexts, AppLocker serves as an essential component of a defense-in-depth strategy. It allows IT administrators to enforce software usage policies, limit execution to verified applications, and prevent employees from running unauthorized or potentially dangerous tools. This level of control is especially important in sectors with strict regulatory requirements or where operational continuity is critical.

AppLocker is commonly used in the following scenarios:

- **Securing workstations** by preventing the execution of scripts, macro payloads, or rogue binaries often used in phishing attacks.
- **Restricting lateral movement** in Active Directory environments by limiting the ability of attackers to execute post-exploitation tools.
- **Locking down shared or public-access systems**, such as kiosks or training machines, to a specific set of approved applications.
- **Facilitating compliance** with organizational or industry-specific regulations by enforcing strict software governance.

Additionally, when integrated with Group Policy, AppLocker can be deployed across large environments and tailored to different user groups or organizational units, making it a flexible and scalable solution.

2.4 Supported Operating Systems

AppLocker is a Windows feature whose availability depends on the edition and version of the operating system. It was first introduced with Windows 7 and continues to be supported in more recent versions like Windows 10 and Windows 11. However, its functionality particularly the ability to create and enforce rules varies across editions.

In general, AppLocker is fully supported in Enterprise and Education editions, where both the creation and enforcement of rules are available and manageable via Group Policy. On Professional editions, AppLocker can often be used to create policies locally, but enforcement capabilities may be limited or unsupported in domain environments.

Home, Starter, Core, and other consumer-targeted editions lack full AppLocker support. The following table summarizes AppLocker compatibility across major Windows versions:

Table 2.1: AppLocker availability on Windows 7

Starter	**Home Basic**	**Home Premium**	**Professional**	**Enterprise**	**Ultimate**
No	No	No	Create policies, but cannot enforce	Create and enforce policies	Create and enforce policies

Table 2.2: AppLocker availability on Windows 8

RT	**(Core)**	**Pro**	**Enterprise**
No	No	No	Yes

Table 2.3: AppLocker availability on Windows 10

Home	**Pro**	**Enterprise**	**Education**
Yes	Yes	Yes	Yes

2.4.1 Functional Limitations in Some Editions

While some editions such as Windows 10 Pro include AppLocker components, they offer limited functionality compared to Enterprise editions. Specifically, rule enforcement through Group Policy is not fully supported or reliable in standalone environments.

In contrast, Enterprise and Education editions offer complete integration, including:

- Centralized rule deployment via Group Policy Objects (GPO)
- Enforcement of multiple rule types (Executable, Script, DLL, MSI, Packaged apps)
- Audit and Enforcement modes for testing and production environments

For these reasons, systems intended for full-scale AppLocker deployment and management should run Enterprise or Education editions of Windows.

2.5 AppLocker Rule Types

2.5.1 Rule Types and Associated File Association

AppLocker supports different categories of rules, each designed to control specific types of executable content. These rule types allow administrators to define precise controls over which files are allowed or denied execution based on attributes such as path, publisher, or file hash. AppLocker operates on the following rule types:

- **Executable rules**: Refers to .exe and .com files that are launched from the local computer, removable media, or from a network location (file share, web server etc.). It also includes applications that are delivered by application virtualization (App-V). This is the file type most commonly controlled as they provide the greatest opportunity for computer compromise

- **Script rules**: Refers to PowerShell (.ps1), Windows command interpreter (.cmd and .bat), VB script (.vbs) and Java script (.js) files, but does not apply to client-side scripts executed by a browser. Note that additional script types (such as Perl) cannot be controlled by AppLocker, although the scripts themselves can be prevented from running by preventing the corresponding script interpreter from running. Also note that Office applications can run Visual Basic for Applications (VBA) code in macros that are not controlled by AppLocker.Unlike almost all executables, installers, packaged apps and DLL/OCX files, many of the several hundred scripts that ship with Windows are not digitally signed. A number of complex path or file hash rules will need to be created to allow them to run. While secondary to controlling executable files, execution of scripts should be considered for high value computers, or those exposed to significant risk. If the organization has enabled code signing for in-house developed applications and packages, the same process can be used for digital signing of scripts to allow simpler (and fewer) publisher rules to be used to allow them to run.

- **MSI installer rules**: Refers to traditional application packages that are installed by the Windows Installer service. It includes the installer packages themselves (.msi), and application patches (.msp). It also includes application transforms (.mst) that specify which components of the installer package should be installed.Windows installers install files and make configuration changes that only users with administrative rights can perform. Assuming that users are not given administrative rights, a good level of control already exists in preventing users from making unauthorized changes. Even after an application is installed, Executable rules must still be in place to allow the application to actually execute. For these two reasons, many organizations choose not to control Windows Installers using AppLocker. Alternatively, they may deem to implement a slightly restrictive policy allowing administrators to install applications as long as they have been signed by a publisher that the computer trusts.

- **DLL rules**: Refers to .dll and .ocx files that are shared between applications and loaded by the applications that require them.

Modern malware targets the shared code that legitimate applications load in an effort to avoid detection. To provide the best possible protection, DLL/OCX files should be evaluated by AppLocker policy. This however this creates an additional dependency in that AppLocker rules must be created that allow every such .dll and .ocx file required for the application to function. If a single .dll or .ocx file that is required by an application is prevented from loading, the application will either fail to run or will generate errors when certain tasks are performed. Consequently, checking of .dll and .ocx files is disabled by default and must be explicitly enabled before it can be used.While organizations must perform rigorous testing to ensure that they include all shared libraries used by their applications, the procedures in this guide should allow this feature to be enabled and used to provide maximum protection against malicious code. However depending on resource availability, some organizations may choose to initially implement AppLocker without enabling this feature, and enable it once they are comfortable in deploying and maintaining it in Production

- **Packaged app rules**: These target Universal Windows Platform (UWP) applications, also known as Microsoft Store apps. They ensure that only authorized modern applications can run within the Windows environment.

Rule Type	**File Associations**
Executable	.exe, .com
Windows Installer	.msi, .msp, .mst
Script	.ps1, .bat, .cmd, .vbs, .js
Packaged App	.appx
Shared Libraries & Controls	.dll, .ocx

Table 2.4: AppLocker Rule Types and File Associations

By using these rule types, organizations can implement fine-grained application control tailored to their security requirements, reducing the risk of unauthorized software execution across various vectors.

2.6 Applocker Rule Objects

AppLocker identifies applications using three primary methods, each corresponding to a different rule type:

- **Path rules** : These rules control application execution based on the file's location in the system. They are easy to configure and commonly used. By default, AppLocker

allows execution from secure directories like C:and C:Files, which are essential for system functionality. However, path-based rules can be risky if users have write access to the allowed paths, as this could enable the placement and execution of unauthorized files.

- **Publisher rules** : These rely on the digital signature embedded in the application by the software vendor. They are particularly useful for managing updates, as they can apply broadly to all versions of a signed application from a trusted publisher, reducing maintenance efforts.

- **File Hash rules** : These use a cryptographic hash to uniquely identify a file. This approach offers the highest level of security, as it ensures that only the exact version of a file is allowed to execute. However, it requires frequent updates whenever files are modified or patched.

Each rule type has its own advantages and disadvantages, which are described in detail below. A summary table is included at the end of the section.

Path Rules

Path rules apply to a specific file or to a folder. If defined for an individual file, the rule includes the full file path. For folders, the rule applies to all files with applicable extensions inside the folder **and its subfolders**.

While absolute paths can be used, it is also possible to use predefined AppLocker variables to account for system-specific folder names or drive letters. However, using absolute paths is generally more secure, as it reduces the risk of attackers exploiting variable path resolution.

Below is a table listing commonly used AppLocker path variables and their Windows environment variable equivalents :

Windows Directory or Drive	AppLocker Path Variable	Windows Environment Variable
`\Windows`	`%WINDIR%`	`%SystemRoot%`
`\System32`	`%SYSTEM32%`	`%SystemDirectory%`
Windows installation directory	`%OSDRIVE%`	`%SystemDrive%`
Program Files	`%PROGRAMFILES%`	`%ProgramFiles%`, `%ProgramFiles(x86)%`
Removable media	`%REMOVABLE%`	–
Removable storage (USB)	`%HOT%`	–

Table 2.5: AppLocker Path Variables

Path Rule Advantages

The advantage of path rules is that they can accommodate future software changes on the target computer far better than the other types – which is why they are used in the creation of default rules for EXE, scripts and DLL / OCX. For example, if you create a path rule for `%PROGRAMFILES%`, any future additions to `\Program Files`, `\Program Files (x86)` or any of their subfolders will be automatically accommodated without changes to the rule.

Path Rule Disadvantages

Path rules however have serious disadvantages compared to the other two types. Most obviously, it does not align well with the policy intentions of application whitelisting. Those defining AppLocker policy may not care where a particular file is run from – they would want the same policy applied to a file regardless of which folder it resides in or whether it is local, on a file share, USB drive or DVD.

The way in which AppLocker treats subfolders of path rules is another serious disadvantage. If AppLocker only cares that a file is run from a particular location, then a user can copy any file to a writable subfolder under the parent path defined in the rule to circumvent policy. The default EXE, script and DLL / OCX rules (which will not be used in this guide) allow anyone to run anything if it resides in a subfolder of `\Windows`. As there are numerous standard user-writable subfolders of `\Windows`, these rules need to have numerous exceptions (described below) defined if they are to be effective. The number of writable subfolders increases further when you consider accounts commonly used by services – such as Local Service and Network Service. Any service running as these accounts that could be made to write content to one of these locations could be used to circumvent a path rule.

As described above, path rules can be defined down to individual executables (by full path and name). However AppLocker has no means of determining whether the file name actually represents the file intended to be allowed. This means that any executable can be renamed to match the conditions of a path rule and will be allowed to run – whether it is the legitimate executable or something completely different.

The final disadvantage is that path rules do not check the integrity of files before allowing them to run. This means that a file that is allowed to run via an AppLocker rule will run equally well if it is the intended version or has been modified for malicious purposes (e.g. by malware).

Due to these numerous, significant disadvantages, path rules should only be used when neither of the two alternative rule types are appropriate. When using path rules, care must be taken to ensure that ACLs on the respective folders (and all subfolders) prevent users from writing content to them.

File Hash Rules

ach file hash rule contains the calculated hashes of one or more files. Before a file is run, its hash is calculated by Windows, and if it matches a hash in any of the rules, the corresponding rule action is applied (allow or deny).

File Hash Rule Advantages

File hash rules avoid the location dependence and integrity issues of path rules. You cannot simply copy a file to a different location or rename a file to circumvent policy as you can with path rules, as the file's hash value is used to uniquely identify it. You also cannot change a single bit within a file whose hash is listed in a hash rule, as the calculated hash will be completely different from the one listed. The one-way nature of hash algorithms makes it infeasible to try to change or create a file that will hash to a specific hash value and still perform its intended function. This gives a very high degree of assurance that the file the user is trying to run is the one listed in the hash rule, and the appropriate action will be applied.

File Hash Rule Disadvantages

While providing the highest assurance of all rule types, hash rules are the least adaptable to change and hence the most expensive to maintain. Every time that a file is up graded or patched, the hash rule must be modified to include the new hash value or the file will be prevented from running. This would require a disciplined change management process, and result in significant administrative overhead. Consider also that up grades/patches may be applied over a period of time, and you may have numerous versions of any given executable in use across an organization – of which the hashes of all must be included in hash rules to allow them to run. The other key disadvantage of hash rules is the sheer number of rules that need to be created, maintained and processed by the computers implementing them. While a single AppLocker rule can contain many hashes, each file needs to be uniquely identified by its hash value. There can be ma ny thousands of

eligible files on a computer, making the task of managing thousands of hash values a difficult one.

While security - conscious companies may accept the increased burden for the very tight control given by hash rules, the general recommendat ion is to avoid hash rules except in very specific circumstances – primarily where the files have not been digitally signed. Files that have been digitally signed should use Publisher rules (described next) which provide the same integrity benefits as hash rules but with far less management overhead.

Publisher Rules

When a file is digitally signed, a hash of the file contents and file attributes (such as the product name and its version) is cryptographically protected (signed) by a publisher. Assuming that publisher is trusted by the user's computer, the computer can validate the integrity of the file and its attributes. Similar to the mechanism used in evaluating hash rules, if the contents or attributes of a signed file is changed, it won't match the information in the signature and will fail validation.

AppLocker uses this validation mechanism in Publisher rules. Rather than matching the hash of the file to be run, Publisher rules make decisions based on any combination of:

- Publisher name (or at its most permissive, any publisher that is trusted by the computer).
- Product name.
- File name (as originally chosen by the publisher, not the current name of the file if it has been changed).
- File version – exact version, specified version and later or specified version and earlier.

Publisher Rule Advantages

Publisher rules provide a similar level of integrity protection as file hash rules, without the management overhead. If a software vendor patches or updates their product, they typically sign the updated files with the same attributes as the previous versions – publisher, product and file name. File version would typically be incremented, but this can be accommodated in a publisher rule by the condition of "this version and above" (or you can use to match any file version). In addition to the obvious security benefits, publisher rules can be used as a way of forcefully retiring obsolete versions that an organization no longer wishes to support, or are known to contain vulnerabilities.

Publisher Rule Disadvantages

The obvious downside to publisher rules is that they can only be used to control applications that have been digitally signed. To control unsigned files, file path or hash rules are the only alternative. The good news is that the majority of software vendors sign

their code to ensure its authenticity and to avoid the negative user experience created by operating system warnings that the "publisher cannot be verified". For organizations that develop their own applications, it is a straightforward process to digitally sign their files to allow publisher rules to be able to control execution. When choosing publisher rules, you need to keep in mind that they are only as secure as the publisher is trustworthy. If you rely on publisher rules and one of the specified publishers has their code signing certificate private key compromised, whoever compromised that certificate could use it to sign their own code . AppLocker would not be able to distinguish between this and legitimate files, and could potentially be used to bypass it. There have also been a number of high profile incidents where trusted certificate issuers have been compromised and fraudulent certificates generated . Fraudulent code signing certificates (that mimic legitimate publisher certificates) could be used to sign malicious code that would be trusted by AppLocker, potentially subverting AppLocker policy. Organizations must make an assessment as to which publishers (and all certificate authorities in the publisher's trust chain) they choose to trust. If they cannot trust a publisher or any of the CAs in the chain, they must use file path or hash rules for the application in question.

The following table summarizes the advantages and disadvantages of the three different rule types. The assessment is qualitative, and the meanings of the assessed items are:

- **Flexibility** – the ability for existing rules to accommodate routine changes such as patching or upgrading to a later version.
- **Location independent** – rules continue to be enforced regardless of where a user tries to execute it from.
- **Object assurance** – the ability to accurately match an object and prevent changes to its properties (such as file name) from avoiding a match.
- **Object integrity** – rules protect the integrity of the objects they target. For example, preventing an infected file from running where a rule allows the original file to run.
- **Ease of use** – rules are easy to create and maintain and are intuitive as to their purpose.

	Flexibility (future proof)	**Location** Independent	**Object** Assurance	**Object** Integrity	**Ease of Use**
Path	High	Low	Low	Low	High
File hash	Low	High	High	High	Low
Publisher	High (Note 1)	High	High (Note 2)	High	Medium (Note 3)

Table 2.6: Assessment of AppLocker rule types

Note 1: assumes that the software vendor adopts a consistent naming and numbering scheme for the files they sign.
Note 2: assumes that the publisher takes adequate steps to protect their signing certificate, and depends on the trustworthiness of all CAs in the chain that issued it.
Note 3: slightly more complex due to the number of attributes that can be used in object definitions, but this complexity gives a great deal of flexibility

2.7 AppLocker Operating Modes

AppLocker operates in one of three modes for each of the rule types listed above:

Not Configured	If no rules are defined, AppLocker is disabled for that rule type. If one or more rules exist, AppLocker operates in "Enforce rules" mode. If a higher-precedence GPO is set to "Not Configured", the system defers to the next applicable GPO in the processing order.
Enforced	AppLocker enforces the specified rules strictly. Any unauthorized actions are blocked, and corresponding events are logged to the Windows Event Log.
Audit Only	Actions are not blocked, but logged as if the rules were enforced—providing visibility into what would have been allowed or denied. This mode is commonly used to test AppLocker configurations before enforcing them in production.

Table 2.7: AppLocker Operating Modes

Audit Mode and Behavior Considerations

The **Audit only** mode allows administrators to test AppLocker rules without disrupting normal system operations. In this mode, applications are not blocked from execution; however, events are logged to indicate whether an action would have been allowed or denied if enforcement were active. This provides a safe environment for evaluating and fine-tuning rule configurations.

Systems typically remain in audit mode until administrators are confident that the rules accommodate all legitimate usage scenarios. Once the rules are validated, the system can be switched to **Enforce rules** mode, in which any action not explicitly permitted is blocked. Even in enforcement mode, all decisions, whether allowing or blocking, continue to be recorded in the Windows Event Log for ongoing monitoring and refinement.

- Selecting *Enforce rules* or *Audit only* has no effect unless at least one rule exists for that rule type.
- If a rule is created while the mode is set to *Not Configured*, the system defaults to *Enforce rules* for that type. To prevent accidental enforcement, it's recommended to explicitly choose *Audit only* before defining rules.

- Enforcement settings defined via Group Policy Objects (GPOs) override local policy settings. When multiple GPOs apply to a system, the one with the highest precedence (applied last) determines the mode.

- If *Enforce rules* is enabled for EXE files and at least one rule exists, the system may also enforce rules on APPX packages—even if no APPX-specific rules are defined. This can prevent APPX apps from running until explicit rules for them are added.

2.8 Known AppLocker Bypass Techniques

2.8.1 LOLBAS Abuse

Living Off The Land Binaries, Scripts and Libraries refer to legitimate, pre-installed Windows utilities that can be repurposed by attackers to execute malicious payloads while avoiding detection. Since these tools are signed by Microsoft and often whitelisted by default, they provide an effective method for bypassing application control mechanisms such as AppLocker. Attackers frequently exploit binaries like mshta.exe, msbuild.exe, regsvr32.exe, and cmstp.exe to run unauthorized code without triggering security alerts. This technique allows adversaries to operate covertly using trusted system components.

2.8.2 Writable Paths and Misconfigurations

AppLocker enforces policies based on file locations, but its effectiveness can be undermined by improper directory permissions. Many paths, including `%TEMP%`, `%APPDATA%`, and `C:\Windows\Tasks\`, are writable by non-privileged users. Attackers take advantage of these directories to drop and execute malicious files within otherwise trusted locations. Without additional rules to explicitly block execution from these paths, even well-configured AppLocker environments can be bypassed. Securing these locations is therefore a critical step in mitigating exploitation risks.

2.8.3 Alternate Data Streams (ADS)

Alternate Data Streams (ADS) are a feature of the NTFS file system that allows data to be stored in a hidden stream attached to a file. Attackers exploit ADS to hide malicious executables within seemingly innocuous files, allowing execution that can evade AppLocker's default rules. For instance, a payload might be written to file.txt:evil.exe and executed without being directly visible in file explorers. Since AppLocker does not natively inspect ADS, specific path rules must be configured to block these hidden streams effectively.

2.8.4 PowerShell v2 and Script Execution

PowerShell Version 2 presents a significant security risk in modern environments due to its lack of support for key protections such as Constrained Language Mode and advanced logging features. Attackers often target systems where PowerShell v2 is enabled, using

it to execute malicious scripts without triggering AppLocker restrictions. Because AppLocker cannot effectively control script execution in this legacy version, organizations are advised to disable PowerShell v2 entirely and enforce script execution rules on modern versions with signed code requirements.

2.8.5 Reflective PE Injection (Fileless Execution)

Reflective PE Injection is a technique used to load and execute portable executable (PE) files directly in memory without writing them to disk. This approach allows attackers to completely bypass AppLocker, which monitors disk-based execution. Tools like PowerSploit's Invoke-ReflectivePEInjection script enable adversaries to store malicious binaries in memory and invoke them programmatically, leaving no artifacts behind. Since this method avoids traditional file-based triggers, it represents a powerful bypass against AppLocker and traditional antivirus mechanisms.

2.8.6 Office Add-ins and VSTO Exploitation

Attackers can exploit Visual Studio Tools for Office (VSTO) add-ins to execute malicious code when opening applications like Word or Excel. These add-ins can run without administrator privileges and are not blocked by AppLocker by default. This makes them a stealthy method for persistence and code execution within trusted programs.

2.8.7 DLL Hijacking

DLL hijacking is a technique in which an attacker exploits the way a legitimate application loads dynamic link libraries (DLLs). Many Windows applications dynamically load DLLs from specific locations, often using relative paths. If the application does not explicitly specify a full path or validate the origin of the DLL, Windows will search for it in a predefined order , including directories that may be writable by users.

An attacker can take advantage of this by placing a malicious DLL with the same name as a legitimate one in a location that is searched before the correct directory. When the application is executed, it unknowingly loads the attacker's malicious DLL instead of the intended one. This allows arbitrary code execution within the context of a trusted, often whitelisted, executable.

2.8.8 Exploitation of User-Writable System Files

In some cases, AppLocker can be bypassed through exploitation of user-writable files located within trusted system directories. A notable example involves three files found in `C:\Windows\System32\AppLocker` : AppCache.dat, AppCache.dat.LOG1, and AppCache.dat.LOG2. These files are writable by the first user who logs into the system after AppLocker has been deployed. Although they are not executables themselves, their write permissions can be abused as part of a broader evasion strategy, allowing an attacker to plant or manipulate data in a location that is typically trusted by default AppLocker rules. This scenario highlights a subtle but impactful misconfiguration, where trusted paths intersect with insecure permissions.

2.9 Technologies, Tools, and Environments Used

PowerShell is a powerful command-line shell and scripting language developed by Microsoft for task automation and configuration management.
It was used as the primary language to develop the HardLocker tool, allowing for automation of AppLocker rule creation, XML generation, and interactive CLI interface.

Visual Studio Code is a lightweight yet powerful source code editor developed by Microsoft, offering support for debugging, version control, and extensions.
It served as the development environment for editing, organizing, and testing the PowerShell scripts that make up the core of the HardLocker project.

VMware Workstation is a virtualization platform that allows running multiple operating systems on a single physical machine.
It was used to build isolated Windows environments for testing the effectiveness and compatibility of AppLocker rules without affecting the host system.

LOLBAS (Living Off the Land Binaries And Scripts) refers to legitimate Windows binaries that can be misused for malicious actions. They were analyzed during the research phase to understand common bypass techniques, and HardLocker was designed specifically to mitigate these threats through tailored AppLocker rules.

XML (Extensible Markup Language) is a markup format used to encode structured data in a way that is both human-readable and machine-readable.
It was used to define AppLocker rule sets that were generated dynamically and applied automatically by the script to enforce consistent system policies.

AppLocker is a Windows feature that restricts the execution of unauthorized executables, scripts, and installers, enhancing application control and reducing the attack surface.
In this project, AppLocker was the main target of the HardLocker tool `secpol.msc`.

Conclusion

To conclude, this chapter provided a comprehensive overview of AppLocker, its strengths, limitations, and common evasion techniques. It also introduced the tools and technologies

used during the project. This theoretical background sets the stage for the practical implementation and hardening efforts discussed in the following chapters.

Chapter 3

Case Studies of AppLocker Evasion Methods

3.1 Introduction

This chapter presents a concrete and practical extension to the theoretical aspects previously discussed regarding AppLocker. Having examined the design, configuration mechanisms, and theoretical protections provided by application whitelisting, we now shift our focus to demonstrating how these controls can be bypassed in real-world scenarios. The techniques explored here were not selected arbitrarily ,they are the result of empirical testing conducted as part of this project's practical phase.

Through two concrete attack simulations leveraging InstallUtil.exe and MSBuild.exe, we illustrate how an adversary can exploit legitimate, trusted Windows components to execute malicious payloads, even in environments where AppLocker is fully enforced. These binaries, which are typically whitelisted due to their Microsoft signatures and legitimate administrative functions, serve as ideal vectors for stealthy code execution. The objective of this chapter is to document these bypass strategies in a structured, academic format, evaluate their feasibility in constrained environments, and highlight the real risks they pose to endpoint defenses. This analysis will set the stage for the following chapter, in which we introduce and evaluate a custom mitigation strategy developed to counter such attacks.

3.1.1 Reasoning Behind Technique Selection

The choice of InstallUtil.exe and MSBuild.exe as case studies in this chapter is grounded in their prominence within the security community as two of the most widely documented and consistently validated AppLocker bypass techniques. Both binaries are part of the standard Microsoft .NET Framework installation and are present by default on most enterprise Windows environments, including Windows 10. Their legitimate purpose, combined with Microsoft signing, often grants them implicit trust within endpoint protection solutions and application whitelisting policies.

What makes these tools particularly suitable for evasion is their ability to execute user-supplied C code at runtime. This allows attackers to craft payloads that run entirely in memory, significantly reducing artifacts on disk and evading many traditional detection methods. Their native capability to execute complex logic without invoking external scripts or binaries enables them to function as highly effective Living Off The Land Binaries (LOLBins). As such, InstallUtil.exe and MSBuild.exe represent ideal vectors for demonstrating realistic, impactful AppLocker bypasses that remain relevant in contemporary security assessments.

3.2 Case Study 1: Bypassing AppLocker Using InstallUtil.exe

InstallUtil.exe is a legitimate utility that comes bundled with the Microsoft .NET Framework and is primarily used for installing and uninstalling Windows Services. Its typical use involves executing methods defined in installer classes of .NET assemblies. However, this functionality can be weaponized by attackers. In this scenario, the binary is exploited

to execute arbitrary code embedded in a disguised service installer, allowing for stealthy execution even in environments where AppLocker is actively enforcing policy restrictions.

The bypass process begins with the development of a Csharp source file that defines a seemingly innocuous program. The class includes a Main() function, which appears benign and is never executed when InstallUtil is used. The real payload is hidden within an overridden Uninstall() method. When InstallUtil.exe is invoked with the /U switch, it triggers this method, allowing the embedded malicious logic to execute. This logic leverages Windows API functions such as VirtualAlloc, Marshal.Copy, and CreateThread to inject shellcode into memory and execute it in a new thread, thus avoiding detection and disk I/O. The shellcode is generated using msfvenom with a typical payload like windows/meterpreter/reverse_tcp, encoded in Csharp, and embedded directly in the source code.

After the code is written, it is compiled into an executable using the .NET Csharp compiler csc.exe. For maximum compatibility and stealth, the compilation targets the x86 platform and enables unsafe code execution. Once the executable is ready, it is delivered to the victim system,usually via USB, file share, or browser download—and executed using the command:

```
C:\Windows\Microsoft.NET\Framework\v2.0.50727\InstallUtil.exe /logfile= /LogToConsole=
    false /U exeshell.exe
```

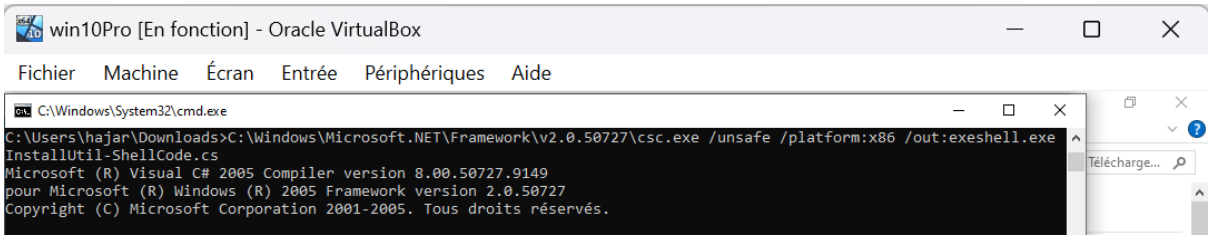

Figure 3.1: Compilation of the source code

This execution chain circumvents AppLocker restrictions because InstallUtil.exe is signed by Microsoft and is considered a trusted binary in most configurations. No graphical window is displayed, and log generation is suppressed by omitting a log path and disabling console output. As a result, a reverse shell is silently spawned, providing the attacker with full control over the compromised system.

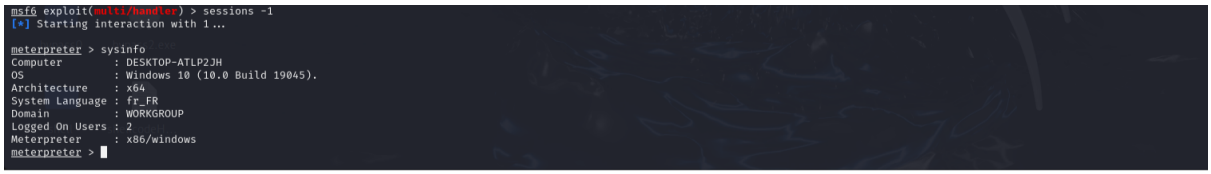

Figure 3.2: Reverse shell successfully established

3.2.1 Case Study 2: Bypassing AppLocker Using MSBuild.exe

MSBuild.exe is another highly trusted binary provided by Microsoft and used internally for compiling .NET applications. It is present on all Windows systems that support Visual Studio or the .NET Framework. The danger of this utility arises from its ability to interpret specially crafted XML-based project files that can include and execute arbitrary C# code using a mechanism called *CodeTaskFactory*. While designed to simplify developer tasks, this feature allows attackers to execute memory-resident payloads without writing executables to disk.

The attack begins by generating a Meterpreter reverse shell payload using `msfvenom` in C# format. This shellcode is then embedded inside a malicious MSBuild project file—an `.xml` file conforming to the standard schema for MSBuild. Within this file, a custom `<UsingTask>` node is used to declare a new task that includes raw C# code wrapped in `<![CDATA[ ... ]]>`. This block defines a class that inherits from the MSBuild `Task` class and overrides the `Execute()` method. When the project file is executed using `MSBuild.exe`, the payload is parsed, compiled, and run directly in memory.

To deploy the attack, the XML file is transferred to the target system, either manually or using a PowerShell command such as:

```
Invoke-WebRequest http://attacker/payload.xml -OutFile payload.xml
```

Then, we executes it with:

```
C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe C:\Users\Public\
    msbuild_shellcode.xml
```

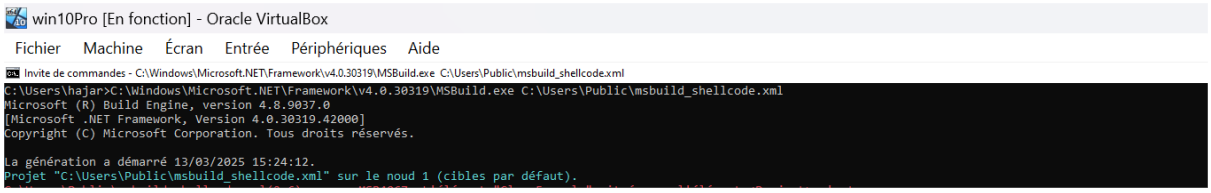

Figure 3.3: Exploit execution using MSBuild

This command triggers the parsing of the project file, compiles the embedded code, and executes it in memory. The result is a fully functional reverse shell back to the attacker, all without writing any executables to disk or invoking untrusted tools. AppLocker fails to block this execution because the binary itself (MSBuild.exe) is Microsoft-signed, and the payload resides within a format typically used by developers, not malware analysts.

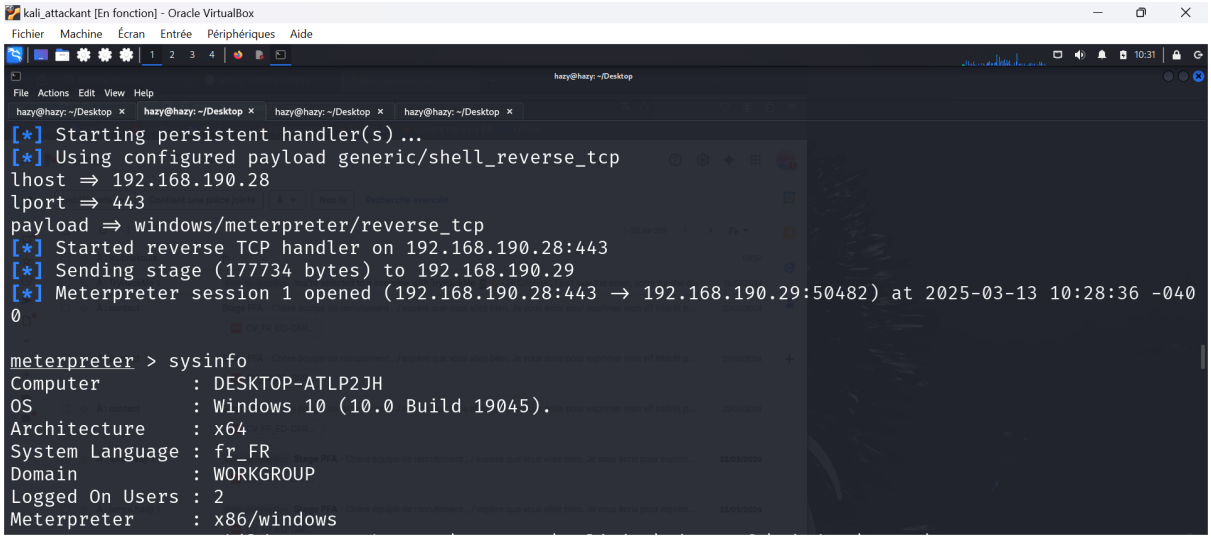

Figure 3.4: Reverse shell establishment

Conclusion

The two attack scenarios presented in this chapter reveal a profound limitation of AppLocker: its dependency on file paths, publishers, and hash-based whitelisting can be subverted by abusing trusted system binaries. These binaries-InstallUtil.exe and MSBuild.exe-are integral parts of the Windows ecosystem, signed by Microsoft, and often excluded from strict enforcement. Attackers exploit this trust to execute arbitrary code in memory, bypassing most defenses and evading forensic detection. These techniques highlight how application whitelisting, while effective in theory, can be circumvented when security policies are not configured with a least-privilege mindset and contextual awareness.

This realization calls for a more comprehensive approach to application control. Organizations must supplement AppLocker with additional layers of defense, such as explicit deny rules for high-risk LOLBins, system-wide monitoring of child process execution, and integration with Windows Defender Application Control (WDAC). In response to these shortcomings, we propose a custom hardening solution based on AppLocker that goes beyond default rulesets. This solution includes automatic rule generation for dangerous binaries, blocking execution from writable directories, and enhanced monitoring. The technical design and implementation of this mitigation strategy will be presented in detail in the next chapter.

Chapter 4

Implementation

Introduction

In this chapter, we present the practical realization of the HardLocker tool, designed to automate the hardening of Windows systems through AppLocker. The implementation is based on a detailed analysis of known bypass techniques and common misconfigurations observed in enterprise environments.

We describe the tool's internal structure, main functionalities, and user interface elements, supported by illustrative screenshots and practical usage examples.

4.1 Design Choices and Testing Environment

HardLocker was developed entirely in PowerShell to ensure seamless integration with native Windows systems, avoid external dependencies, and simplify administrative usage.

All testing and validation were carried out on a Windows 10 Professional virtual machine, a deliberate choice for both compatibility and realism. Windows 10 remains widely used in production environments and provides an ideal platform to study known AppLocker bypass techniques. Its balance of flexibility and legacy support made it preferable for experimentation and research purposes compared to Windows 11, which enforces stricter security policies that may hinder low-level testing.

4.2 General Architecture

The architecture of the HardLocker tool is designed in a simple and modular way to make it easier to manage, understand, and improve in the future. The tool was fully developed in PowerShell and works directly with the AppLocker system using built-in Windows commands.

It is divided into several parts, each with a specific role. The main menu is the starting point, where the user can choose what they want to do, like applying a rule profile, scanning the system, exporting the current policy, rolling back changes, or checking the logs. Each action is handled by a separate function, which helps keep the script organized and easier to maintain.

The rule profiles (Basic, Hardened, or Custom) are already included in the script and are applied depending on what the user selects. These rules are based on file path, publisher, or file hash, and the user can choose between AuditOnly or Enforced mode depending on the objective.

All components of the tool work together smoothly, while staying independent. This makes the tool flexible and allows for future updates or the addition of new features without affecting the existing structure.

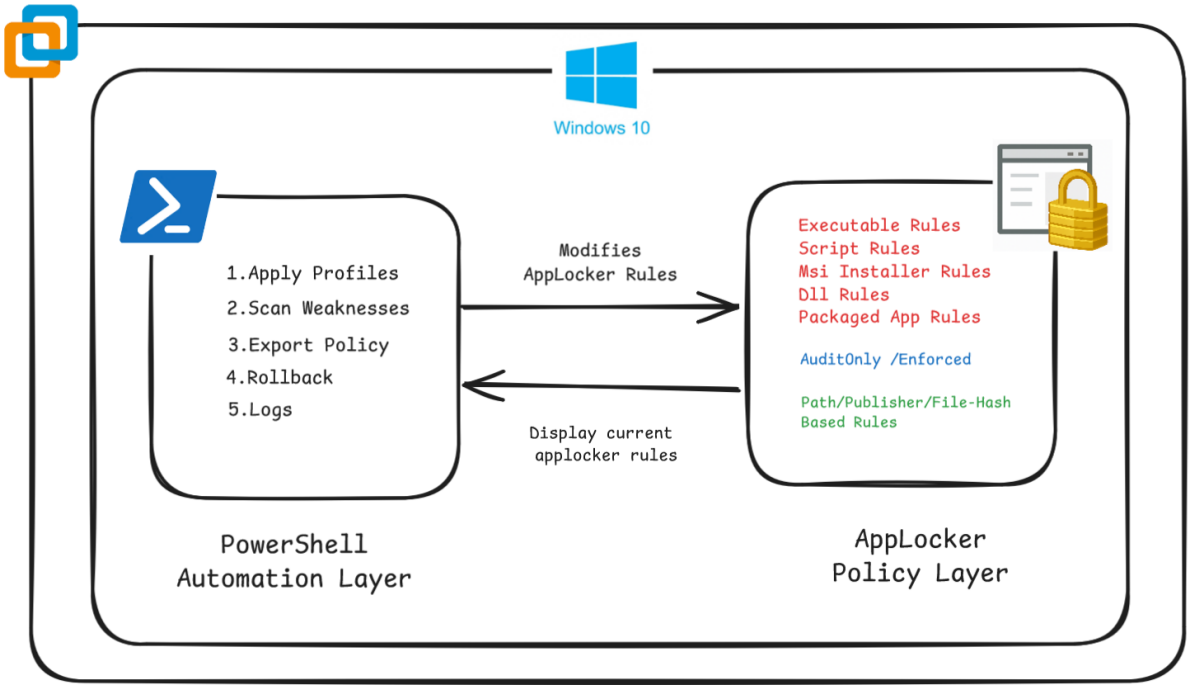

Figure 4.1: Architecture of the HardLocker tool

4.3 Main Features Implementation

The main menu is the central point of interaction between the user and the HardLocker tool. It is designed to be simple and easy to navigate, using a clear text-based interface built with PowerShell. The menu displays the available options such as profile selection, scanning the system, exporting or rolling back policies, and viewing logs. Each option is numbered and described briefly, so users can easily understand what each one does.

The interaction logic is straightforward: when the user selects an option, the script runs the corresponding function. Input is handled securely, and invalid choices are managed by displaying an error message and prompting the user to try again.

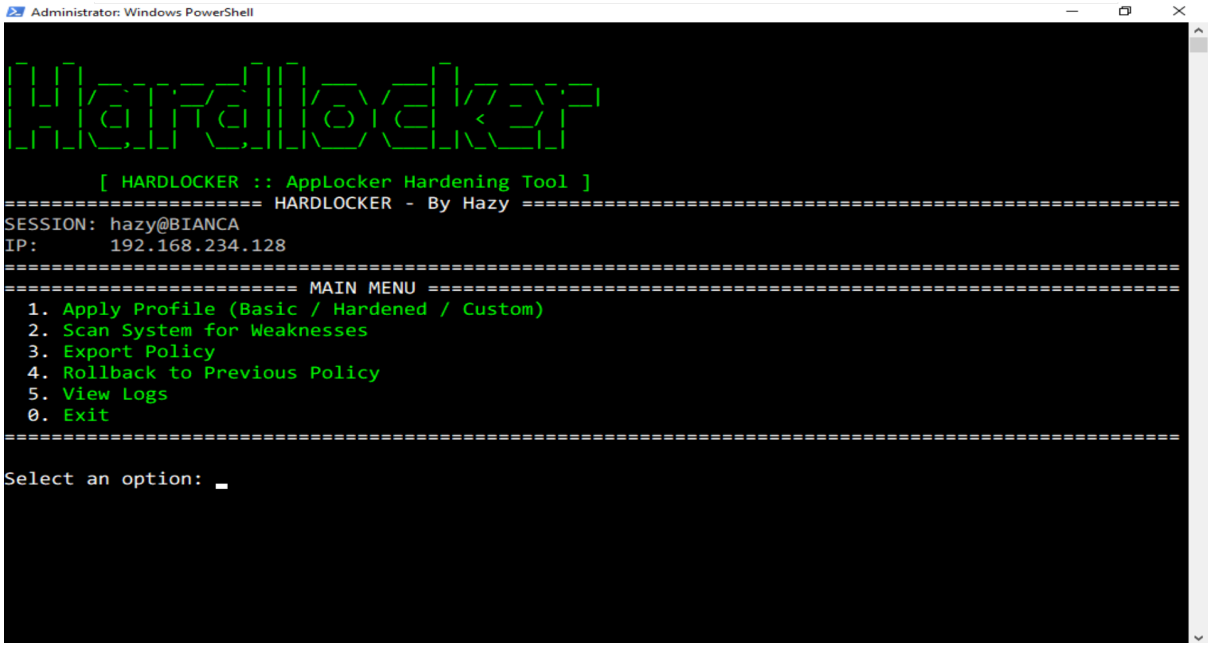

Figure 4.2: HardLocker Menu

4.3.1 Profile Selection Module

The tool offers three predefined profiles to match different levels of security:

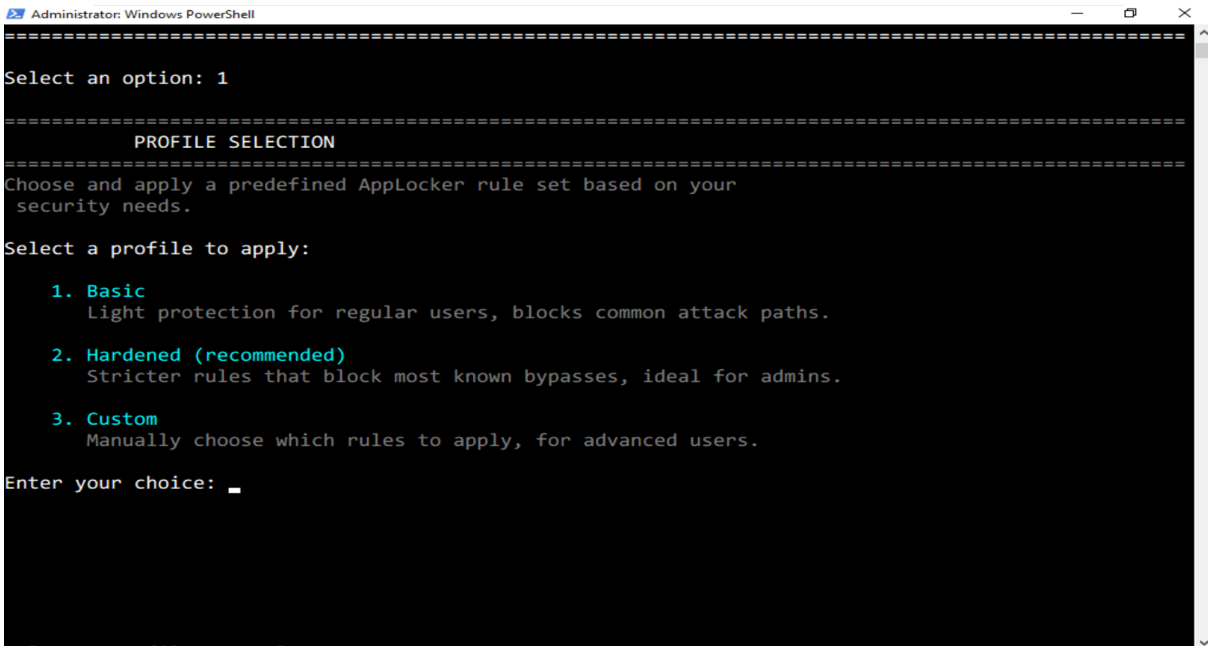

Figure 4.3: HardLocker Profiles

- **Basic**: For users who want to improve security without affecting normal usage. The rules in this profile are selected carefully to avoid blocking legitimate apps.

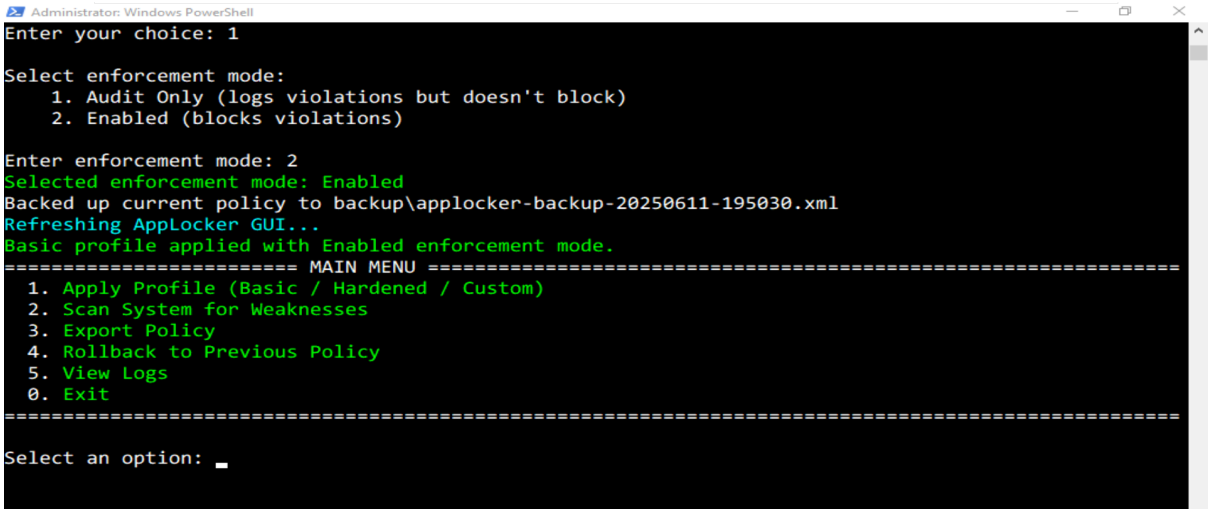

Figure 4.4: Enter Caption

As shown in the image above, after selecting the Basic profile and the enforcement mode, the tool automatically backs up the current policy, applies the selected rules, refreshes the AppLocker interface, and redirects the user to the main menu to continue.

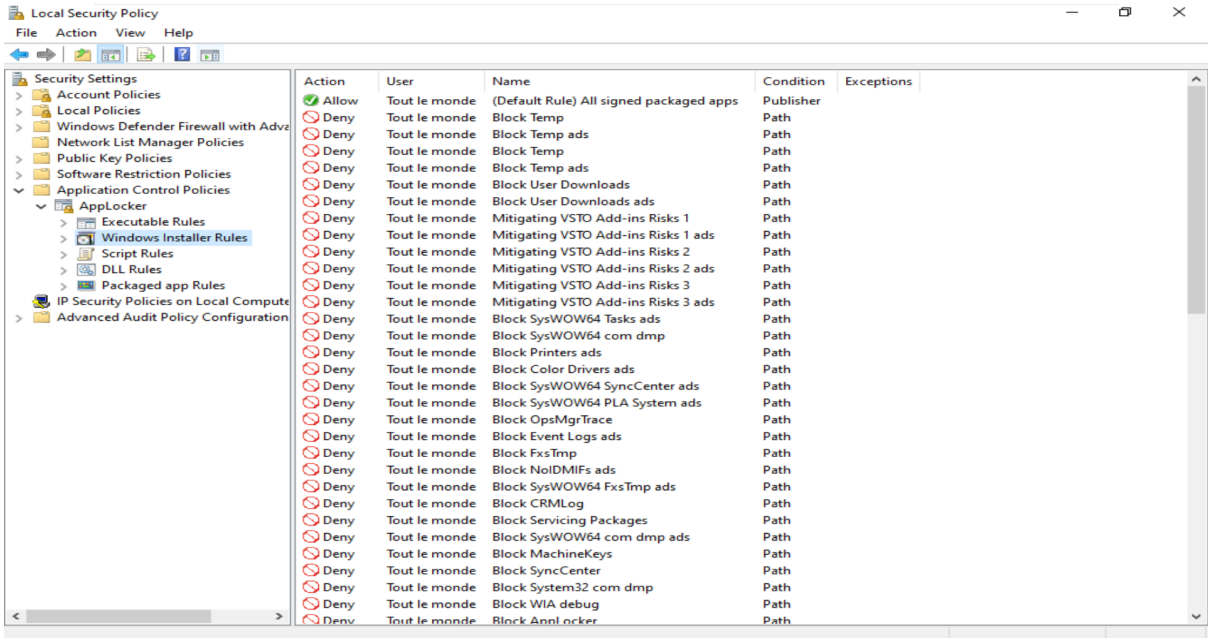

Figure 4.5: Refreshed AppLocker interface showing basic rules

- **Hardened**: Provides stronger protection, targeting more aggressive attack techniques. It is suitable for sensitive environments where maximum security is required. Applied the same way as the basic profile.

- **Custom**: Allows the user to select specific rules manually. This profile is ideal for experienced users who know their environment and want more control.

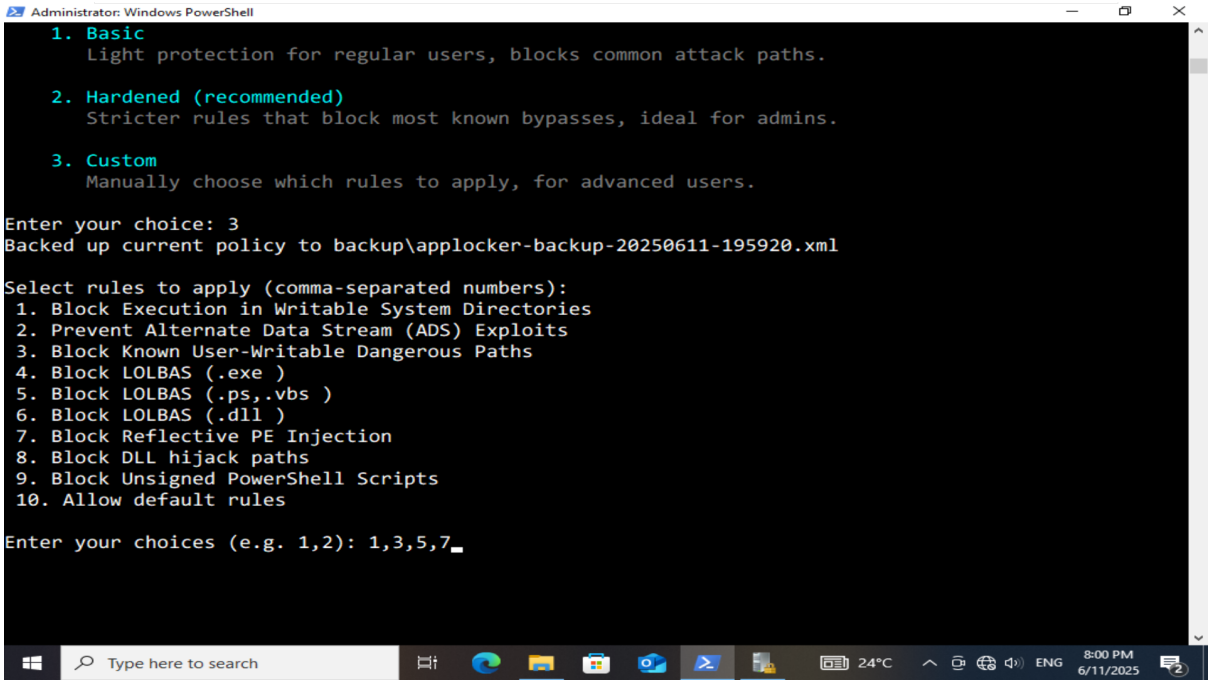

Figure 4.6: The custom profile

With the Custom profile, the user selects specific rules from a list, then chooses the enforcement mode. The tool backs up the current policy, applies the selected rules, and saves them in a custom policy file.

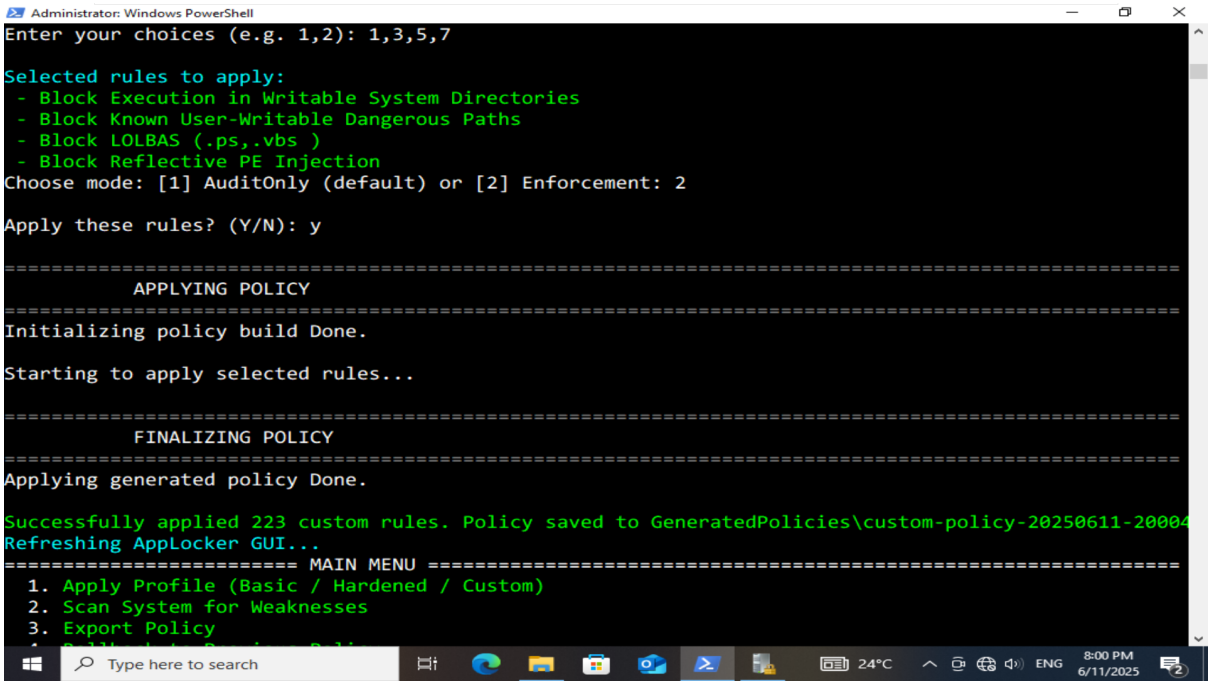

Figure 4.7: Custom rule set loaded in AppLocker

and the AppLocker interface is refreshed once the rules are applied.

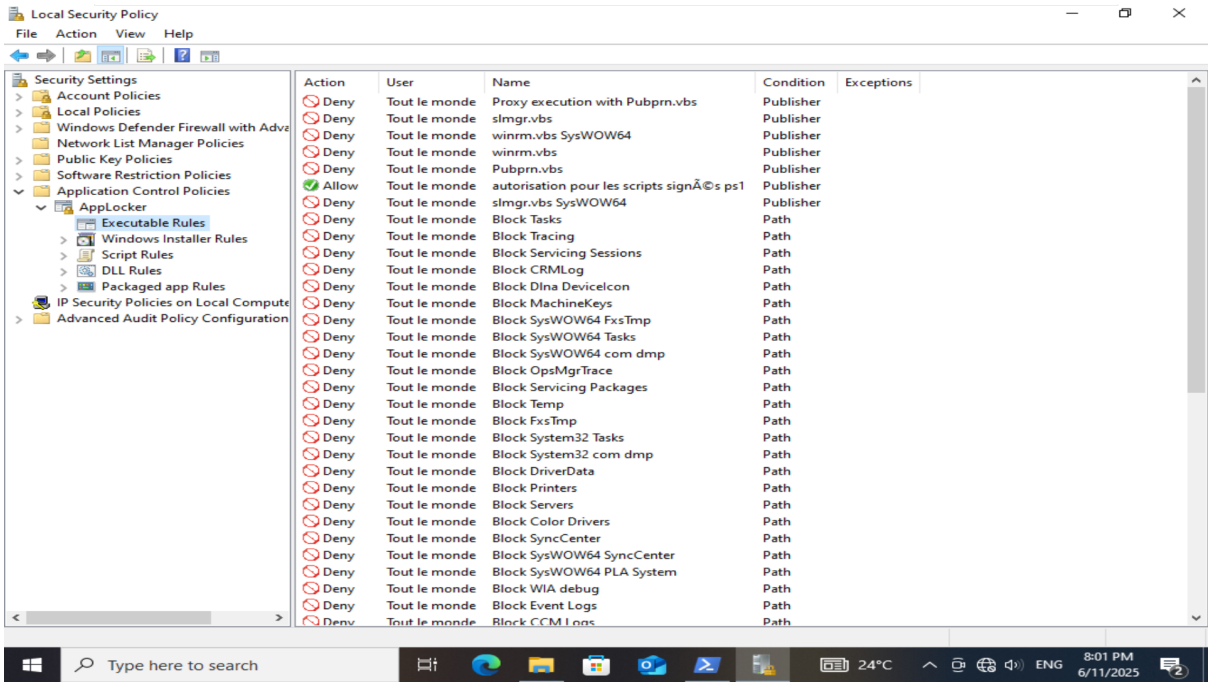

Figure 4.8: Custom rules visible in AppLocker

Technically, the rules for each profile are stored in XML format inside the script. When the user selects a profile, the corresponding XML is parsed and applied using built-in PowerShell cmdlets like Set-AppLockerPolicy. A unique ID is assigned to each rule to avoid conflicts or duplication

4.3.2 Weakness Detection

Before applying any profile, the tool offers a scan feature to detect potential security weaknesses.

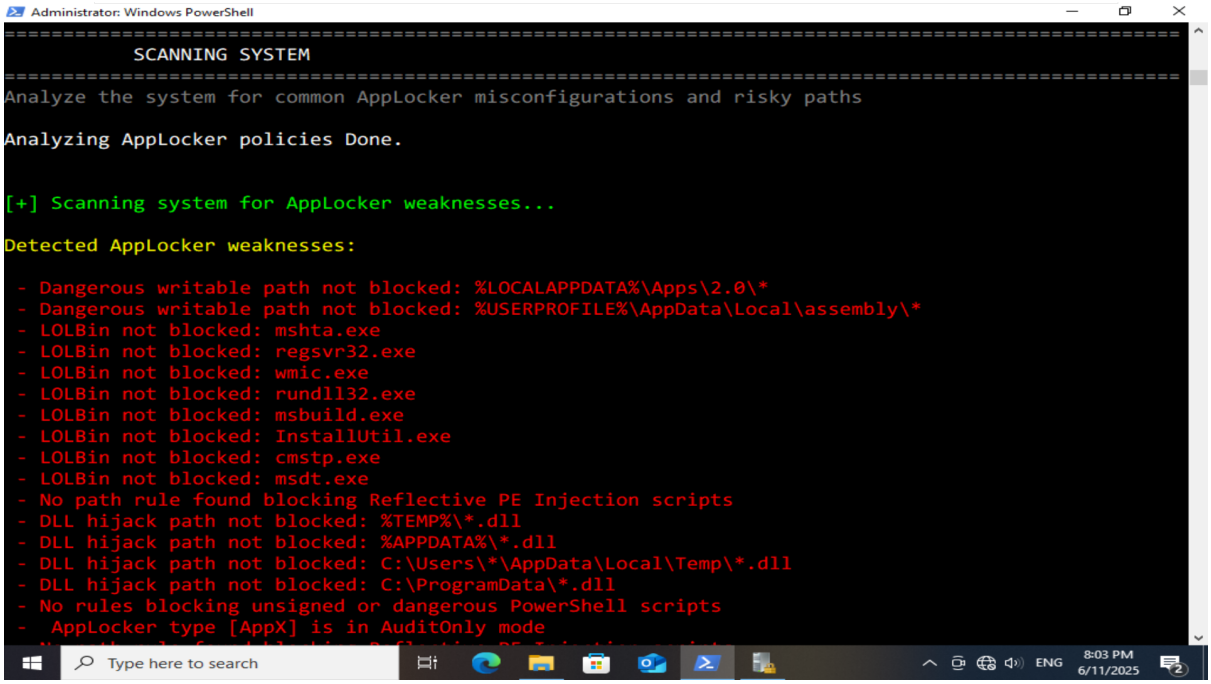

Figure 4.9: Scanning sytem for weakness

This scan checks several common issues, including:

- Absence of AppLocker rules in key categories (e.g., executable, script, DLL).
- Presence of known writable or risky paths.
- Existence of bypass opportunities (e.g., LOLBins that are not restricted).

The detection is done by querying the current AppLocker policy and comparing it with known patterns. The result is displayed in a readable format so the user can decide whether to apply a full profile or customize their own.

4.3.3 Policy Exporting

To give users more control, the tool includes a feature that allows exporting the current AppLocker policy. This is done using the **Get-AppLockerPolicy** cmdlet, which outputs the policy in XML format.

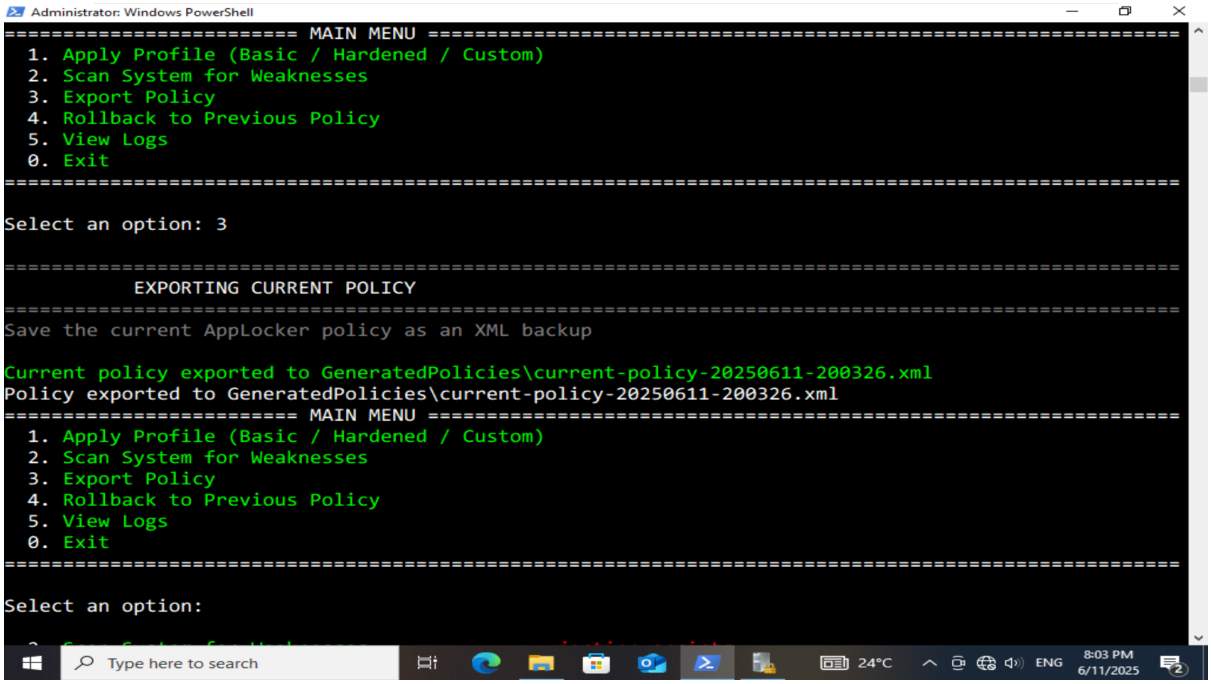

Figure 4.10: Exporting Current Policy

The exported file can be saved for backup, shared with other systems, or reviewed manually. This feature is useful for documentation, auditing, or reapplying policies in the future.

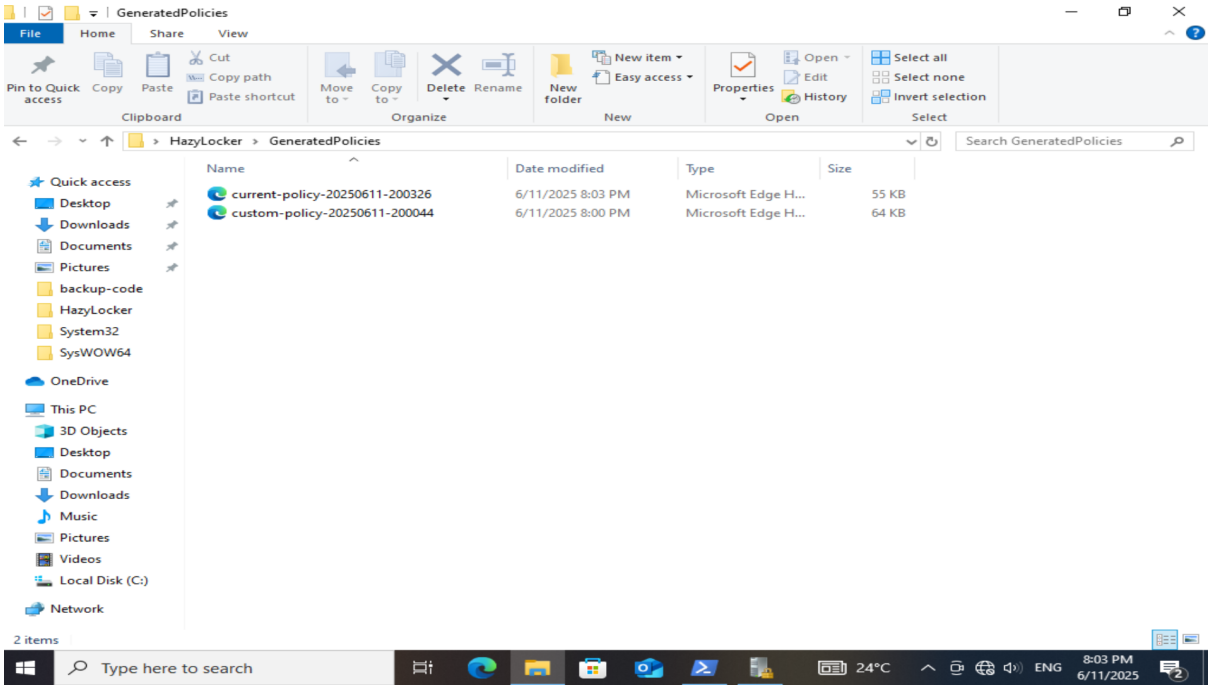

Figure 4.11: Saved Exported rules

4.3.4 Rollback Feature

To avoid accidental misconfigurations, the tool automatically saves a backup of the current AppLocker policy before applying any changes. This backup can be restored later using the rollback function.

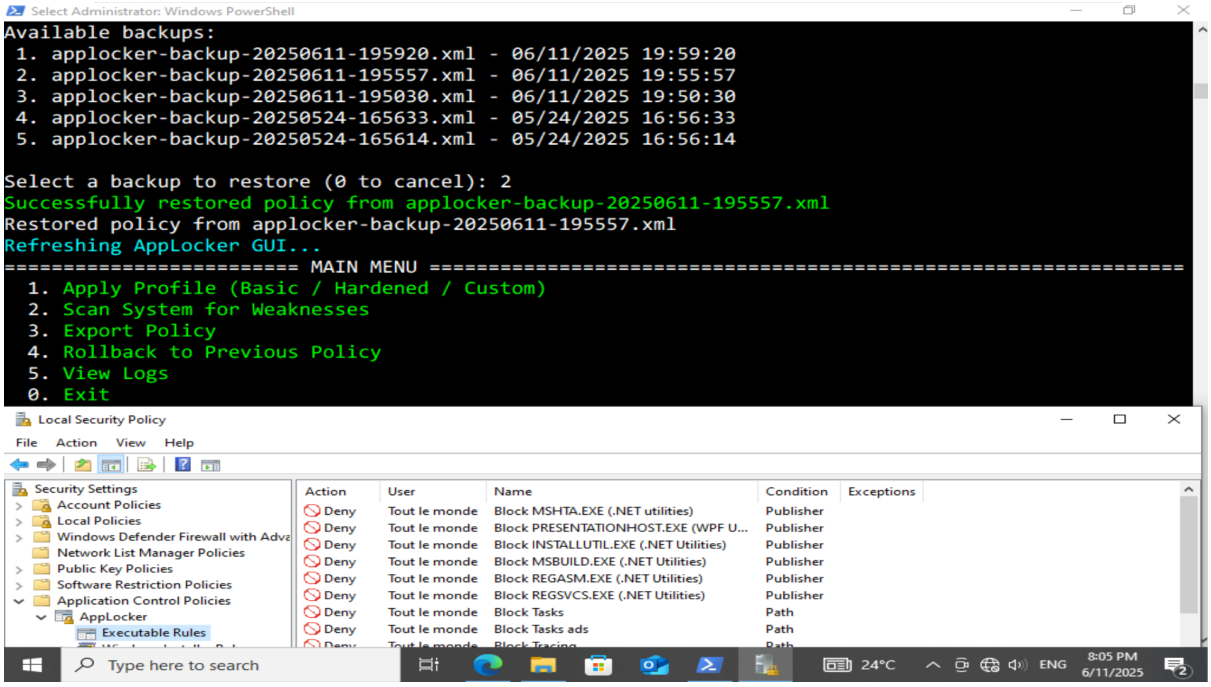

Figure 4.12: Rollback functionality

The rollback mechanism works by storing the exported policy file in a specific folder and reimporting it when the user selects the rollback option. This provides a safety net and allows users to experiment with profiles without risk.

4.3.5 Log Viewer

Every action taken by the tool is logged for transparency and troubleshooting. Logs are stored in a local file that records the date, selected options, applied rules, scan results, and any errors.

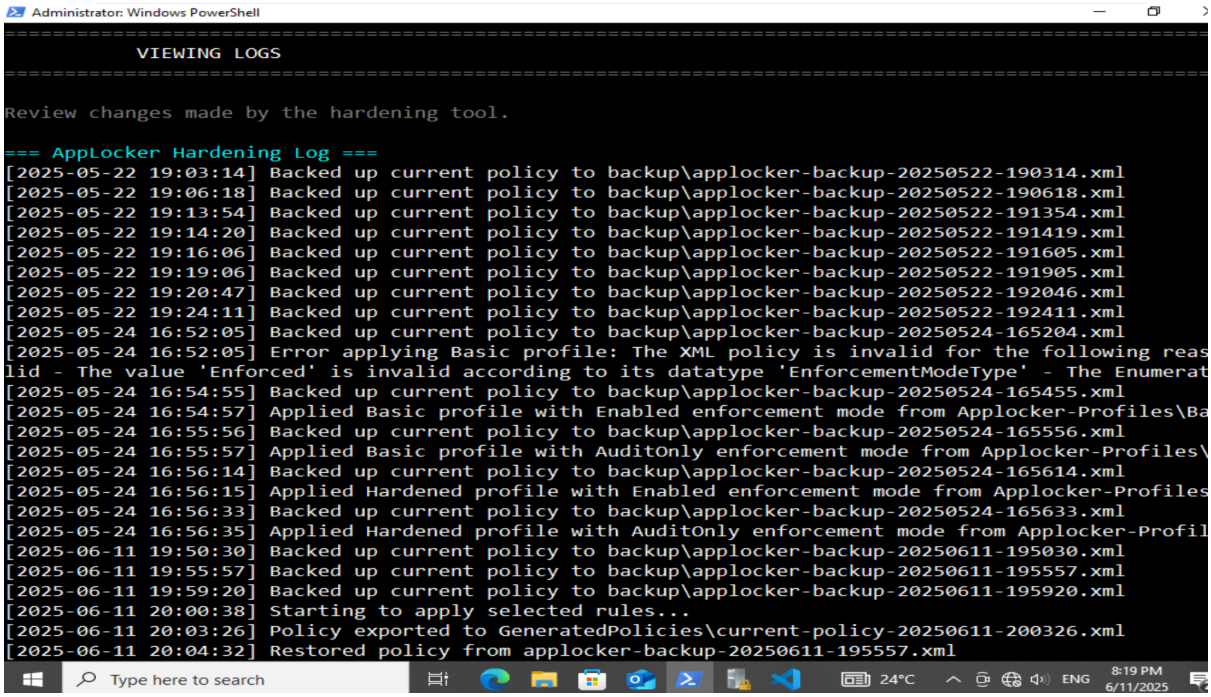

Figure 4.13: HardLocker logs

This feature helps users understand what changes were made, especially in environments where multiple policies might be tested. Logs are also useful during audits or when debugging unexpected behavior.

4.4 Post-Deployment Blocking Test

Once the hardened rules were applied using the HardLocker tool, the effectiveness of the configuration was confirmed by triggering a blocked execution. For demonstration purposes, the execution of INSTALLUTIL.EXE was attempted and was immediately blocked by AppLocker, as expected. Two validations were captured:

AppLocker Event Log (Event ID 8004):
The Windows Event Viewer showed that the binary `%WINDIR%.NET2.0.50727.EXE` was prevented from running under the **AppLocker EXE and DLL policy**.

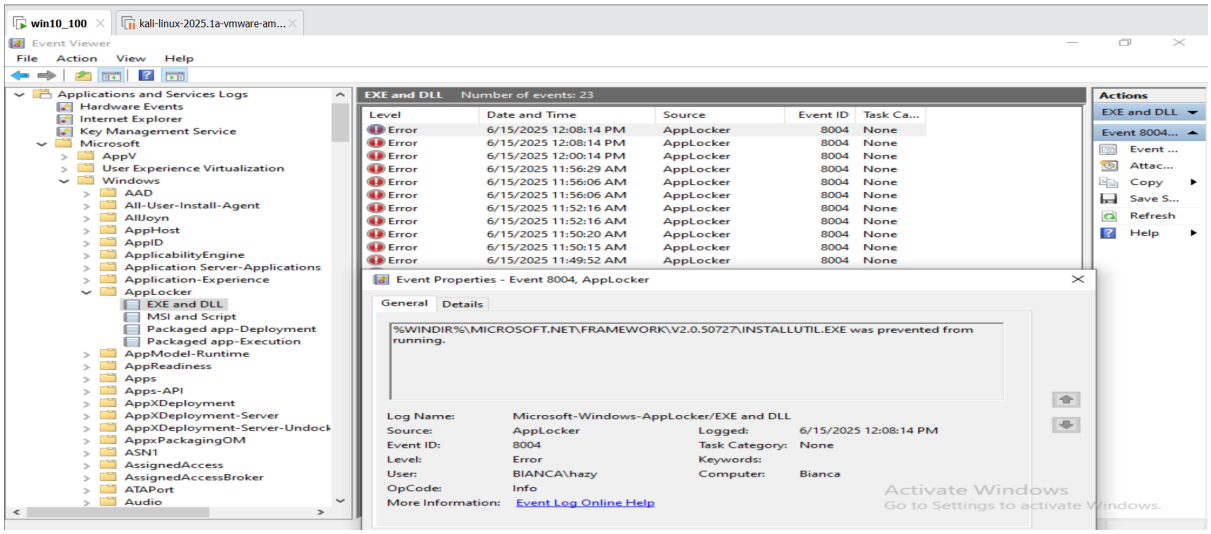

Figure 4.14: AppLocker Event ID 8004 showing the blocking of INSTALLUTIL.EXE

System Notification :
A system-level popup confirmed that the application was blocked by the system administrator, reinforcing that the rule was correctly enforced.

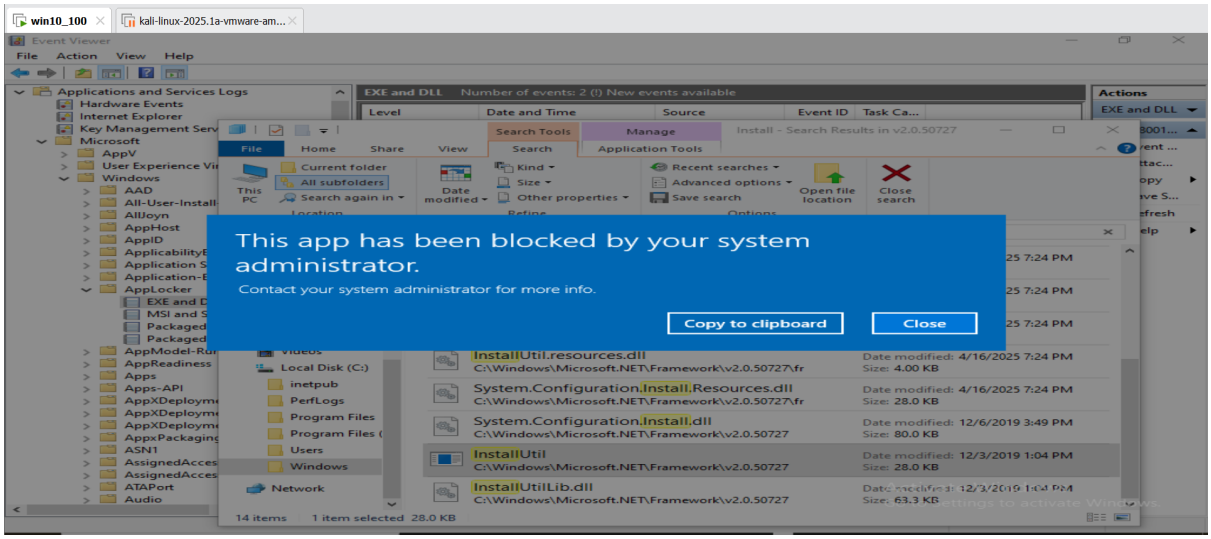

Figure 4.15: System alert showing the application was blocked by AppLocker

These results confirm that the AppLocker hardening profile was successfully applied, and unauthorized applications were effectively restricted as designed.

Conclusion

In this chapter, we described the key components of the HardLocker tool, focusing on how each feature was designed and implemented. The combination of profile-based rule application, scanning, rollback, export, and logging helps make the tool practical, flexible,

and safe to use. Each feature supports the main goal of the project: to make AppLocker easier to manage and more effective in securing Windows systems.

General Conclusion and Future Work

This report marks the culmination of our end-of-study project as part of the Cyber Defense and Embedded Telecommunication Systems program at the National School of Applied Sciences in Marrakech. The project was carried out within LMPS Group, in its Offensive Security (Red Team) division. Its objective was to design and implement an automated methodology for hardening Windows systems through the creation of a custom tool named HardLocker, leveraging AppLocker policies and PowerShell scripting.

In the first stages, we focused on understanding the inherent limitations of native AppLocker policies and how these can be bypassed by adversaries in real scenarios. This analysis revealed the need for a more structured, automated, and adaptable hardening approach. We then moved into the technical realization phase, gradually developing and integrating features such as automated rule application, system scan capabilities, rollback functionality, and support for predefined profiles. Testing in a controlled environment demonstrated the tool's effectiveness in mitigating several known bypass techniques, such as those using LOLBAS binaries or alternate data streams.

The tool was conceived as a bridge between offensive knowledge and defensive solutions , using the logic of how attacks are conducted to proactively block them. This dual vision allowed us to reinforce not only our technical skills in Windows internals and PowerShell automation, but also to explore how security tools can be made scalable, auditable, and usable in real environments.

From a personal and academic perspective, this project was an exceptional learning opportunity. It strengthened our scripting and system hardening skills and gave us hands-on exposure to red teaming workflows. Throughout the development process, we encountered real-world technical constraints that required practical problem-solving and helped us mature professionally.

Looking ahead, several enhancements are envisioned. A primary objective is to fully integrate Hardlocker into Active Directory environments, allowing for centralized and scalable deployment of security rules. Another important evolution involves the development of a scanning module capable of analyzing historical execution logs and system events to automatically suggest directories that should be restricted , making the tool more adaptive and intelligence-driven.Also future versions could include a behavioral learning module

that adjusts rule sets dynamically based on user activity and system role. In addition, introducing predefined configuration templates for different machine types ,such as domain controllers, developer workstations, or kiosk would enable faster and more relevant rule deployment.

Finally, we also aim to provide options for exporting rules to Excel for better visualization and auditability, and eventually integrate Hardlocker with SIEM systems and a graphical user interface to make it more accessible to larger security teams.

This project is the result of several months of dedication and commitment. It allowed us to merge theory and practice, and confirmed our interest in building efficient, offensive-informed security solutions. Beyond the technical outcome, it shaped our mindset and vision for a career in cybersecurity ,where automation, adaptability, and usability are key pillars for modern defense.

Bibliography

[1] : Microsoft Learn, « Applications that can bypass App Control and how to block them », https://learn.microsoft.com/en-us/windows/security/application-security/application-control/app-control-for-business/design/applications-that-can-bypass-appcontrol, [Online 12/03/2025].

[2] : SEI Carnegie Mellon, « Bypassing Application Whitelisting », https://insights.sei.cmu.edu/blog/bypassing-application-whitelisting/, [Online 12/03/2025].

[3] : Black Hills InfoSec, « PowerShell Without PowerShell – How To Bypass Application Whitelisting », https://www.blackhillsinfosec.com/powershell-without-powershell-how-to-bypass-application-whitelisting-environment-restrictions-av/, [Online 10/03/2025].

[4] : GitHub, « AppLocker repositories », https://github.com/topics/applocker?o=desc&s=forks, [Online 17/03/2025].

[5] : ired.team, « Using MSBuild to Execute Shellcode in C# », https://www.ired.team/offensive-security/code-execution/using-msbuild-to-execute-shellcode-in-c, [Online 18/03/2025].

[6] : Evi1cg, « AppLocker Bypass Techniques », https://evi1cg.me/archives/AppLocker_Bypass_Techniques.html#menu, [Online 08/04/2025].

[7] : LOLBAS Project, « LOLBAS Binaries », https://lolbas-project.github.io/lolbas/Binaries, [Online 08/04/2025].

[8] : Cyderes, « How attackers turn mshta against you », https://www.cyderes.com/blog/how-attackers-turn-mshta-against-you, [Online 26/03/2025].

[9] : GitHub, « AppLocker Guidance », https://github.com/nsacyber/AppLocker-Guidance, [Online 16/04/2025].

[10] : GitHub, « AppLocker Hardening Policies », https://github.com/MotiBa/AppLocker/tree/master, [Online 16/04/2025].

[11] : Naim Gaberlo, « Penetration Testing: Application Whitelist Bypassing », University of Piraeus, 2021

[12] : GitHub, « Ultimate AppLocker Bypass List », https://github.com/api0cradle/UltimateAppLockerByPassList, [Online 14/04/2025].

[13] : Hitco, « Prevent bypassing AppLocker using Alternate Data Streams », https://hitco.at/blog/howto-prevent-bypassing-applocker-using-alternate-data-streams/, [Online 17/04/2025].

[14] : HackingArticles, « Windows AppLocker Policy – A Beginner's Guide », https://www.hackingarticles.in/windows-applocker-policy-a-beginners-guide/, [Online 13/03/2025].

[15] : Miko Narhri, « Applocker Implementation », Metropolia University of Applied Sciences, 2024.

[16] : GitHub, « AaronLocker », https://github.com/microsoft/AaronLocker, [Online 30/04/2025].

[17] : GitHub, « Applocker-Hardening », https://github.com/simeononsecurity/Applocker-Hardening, [Online 30/04/2025].

[18] : Microsoft, « AppLocker Design Guide » ,Microsoft Corporation , 2013.

Appendix

Default Rules

When creating rules through the AppLocker interface, you're given the option to add default rules. These help prevent users from accidentally locking themselves out by applying overly strict configurations. For example, if a local admin forgets to set the enforcement mode to "Audit only" and creates restrictive rules, they might lose access to essential tools like the MMC console. To avoid this, AppLocker includes permissive default rules that ensure the system remains usable.
The default rules for Executable rules, Script rules and DLL rules (if enabled) are:

Action	**Subject**	**Object**	**Exceptions**
Allow	Everyone	`%PROGRAMFILES%\*`	No exceptions
Allow	Everyone	`%WINDIR%\*`	No exceptions
Allow	BUILTIN\Administrators	`*`	No exceptions

For the *Windows Installer* rule type, the default rules are:

Action	**Subject**	**Object**	**Exceptions**
Allow	Everyone	Signed by any publisher	No exceptions
Allow	Everyone	`%WINDIR%\Installer\*`	No exceptions
Allow	BUILTIN\Administrators	`*`	No exceptions

For the *Packaged apps* rule type, the default rule is:

Action	**Subject**	**Object**	**Exceptions**
Allow	Everyone	Signed by any publisher	No exceptions

AppLocker Events

All AppLocker events are recorded in the "Applications and Services" logs under the path:

- `Microsoft-Windows-AppLocker/EXE and DLL`
- `Microsoft-Windows-AppLocker/MSI and Script`
- `Microsoft-Windows-AppLocker/Packaged app-Deployment`
- `Microsoft-Windows-AppLocker/Packaged app-Execution`

A list of relevant AppLocker event IDs can be found in the following table.

Table 4.2: List of AppLocker Events

Event ID	Level	Description
8000	Error	Application Identity Policy conversion failed. Indicates the policy was not correctly applied.
8001	Information	The AppLocker policy was applied successfully to this computer.
8002	Information	`<File name>` was allowed to run — file permitted by AppLocker rule.
8003	Warning	`<File name>` was allowed but would have been blocked if in Enforce mode.
8004	Error	`<File name>` was blocked — denied by AppLocker in Enforce mode.
8005	Information	`<Script or MSI>` was allowed to run — permitted by rule.
8006	Warning	`<Script or MSI>` was allowed but would be blocked under Enforce mode.
8007	Error	`<Script or MSI>` was blocked from running.
8008	Error	AppLocker applied on unsupported Windows edition (SKU not supported).
8020	Information	`<Packaged app>` was allowed to run (Windows 8 / Server 2012+).
8021	Warning	`<Packaged app>` would have been blocked under Enforce mode.

8022	Error	`<Packaged app>` was blocked from running.
8023	Information	`<Packaged app>` was allowed to install.
8024	Warning	`<Packaged app>` was installed but would be blocked in Enforce mode.
8025	Error	`<Packaged app>` was prevented from installing.
8026	Error	No packaged apps can run because no Packaged app rules are configured, but Exe rules are enforced.
8027	Warning	No Packaged app rule configured.

Snippet of Rules Designed to Address AppLocker Limitations

The XML snippet below illustrates rules from the custom AppLocker policy implemented to mitigate well-known weaknesses, such as path-based bypasses.These rules were applied using PowerShell with the `Set-AppLockerPolicy` cmdlet.

```
<?xml version='1.0' encoding='utf-8'?>
<AppLockerPolicy Version="1">
  <RuleCollection Type="Exe" EnforcementMode="AuditOnly">
    <FilePathRule Id="00000000-0000-0000-0000-000000000001" Name="Block
    ↪  Tasks" Description="" UserOrGroupSid="S-1-1-0" Action="Deny">
      <Conditions>
        <FilePathCondition Path="%WINDIR%\Tasks\*" />
      </Conditions>
    </FilePathRule>
    <FilePathRule Id="00000000-0000-0000-0000-000000000002" Name="Block Tasks
    ↪  ads" Description="" UserOrGroupSid="S-1-1-0" Action="Deny">
      <Conditions>
        <FilePathCondition Path="%WINDIR%\Tasks:*" />
      </Conditions>
    </FilePathRule>
    <FilePathRule Id="00000000-0000-0000-0000-000000000003" Name="Block
    ↪  Tracing" Description="" UserOrGroupSid="S-1-1-0" Action="Deny">
      <Conditions>
        <FilePathCondition Path="%WINDIR%\Tracing\*" />
      </Conditions>
    </FilePathRule>
    <FilePathRule Id="00000000-0000-0000-0000-000000000004" Name="Block
    ↪  Tracing ads" Description="" UserOrGroupSid="S-1-1-0" Action="Deny">
      <Conditions>
```

```
        <FilePathCondition Path="%WINDIR%\Tracing:*" />
      </Conditions>
    </FilePathRule>
  </RuleCollection>
</AppLockerPolicy>
```