Federal Office
for Information Security

Smart App Control

Analysis of the Smart App Control feature from Windows 11

Document history

Version	***Date***	***Editor***	***Description***
1.0	29.11.2023		Final Version

Table 1: History of the document.

Federal Office for Information Security
P.O. Box 20 03 63
53133 Bonn
Internet: https://www.bsi.bund.de

Table of Contents

1 Introduction

1.1 Generalities

Smart App Control (further on abbreviated by "SAC") has been introduced with Windows 11 version 22H2 (version 22572 or higher) by Microsoft as a new security feature, even present in Microsoft's promotion videos (Microsoft). The main idea of this feature – as announced by Microsoft – is to block the unsafe applications, just allowing the safe ones to run. Presented as an "intelligent", "cloud-powered security service", this one is learning to understand which applications are safe to run or not. But the most general emphasis relative to SAC is the enhancement of the user experience concerning security. The user should just be able to use his or her computer without having to worry about potential security concerns regarding applications. Announced as free and part of the operating system (further on abbreviated by "OS"), this is one of the new security features of Windows 11.

Note this new security feature is disabled if the computer is running Windows in S mode. Also, if sending of optional diagnostic data (Microsoft) has been turn off during the installation procedure or during the evaluation phase of Windows 11, SAC is automatically turn off (deactivated) also.

SAC works alongside another security software, such as Microsoft Defender Antivirus and Microsoft Windows Defender Application Control (Microsoft) or non-Microsoft antivirus tools, for added protection (Microsoft). SAC is claimed by Microsoft to provide "*application control for consumers*" and to allow "*enterprise customers to create a policy that offers the same security and compatibility with the ability to customize it to run line-of-business (LOB) apps*" (Microsoft). One significant point concerns the fact that SAC can only be used on new Windows 11 installs. SAC is only activated when Windows has been either reset or when it has been just freshly installed, a state also called "out-of-the-box". In this context, starting with the operating system, SAC is by default in evaluation mode. It means the feature is leaning from the user, checking if the behavior of this one is compatible with the security feature. During this time, SAC does not prevent the execution of any application. At worst, it just informs the users about that a given application would have been blocked if SAC would have been turned into enforcement (with a real application blocking capacity) mode.

The technology of SAC is described by Microsoft in multiple medias and for the sake of concision, we propose in the current paragraph a resume of the different statements made by Microsoft. In the Microsoft's presentation of SAC (Microsoft), it is written that SAC is "blocking apps that are malicious or untrusted". If malicious directly refers to (known or unknown) malware, the notion of "untrusted" app is a bit blurrier. Officially, Microsoft explains the "trust" evaluation of an application is based on two criteria. At first, the application must be digitally signed, meaning the signature procedure involves a digital certificate that verifies the identity of the developer of that application, and this one has not been tampered with by somebody else after the developer published it. Then, the evaluation is based on "Microsoft's experience" provided through their "intelligent cloud-powered security service" (Microsoft), notified every day with "a huge number" of different applications. Only in the case where the cloud-based analysis service could perform a reliable prediction (based on the experience Microsoft claims to have with the application and the signature of that application), the application may be allowed to be executed or not. If for any reason, it is not possible to get a confident prediction for a given application, this last one is by default considered as untrusted and blocked. The procedure is without appeal, meaning it is not possible to bypass the protection by adding exceptions from the user or the administrator side. The only possible action is to provide feedback to Microsoft, potentially disclosing the blocked application or to require from the original developer of this application to sign it with a valid certificate. Another point is that when SAC is deactivated, it cannot be turn on anymore. This choice is explained by Microsoft "to be sure that there are not already untrusted apps running" in the system before SAC.

In a nutshell, SAC is an automatic and optionless security feature whose purpose is to block applications running on the system if they are not trusted by the security analysis provided by the Microsoft's cloud-based

backend. The only thing that the user may decide is to deactivate it or not – considering that the deactivation procedure is definitive, meaning there is no way back to the activated state.

This brief overview of SAC technology – based on Microsoft's documentation – potentially raises some questions. Especially, it is relevant to understand how this security feature really works. If for practical reasons it is not possible to perform a cloud-based analysis of the learning models used by Microsoft, it remains possible to evaluate "in a black box context" this new security feature. It is worth noting what is provided to Microsoft's cloud-based backend and which is the feedback of that cloud-based analysis. It is possible to specify related questions based on sub-topics. For instance,

- which kind of executable files are in an application concerned by SAC?
- which are the parameters accessible (nor not) configuring SAC?
- where is the initial entry point of the SAC feature?
- which elements of Windows "out of the box" are involved in SAC, how do these elements work internally?
- what do really mean the different modes of SAC (evaluation, enforcement), and how is it possible to pass from one to another?

The resumed answers to these questions are provided in section 1.4.

All these questions frame the context of the present study. The goal is first to provide a technical overview of the SAC feature. As a natural corollary, this document aims removing the mystery about real capabilities of SAC, especially describing its capabilities of detection threats or about the reality behind the new feature. Without ever pretending to be perfectly exhaustive, this study documents the main points of the SAC feature as it is principally designed, removing sometimes internal or irrelevant programming details.

To proceed, the document is structured as follows. The section 1 is an introduction to SAC, covering existing work about SAC and giving an overview of the SAC architecture. The section 3 concerns the general concepts driving SAC, especially the different phases, modes and elements involved in this new security feature. In addition, this second section define the vocabulary which will be used (and required to be understood) for this study. The section 4 is a technical dive into the most relevant elements of SAC, retracing mainly the main steps of a submission of an unsigned executable file to the cloud-based analysis of Microsoft. Also, this section covers the network analysis of SAC (data sent and received to and from Microsoft) and the SAC switching mode procedure (from evaluation to either enforcement or deactivate). The section 5 mainly concerning logging and configuration possibilities when interfacing with SAC. In the end, the section 6 draw the final remarks about this work, especially taking care to resume the main findings of this study but also talking about potential implications of them and questions raising from them.

1.2 State of the art and existing work

It is important to mention that SAC technology has been partially documented by others, especially by Javier Redondo (alias "n4r1b") on his own blog. Based on two main posts, its own study documents first the initialization of the SAC feature in kernel-mode (n4r1b) and about the kernel execution flow of SAC (n4r1b). This technical work has the main advantage of laying the foundations of what is SAC but also to describe some internal details about the kernel of Windows. The present works aims to cover what has not been covered or what could have changed from the time the analysis has been initially performed. Also, the current study tries to provide a more generic and conceptual view of the big picture of what SAC is instead of diving into any implementation details which are – and we will explain that further – not always directly related to SAC.

The initial configuration part has already been explored deeply and this why the intended reader is encouraged to read the post for further details. The second part is rather technical, and it covers a lot of different elements, all togethers at the same time. In our case, in addition of having updated what deserved to be, we aim to provide a more generic view of SAC, especially in the kernel part. Also – and this may be the

main contribution to this work – we provide the analysis of SAC in user-mode (namely in Microsoft Defender Antivirus) and from the network points of view, which was out of the score pf the n4r1b's blog.

1.3 Overview of SAC architecture

SAC technology is a set of dispersed elements within the system. It is not a single service, DLL or driver that would take care of this security feature. It is a technology spread in different components all over the system which aims to block malicious or untrusted apps. Even if SAC is a new feature shipped with Windows 11, this one relies for a large part on existing code, mainly Windows Defender Application Control (further on abbreviated by "WDAC") from Code Integrity (ci.dll) in the kernel. One may consider SAC as a technology built on top of existing features, linking together different components of Windows to provide a new feature.

SAC technology can be split over four main phases. The first one relies on the initialization of the technology. By checking SAC is enable, it consists mainly of knowing in which state (enforce, evaluation or deactivated) it is. This phase happens once for all, at booting time. The second phase checks if an executable image loaded in memory should be verified to know if it is legitimate to be executed or not. This verification is performed in a third phase, mainly by requesting external feedback to the Microsoft's cloud-based backend via Microsoft Defender Antivirus. In this last case, sub-phases apply by extracting information from the analyzed file, processing them, and asking an analysis coming from the Microsoft's cloud-based backend. In the end, the fourth phase applies the decision returned by Microsoft Defender Antivirus analysis.

It is worth noting that the different phases of SAC belong in kernel-land and in user-land, notwithstanding the internet connection with Microsoft's servers, as represented in Figure 1. The final mechanisms where the analysis is triggered and where the decision is finally applied belongs in kernel-mode. But the decision making (to know if an image must be executed or not) procedure belongs to user-land, directly provided from Microsoft Defender Antivirus's analysis, and more specifically related to a notification to the Microsoft's cloud-based analysis.

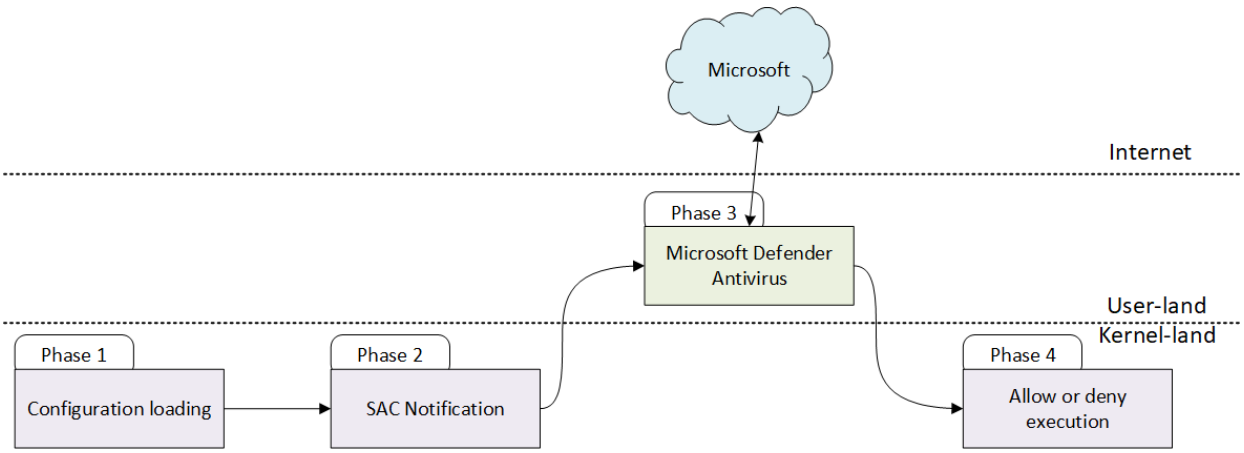

Figure 1. Main phases of SAC technology.

The different phases of the SAC procedures are described in the sections 3 and 4. Because it has been extensively covered in (n4r1b), the present study does not cover in detail the configuration loading of SAC in kernel-mode (concerning the ci.dll library). This is the main reason why the present study focusses more on phases 2, 3, and 4, from ci.dll to Microsoft Defender Antivirus.

The ci.dll library usually plays a central role in implementing core functionalities of the WDAC. It is responsible for critical tasks, including parsing and enforcing of WDAC policies. In this context, WDAC policies works as a centralized point for configuring and managing the WDAC capabilities, with SAC being one of these capabilities. Within the context of the WDAC, Microsoft Defender Antivirus extends additional verification capabilities of the WDAC. This includes providing in-depth analysis, conducted by the Microsoft cloud-based backend, as an integral part of SAC. Once Microsoft Defender Antivirus has performed its

analysis work, this one delivers a HTTPS request to the Microsoft's cloud-based backend for further reputation information concerning the targeted application. Based on the received answer, feedback is provided to the kernel (Ci.dll) for taking a decision (blocking the application or not, notifying the user or not).

1.4 Questions & Answers regarding the project

This study has been driven with the mindset to answer specific questions concerning SAC. This subsection resumes all the different questions answered in the SAC study. The questions are divided into different subsets each representing a phase of the SAC feature (as described in this study through the section 1.3).

1.4.1 Kernel pre-notification

Question	In which context SAC is notified in kernel mode?
Answer	SAC is notified as part of the image mapping procedure, especially when a new process is about to be created. Instead of using a dedicated Mini-Filter technology as any antivirus software would do, the notification is directly performed by the kernel within ci.dll. More information in section 3.3.1.

Question	Which kind of files are required to be analyzed by SAC?
Answer	Except for Microsoft-signed applications, all user-mode applications are automatically analyzed by SAC. It means that executable files (.exe, .dll) and installers (.msi) are relevant for SAC. Drivers, script files (for instance, powershell or javascript), documents, and other non-executable files are not submitted to SAC for analysis. More information in section 3.3.1.

Question	What is the relationship between SAC and WDAC policies?
Answer	The notification from the kernel to SAC is initialized within the WDAC policies management, in ci.dll. In the kernel of Windows, SAC can be directly enabled or disabled through a WDAC policy. But such a WDAC policy is not configurable as a regular WDAC policy, where it is possible to detail specific behaviors. It means that if SAC is disabled by WDAC policy, the whole feature is in the end deactivated. In the kernel, SAC is part of the WDAC evaluation procedure, with some lines of codes added to the WDAC evaluation procedure. More information in section 4.1.

Question	How does the kernel notify Microsoft Defender Antivirus for analysis?
Answer	The kernel notifies Microsoft Defender Antivirus though RPC. The notification procedure is performed in two steps. At first, the kernel calls an RPC interface in a service called "cryptcatsvc". Then, the cryptcatsvc service calls with an RPC interface to Microsoft Defender Antivirus to transfer the initial notification of the kernel. More information in sections 4.1.2 and 4.1.3.

1.4.2 Microsoft Defender Antivirus Analysis

Question	How do the kernel-mode components communicate with user-mode Microsoft Defender Antivirus in the context of SAC?
Answer	The communication is performed through an RPC interface which exports different functions to interface with Microsoft Defender Antivirus. This interface is protected through a specific SDDL definition. More information in section 4.1.3.

Question	What are the different modules involved in Microsoft Defender Antivirus for SAC?
Answer	There is not a single module for SAC, but instead of, many elements used for different purposes. For instance, in Microsoft Defender Antivirus, it involves MsMpEng.exe, MpClient.dll, MpSvc.dll, and MpCommunication.dll as the main components used in the context of SAC. More information in section 3.1.11 and accord section 4.

Question	What are the main steps in the analysis process of Microsoft Defender Antivirus for SAC?
Answer	The first step is to ensure that SAC technology is correctly initialized (and if not, Microsoft Defender Antivirus initializes it) before asking for a memory scan operation to Microsoft Defender Antivirus. This scan procedure extracts different data (see section 4.1.5.2) from the executable file before requesting the Microsoft's cloud-based backend to get feedback about the file's reputation processed during a cloud-based analysis. More information in section 4.1.4.

Question	Which information is retrieved from the analyzed file?
Answer	Signatures and hash of the file's content are the main elements extracted from an analyzed file, including some metadata such as the full path name where the analyzed file belongs. In no case we observed, the full content of the analyzed file is disclosed during the cloud-based analysis of Microsoft. More information in section 4.1.5.2.

1.4.3 Microsoft Defender Antivirus and Cloud Communication

Question	What is the network communication protocol used?
Answer	HTTPS is the protocol used to communicate with the Microsoft's cloud-based backend. More information in section 4.1.5.1.

Question	Are there any security measures to secure the communication?
Answer	A check of the certificate chain is performed to ensure the communication is not intercepted with the WinHTTP API. More information in section 4.1.5.1.

Question	Which information is provided to the Microsoft cloud-based backend?
Answer	To get a confident prediction for each of the file to be analyzed, a comprehensive report is generated for each of them by Microsoft Defender Antivirus. This report is serialized with the "Bond" framework and transferred through a HTTPS POST request to the Microsoft cloud-based backend, where each report will be used for an analysis. More information in section 4.1.5.2.

Question	Which information is retrieved from the Microsoft cloud-based backend?
Answer	The responses from the cloud-based analysis are presented in the form of dynamic signatures, serialized using the "Bond" framework. This signature encapsulates the outcomes of the cloud-based analysis, including feedback about the file reputation and potentially a status update for SAC (from evaluation to enforcement or deactivated). More information in section 4.1.5.2 and 4.1.6.2.

Question	What is happening when there is no internet connection?
Answer	When there is no internet connection, there is no possibility to perform a cloud-based analysis. In this context, there is a difference if the file about to be executed had already been correctly analyzed by SAC. If it had been already allowed, there is a cache mechanism - which has previously been set for allowed files - guaranteeing the execution of that file. Otherwise, if the cache mechanism is not present or the file has never been analyzed, due to a lack of feedback, the execution is refused. More information in section 3.3.7.

1.4.4 Kernel post-notification

Question	What is the content of the answer returned by Microsoft Defender Antivirus?
Answer	The content of the answer contains many elements but two are the most relevant. On the first hand, a set of values representing the reputation of the submitted metadata representing the file being analyzed. On the other hand, it may have a specific set of values requesting a configuration update of SAC, mostly to switch from evaluation to enforcement or deactivated mode. More information in section 3.3.7.1.

Question	What are the resulting actions based on the file's reputation?
Answer	There are three possible results: known to be good, known to be malicious or unknown. The result is a combination of values where a threshold defines a limit used to know if the file’s reputation is sufficient or not. More information in section 3.3.7.1.

Question	How the status of SAC impacts the answer given to an analyzed file?
Answer	If the reputation of the file is known to be good, the execution of that file is guaranteed. Otherwise, if the reputation is unknown or known to be malicious, the resulting actions are twofold, depending on the status of SAC. Either SAC is in evaluation mode and in this case, there is only a notification displayed to user’s eyes, but the execution is guaranteed, or SAC is in enforcement mode and in this context, execution is refused with a message box displayed to the user. More information in section 3.3.7.2.

1.4.5 SAC switching state

Question	Is the Microsoft cloud-based backend able to change the state of SAC?
Answer	In the cloud-based analysis answers returned after an analysis, there may be a specific answer providing this information to turn SAC into enforcement mode or to deactivate it. In addition, there is a manual procedure through the Smart App Control setting in Windows Security settings. In the end, there is a synchronization between the configuration of Microsoft Defender Antivirus and ci.dll. More information in section 4.1.6.2.

1.4.6 Log & Tracing

Question	Is there a way to observe SAC technology through ETW?
Answer	SAC is neither a single nor a dedicated component, meaning that there is no direct way to collect all logs related to this one. But it is still possible to track kernel and Microsoft Defender Antivirus's ETW providers to monitor the activity. More information in section 5.2.

Question	What are the configuration options of SAC?
Answer	The SAC configuration can be easily accessed through the Smart App Control setting in Windows Security settings or Windows policies. But the internal configuration of SAC mostly depends on Microsoft Defender Antivirus and many other configuration components, including third party software installed (i.e.: other antivirus software). More information in section 5.1.

2 Executive summary

As defined with the German Federal Office for Information Security, the following analysis has been performed on the release of the Windows 11 Enterprise system build VL 22621, 64-bit, English (United-States) language. The analysis presented in this work was performed by applying static and dynamic code analysis methods using the Windbg debugger and the IDA disassembler.

The present document analyses the **Smart App Control** (SAC) feature introduced in Windows 11 version 22H2 (version 22572 or higher) by Microsoft. This feature is presented by Microsoft as a "cloud-powered security service", which first learns from the user's activity to understand which applications are safe to run or not. When the learning (evaluation) period is over, SAC can be turned on or off, depending on the Microsoft's cloud-based backend decision if the user does not force SAC to be turned on or off.

The main objective is to provide an in-depth technical analysis of the components of SAC and their interactions together. To proceed, a deep reverse engineering analysis of the different components of SAC was performed and experiments were performed when necessary. The present document proposes a deep technical overview of the SAC feature, especially in the context where a file is about to be executed by the system and SAC will be therefore notified for an analysis.

In this context, the present study determines which files SAC checks and how SAC ensures that these files are not replaced after a successful check. SAC evaluates user-mode applications (see section 3.3.1). It concerns applications (EXE and DLL files) and installers (MSI files). Drivers are directly handled by the kernel by regular Code Integrity checks (including AppLocker and WDAC if configured). Scripts are not directly analyzed by SAC, but programs responsible to interpret scripts can be analyzed by SAC as any regular executable files. There is an exception for files digitally signed by Microsoft, otherwise, all files (including executables digitally signed by Authenticode certificates) are transmitted by SAC for a cloud-based analysis. The tampering protection is ensured with a mechanism of Kernel Extended Attributes (definition given in section 3.1.4), ensuring that a file modified after an analysis will be automatically retransmitted to Microsoft for a subsequent analysis if it is about to be executed. This protection is designed so that only drivers (administrator rights only would not be sufficient) can tamper with such a mechanism.

Despite the fact SAC is presented to be a new feature and it directly provides a new security functionality, SAC is not a new "component" of Windows. This feature mainly relies on existing elements of Windows (namely Code integrity check in the kernel, WDAC, and Microsoft Defender Antivirus). Say otherwise, SAC is not implemented in a single file but in different elements of the OS. This is the kernel-mode Code Integrity check (ci.dll) which notifies Microsoft Defender Antivirus for further analysis, resulting in the end in an HTTPS request to the Microsoft's cloud-based backend (see section 3.3). From a technical point of view, SAC is mostly a subset of lines of codes inserted in existing features to link together different components which already exist in the system for different purposes.

The main difference between SAC in evaluation mode and SAC in enforcement mode is in the way the feedback provided by the cloud-based analysis is considered (see section 3.3.7). On the one hand, when SAC is still in its evaluation mode and if the feedback is negative, a message is displayed to the user, informing the last one that the expected file to be executed would have been blocked if SAC would have been set in enforcement mode. On the other hand, if SAC is set to be in enforcement mode, the decision of the cloud-based analysis is respected by the system which can refuse the execution of a file, depending on the cloud-based feedback.

To proceed to such an analysis, the present work analyses which data is transferred to the Microsoft's cloud-based backend during an analysis notification (see section 4.1.5). The data is transferred by HTTPS connection using the "Bond" framework to encode the content data. When decoding the information exchanged (see section 4.1.5.2), we observe metadata about the analyzed file, including its digital signature (if there is at least one) and hashes from its content. Also, there are some file system metadata such as the full path name of the analyzed file, but not its full path name. The data transferred also include information

about the Microsoft Defender Antivirus software (for instance, configuration and version numbers) and the system (for instance, the current version of Windows 11). We did not observe that the full content of a file is sent to Microsoft's cloud-based backend during a SAC notification.

In the last section 5, we provide information about the configuration and logging capabilities of the SAC feature.

In our humble opinion, SAC may be considered as a kind of automatic way to use WDAC. The trade-off for the user or organization is to provide optional diagnostic data to Microsoft. This acceptance of this trade-off should be evaluated by any organization. Configuring a WDAC policy for many computers in an organization and maintaining it for different kind of users can be a complex task. By "outsourcing" the WDAC policy definition to the Microsoft's cloud-based backend evaluation related to Microsoft Defender Antivirus, SAC automatizes the full process of WDAC policy definition and maintenance. That way, the files are executed in the system if and only if their reputations are known to be good. This design introduces three direct consequences.

- Users having a large variety of executable files whose reputation is unknown would see a lot of their applications blocked by SAC. This concerns for instance developers who create and execute different instances of a single program when developing it. This is why SAC evaluates if the feature may "fit" to the user's usage of the system and if not, the evaluation provided by the Microsoft's cloud-based backend decides to deactivate SAC for this kind of users. In this context, some users are automatically excluded from SAC if they do not decide directly to turn SAC to enforcement mode manually.
- Since it is not possible to turn SAC back to evaluation or enforcement mode once it has been deactivated without resetting the whole Windows 11 operating system (with a fresh install, for instance), it means that the user's behavior and the system's configuration during evaluation mode is decisive.
- It explicitly requires almost always an internet connection to be efficient. Otherwise, the user's system can be turned hard to use, except for Microsoft signed applications. As a direct consequence, it also means the agreement of the users to disclose information to Microsoft.

If SAC is not an antivirus by itself, it mainly relies on Microsoft Defender Antivirus. But at the opposite of Microsoft Defender Antivirus which uses regular standalone drivers to be notified when a file is about to be executed, the SAC feature is deeply embedded into the kernel, and it directly notifies Microsoft Defender Antivirus. This might raise the question of the central position of Microsoft Defender Antivirus in this context and the position of this one comparing to other antivirus vendors which may be running in the system.

3 Concept and terms

3.1 Terms specific for SAC analysis

Regarding the Smart App Control technology, it is relevant to define specific terms that will be used subsequently in this report. The definitions provided are coming from Microsoft or other sources duly referenced when it is relevant. But in some specific cases, when there is no valid definition which is either enough accurate or relevant for specific concepts and terms, we propose our own definition. In these cases, we explicitly explain that the definition is provided by us.

3.1.1 Configurable and unconfigurable Code Integrity

The notion of Code Integrity in Windows is spread over two main concepts. These two concepts are already largely covered in the AP-7 from Sisyphus Project (ERNW GmbH). On the first hand, there is a what we call the **unconfigurable code integrity**. It concerns all checks performed by Windows to check if an executable file (whatever is its kind) is valid to be executed or not. It principally matters for drivers as a security feature, only loading signed drivers. Also, the unconfigurable Code Integrity concerns the security of the kernel of Windows, first introduced with **Kernel Patch Protection** (KPP) (Microsoft), informally known as "*PatchGuard*" and then enhanced by **Memory integrity** (Microsoft), also called hypervisor-protected code integrity (HVCI). The Memory integrity is based on **Virtualization-based Security** (VBS), a technology using the Hyper-V hypervisor to enhance the security of the Windows operating system. The system is divided into at least two virtual machines called Virtual Trust Levels (VTLs), with at least one owning a "secure kernel" protected by the hypervisor and in charge of the security checks.

The notion of **configurable Code Integrity** concerns the possibility for the administrator of the machine to define which code should be allowed to be executed on the system. It concerns AppLocker (Microsoft) and WDAC (Microsoft) implemented in the ci.dll library. In both cases, based on different criteria (called WDAC policies or AppLocker rules) including metadata such as the execution path, the file name or the installer provider's identity or the content of the file itself (signature, hash, …), it is possible to define if an executable file will be allowed to be run by the system or not. And because this kind of criteria is freely configurable by the user, we define it in opposition to the unconfigurable one own by the kernel of Windows.

An historical retrospection about the notion of Code Integrity in Windows is provided in section 3.2 for further reading.

3.1.2 Remote Procedure Call

In our context, a Microsoft **Remote Procedure Call** (RPC) (Microsoft) defines a programming technology used for inter-process interactions, and more precisely by calling a function potentially in another process. The goal of this technology is to hide the communication details for the programmer who can focus on the details of the application's implementation.

This technology can be used for inter-process communication facility, transferring the execution flow of a given process to another one and vice-versa. Three levels of RPC interface are available. At first, on a network where it is possible to call a function hosted by a process running on a different machine on a network. In SAC, this capacity is not used. A second level is the effective inter-process level, meaning a given process calls a function in another process. The last level is local to a process, meaning a function is called in the context of a single process using RPC technology. Although perfectly superfluous comparing to a regular function call, this last case is comparable to localhost for network, when an application requests a server or the same physical machine. The last two RPC levels (inter-process and local to a process) are used in SAC.

RPC architecture involves a set of "client" and "server" applications (wherever are located these applications, including the possibility to confuse them into a single application). The server application provides an interface of functions "exported" to any application connecting to it. To export such an interface, the server

application registers itself giving it a "name" based on a GUID though a specific API, including the `RpcServerRegisterIf3` function.

Concerning the client application, this one must interface with the server. The server is identified with its GUID by the client. This is usually the conjunction of the `RpcBindingCreate` and the `RpcBindingBind` functions which is used to get access to the server's RPC interface for a client.

Comparing to other programming technology, RPC has many advantages. At first, it allows to authenticate client applications based on their access token (inherited from the user running the process). That way, it is possible to filter who can access to a specific RPC interface. Then, the RPC technology provides "impersonation" techniques (Microsoft), allowing the server's function called to execute in the security context of the caller, limiting potential elevation of privileges. In the end, the RPC technology when used locally relies on Advanced Local Procedure Call (ALPC), an undocumented technology used to exchange messages in Windows which has relatively good performances.

3.1.3 Notion of Signature

The notion of signature is twofold in this study. On the first hand, one may consider a signature as a cryptographic **digital signature** (Microsoft) used to sign a binary file. Necessary for a driver (Microsoft), this technique can be extended to any kind of MZ-PE executable files (DLL, executable, driver) or Microsoft Software Installer (MSI) file.

On the other hand, one may consider **the signature of the Microsoft Defender Antivirus**. Partially documented thanks to a work of reverse engineering in (Bar and Attias), the antivirus' signatures help to identify executable files already known as malware. In this document, we make the distinction between **static signatures and dynamic signatures**. On the first case, static signatures are used to identify already known malware and they must be preloaded into the Microsoft Defender Antivirus's signatures database[1]. On the second case, dynamic signatures are more "compartmental" signatures, not targeting a specific malware but instead a specific behavior used by malware. That way, it is possible to catch potential new or unknown threats using some specific and well identified behaviors. With both approaches (static and dynamic signatures), Microsoft Defender Antivirus provides a protection against both well-known and potentially unknown malware.

In SAC, this is especially the notion of digital signature which is relevant. A digital signature is a cryptographic procedure used to ensure the integrity of a binary file and the identity of the person who signed the binary file. As a specific point, the integrity check provided by the signature is based on a hash procedure. This hash is useful for Code Integrity check.

There are two different ways to compute the hash of a binary file. On the first hand, a flat file hash method can be considered, meaning that the hash of the file is the one computed from the file hash, as the file's content is stored on the hard drive. On the other hand, the WDAC uses the **Windows Authenticode Portable Executable Signature Format** (Microsoft). This one omits the file's checksum, the Certificate Table, and the Attribute Certificate Table, removing management overhead to avoid revising the policy hash rules when the digital signature on the file is updated. In the end, different kind of hashes (SHA-1 or SHA-256) may be produced based on how the file is signed and the scenario in which the file is used. For instance, if the file is page-hash signed, the WDAC validates each page of the file, avoiding loading the entire file in memory to calculate the hash.

[1] *The Microsoft Defender Antivirus's signature database, which is typically located at 'C:\ProgramData\Microsoft\Windows Defender\Definition Updates,' contains threat detection resources. Within this directory, files with the '.vdm' extension house a collection of multibed files named 'modules.' These modules serve as repositories for essential malware signatures and patterns.*

3.1.4 Extended Attributes and Kernel Extended Attributes

The notion of **Extended Attributes** (EA) refers to a file system feature used to associate metadata with a file stored on disk. On Windows, they are mainly supported on NTFS (and on FAT) under the name of NTFS streams (Microsoft). An EA is a set of a name of the attribute (which is a regular string of characters) and the data associated to this attribute. If it is possible to directly interface with them from the command line and with the user-mode API (Microsoft), the real API interface belongs in kernel-mode with the `ZwQueryEaFile` and the `ZwSetEaFile` functions.

The notion of **Kernel Extended Attributes** (Kernel EA) relies on regular EA, but they are only accessible to kernel-mode code (any user-mode attempt to temper with Kernel EA will be silently ignored) (Microsoft). Introduced with Windows 8, only NTFS supports Kernel EA. To set a Kernel EA, this extended attribute must begin with the prefix "$Kernel." followed by a valid EA name string. Only accessible to drivers (being administrator would not be enough to modify it), this mechanism is used to boost the performance of image file signature validation. Note that under specific circumstances, an auto delete of Kernel EAs feature was added to NTFS.

3.1.5 Microsoft's cloud-based backend

We define as **Microsoft's cloud-based backend** the infrastructure provided by Microsoft to support an analysis about a file to be executed in the context of SAC. It is a set of servers maintained by Microsoft to provide feedback concerning the reputation of a given application, but also to control remotely the current configuration of the SAC feature, on the machine where SAC is deployed.

3.1.6 Cloud-based Analysis

We define as **Cloud-based Analysis** the notification performed by Microsoft Defender Antivirus in the context of SAC to the servers of the Microsoft's cloud-based backend for further analysis. Symbol names of functions and structures refer to this kind of notification as **reports** generated by the Microsoft Active Protection Service (abbreviated MAPS and internally referenced by the "**Spynet"**name in Microsoft's symbols). The MAPS is a general component of Microsoft Defender Antivirus, used in different contexts for different purposes, including SAC. Disregarding internal details related MAPS, we can see the report generated as a structure transferred to Microsoft's servers and containing information about different kinds of resources (executable file, but also, scripts, URL, …) to be analyzed (even if in SAC, it is about the executable file only). To maintain consistency with Microsoft's debugging symbols, we refer to reports generated by Microsoft Defender Antivirus in the context of SAC by "Synet Reports". Factually, this cloud-based analysis allowed through MAPS is an extension of the capabilities of Microsoft Defender Antivirus based on the Microsoft's cloud-based backend. In SAC, such a notification is conceptually in the spirit of the "Intelligent Security Graph" (Microsoft).

3.1.7 Pre and Post Notification

For convenience reasons while studying SAC, it makes sense to define a temporality during the analysis. As explained in sections 3.3, the SAC procedure is designed to notify Microsoft Defender Antivirus for retrieving an analysis provided by the Microsoft's cloud-based backend. In this context, we consider as **pre-notification** the period starting with the kernel notification that a file must be evaluated by SAC through a notification to the Microsoft's cloud-based backend performed by Microsoft Defender Antivirus. Similarly, we consider as **post-notification** the period starting with the answer retrieved from the cloud-analysis to the final decision held by the kernel. That way, it is possible to reference a specific context during the SAC notification procedure.

3.1.8 Eligibility of an image

We call **eligibility** the potential of a mapped image in memory to be selected for a specific kind of analysis. It references the ability for an application to be selected, based on its own characteristics, for a specific procedure. In SAC, it concerns the kind of applications which are eligible to be analyzed by SAC.

3.1.9 Smart App Control modes

The SAC feature may belong in three different modes, exclusive with each other. Theses modes define the reaction of a SAC facing an image to analyze but also the effective status of the feature (on or off). Only the "**evaluation mode**" where SAC learns about the system to know if the last is a good candidate for such a security feature is clearly defined over Microsoft's documentation (Microsoft) and the graphical interface of the Smart App Control setting in Windows Security settings. By an abuse of language based on the learning operation performed by SAC in the "evaluation mode", we call it sometimes "**learning mode**".

The two other modes correspond to the activation or deactivation of SAC as a security feature. If activated, it means that SAC has somehow transited from the evaluation mode to another mode where it does not learn about the system but where it applies the decisions allowing or refusing applications to run. This corresponds to set SAC to "on" in the Smart App Control setting in Windows Security settings. For the sake of simplicity, we propose to call this specific mode the "**enforcement mode**".

If SAC is deactivated (corresponding to "off" in the Smart App Control setting in Windows Security settings) for any reason, we consider that SAC is in the "**deactivated mode**". In this context, SAC feature does not operate anymore, meaning that the kernel does not provide any more notifications to Microsoft Defender Antivirus but also that there is no more request from Microsoft Defender Antivirus to the analysis returned by the Microsoft cloud-based backend.

3.1.10 Cryptcatsvc service

This service is used to manage cryptographic catalog files. According to the Microsoft's documentation (Microsoft), a digitally signed catalog file (*.cat*) contains a collection of cryptographic hashes, or *thumbprints*. Each thumbprint corresponds to a file that is included in the collection. Since ci.dll is vested with the signature validation enforcement, this one has an interface with the Cryptographic Catalog Service.

3.1.11 Microsoft Defender Antivirus

Microsoft Defender Antivirus is the antivirus software developed by Microsoft and deployed in Windows out-of-the-box as the default antivirus. Usually stored in "C:\Program Files\Windows Defender", the true location of the active version of Microsoft Defender Antivirus is technically referenced by the following registry value: "HKLM\SOFTWARE\Microsoft\Windows Defender\ProductAppDataPath". This software is a complex software whose architecture is far beyond the scope of this document. Nevertheless, we need to cover some of these modules for a better understanding.

As any antivirus software, Microsoft Defender Antivirus has one foot in the kernel with a set of drivers and one foot in user-mode with regular executable file and associated libraries. In the scope of the project, this is the analysis of the user-mode components of Microsoft Defender Antivirus which is relevant. Involved in the SAC feature, there are five main components.

- MsMpEng.exe: this is the antivirus' main application. This one is a regular service holding the different modules and libraries attached to the antivirus. This is the application which is automatically launched as a service to interface with the drivers' notifications for analysis.
- MpSvc.dll: This library is the central part of the antivirus, holding the general configuration of the antivirus service at initialization time and then housing the main analysis functions. This library is responsible to perform the heavy job concerning analysis of and kind of resources submitted to Microsoft Defender Antivirus.

- MpClient.dll: This library has two sides, and it represents the client interface of the antivirus. At first, this DLL is mandatory for a process that would like to directly interface with Microsoft Defender Antivirus. This one may load a specific configuration per process which would allow a "per requesting process" analysis configuration. But also, MpClient.dll is directly loaded and it is used in Microsoft Defender Antivirus as a support in some analysis operations.
- MpEngine.dll: Potentially stored in another location than the active directory of Microsoft Defender Antivirus (location of this library is resolved at runtime), this DLL is responsible – among other things – to coordinates what is internally called "signals" but which in the end corresponds to central tasks performed by the antivirus, especially the analysis of submitted resources.

3.1.12 WinHTTP framework

Microsoft Windows HTTP Services (usually abbreviated WinHTTP) (Microsoft) is an API from Windows used to provide a high-level interface to the HTTP Internet protocol. It is designed for server applications, meaning there is no interactions expected from a potential user, in contrast to Windows Internet (usually abbreviated WinINet) (Microsoft) which is also an API providing a high-level interface to access HTTP Internet protocol. The WinHTTP framework serves as the foundation for building networked applications on the Windows platform, offering essential features such as customizable timeouts and secure communication. In SAC, this API is used to establish the communication with the Microsoft's cloud-based backend.

3.2 Evolution of the notion of code integrity check over the different versions of Windows

The notion of code integrity check is a long evolvement over the different versions of Windows. Aiming to protect personal and corporate data, the goal is to control tightly which program is allowed to be executed in the system, based on different criteria. Containing different features over the different versions of Windows, SAC is the ultimate piece of the effort made by Microsoft to ensure that legitimate code runs in their customer's systems.

At the beginning, starting with Windows Vista and only for 64-bit versions of Windows (and subsequent versions), drivers must be digitally signed (Microsoft). This was a strong requirement introduced in Windows Vista to prevent malware authors to develop their own drivers, most of the time for rootkit purposes. It means that any unsigned kernel-mode software will not load and will not run on x64-based systems. At that time, it was necessary to obtain what Microsoft called a "Software Publishing Certificate (SPC)" from a Certificate Authority (CA) accredited by Microsoft. The kernel of the operating system checks the digital signature embedded in the driver file (mandatory for boot driver) or in the catalog file associated with the driver. If the signature is valid and the certificate root chain is correctly validated by the kernel, the driver can be loaded.

As a side note, regarding kernel security introduced by Windows Vista, the notion of code integrity covers what was called Kernel Patch Protection (KPP) (Microsoft), informally known as "PatchGuard". This security aims to avoid any malicious manipulation of the kernel integrity (integrity of the kernel code and main data structures instantiated in memory) without being perfect (Luc Reginato).

The signature check security (which, at first, only concerned 64-bits drivers) has been extended to user-mode applications but with a different logic. With the kernel code integrity check, there was nothing configurable. Users must accept that only signed drivers are loadable, not matter which driver it is since the last is signed. Since Windows 7 (Enterprise and Ultimate versions), this procedure is enhanced by configurable security checks calling AppLocker, validating if an application is allowed to run on a given system. AppLocker helps to control which apps and files users can run (Microsoft) and it can be configured over Group Policies (GPO). This control operates over executable files, scripts, Windows Installer files, dynamic-link libraries (DLLs), packaged apps, and packaged app installers.

AppLocker has been designed to prevent end-users from running unapproved software on their computers. But this feature does not meet the servicing criteria for being a security feature, as defined by the Microsoft Security Response Center (MSRC) (Microsoft). This is why Microsoft introduced with Windows 10, in addition to AppLocker, the WDAC to allow organizations to control which drivers and applications are allowed to run on Windows, based on rules. With these rules, it is possible to define which program (applications, drivers, DLLs, scripts, …) is allowed to be executed based on different criteria (code signing certificates, applications' attributes, reputation, installer, and launcher filiation, …), extending possibilities provided by AppLocker. Note that both technologies (AppLocker and the WDAC) can, and they should be used in conjunction, especially in an environment where different versions of the Windows operating system operate.

In parallel to the WDAC, Windows 10 (and higher) proposes authorize of the execution of an application based on its reputation, with the Intelligent Security Graph (ISG). Defining an application control policy can be a complex task for organizations, especially for those which have a limited control over their applications ecosystem (Microsoft). Hence, it is possible to configure the WDAC to automatically allow applications that Microsoft's ISG knows as having a good reputation (Microsoft). Such an evaluation is performed through an internet request to the servers of Microsoft. This feature is highly limited and not recommended for binaries involved in the boot of the system (since no internet connection may not be available and ISG is not driver applicable) or business critical (since Microsoft might not be aware of them). The reputation of an application is based on feedback provided by "trillions of signals collected Windows endpoints" (such as power Microsoft Defender SmartScreen and Microsoft Defender Antivirus) or other sources "internal to Microsoft"

(Microsoft). These signals are processed over machine learning analytics to help classify applications as having "known good", "known bad", or "unknown" reputation. Since this feature is based on machine learning and, according to Microsoft, on a day-to-day updated input data, the reputation of an application may change from one day to the next.

Aside application execution control, Windows 10 (and higher) provides also a code integrity feature relying on virtualization-based security (VBS) (Microsoft) to improve the protection of the kernel integrity and more generally exploits in the Windows kernel (Microsoft). Called hypervisor-protected code integrity (HVCI) or hypervisor enforced code integrity, this feature originally written in Device Guard is today called “Memory integrity” (Microsoft). This one ensures that the kernel of Windows is not altered in a malicious way, in addition to critical structures. Also, based on hypervisor technology, there are extra protections aiming to limit potential exploitation of vulnerabilities.

Starting with Windows 11 version 22H2, a new feature called Smart App Control has been introduced. This feature is directly based on the WDAC, as explained by Microsoft (Microsoft). This feature is only available on clean installation of Windows 11 to start in evaluation mode[2]. In this mode, the feature learns for the system which applications are executed and it the system is a good candidate to be protected by SAC. To make “confident prediction about its safety”, each application is evaluated before running (Microsoft). In this mode, all applications are in the end allowed to be executed. But after a certain period in evaluation, SAC switches in enforced or deactivated mode, since the learning procedure has decided if the system is a good candidate for SAC. In the enforced mode, an unknown or untrusted application is going to be effectively blocked on the system. Otherwise, if SAC is deactivated, SAC is not notified for analysis purpose, and it does not provide any protection. But other WDAC/AppLocker policies and rules still apply. To avoid any gap in the continuity of the security, it is not possible to reactivate SAC once the last has been deactivated, except by resetting to zero Windows 11.

Starting with Windows Vista, the notion of code integrity has been evolved in different context, from the self-protection of the system with unconfigurable code integrity to the configurable code integrity provided through the different features carried out first by AppLocker and then the WDAC. Concerning the WDAC, features such as ISG has been designed on top of it, same with SAC that is directly based on WDAC. In these last two cases, there is a strong emphasis on the user experience, allowing to simplify the configuration and the use of advanced security features such as the WDAC.

In the Figure 2, we represent the main steps one can consider in Code integrity over the different versions of Windows.

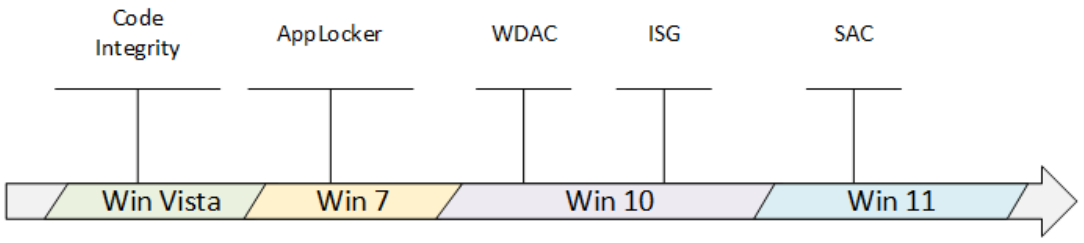

Figure 2. Historical timeline of the features taking part in Code Integrity over the different versions of Windows.

[2] Provided that the sending of optional diagnostic data is allowed, which is the case in the default configuration.

3.3 Execution Flow of Smart App Control

Creating a process in Windows is a several stages procedure carried out between the kernel of the operating system and different user-mode components (mainly the API involved in calling the `CreateProcess` or the `StartService` functions). If the creation of a process asked by a user is most of the time initiated by the user-mode API call, the heavy work is performed in kernel-mode.

In the kernel of Windows (ntoskrnl.exe), a new process is usually related to the `NtCreateUserProcess` function. This one is a several stages procedure (Yosifovich) resulting in the creation of a new environment able to allow the execution of the executable file. Part of this procedure, in between opening an executable file to create a section object (that is to say, a memory mapped file object from user-land's point of view) and the initialization of the EPROCESS and subsequent structures, there is a code integrity step. The transition point from the regular process creation procedure to the code integrity assessment is performed within the `CiValidateImageHeader` function.

The validation of an image in code integrity is also a serval stages procedure. This procedure is distributed among several functions and subfunctions called for different purposes, mainly in the WDAC. In Figure 3, we represent the main stages of the SAC procedure, and the following list resumes the main stages associated. The operations performed in each stage are described with more detail in the subsequent subsections.

1. Check which kind of image mapped in kernel-mode memory is eligible to an evaluation (driver, script, application, signed, unsigned, ….).
2. Extract (the signature) and compute (the hash) information from the executable file mapped in memory to perform integrity checks.
3. Apply the WDAC policies to the image. It means checking if a rule defined in a policy matches the loaded image.

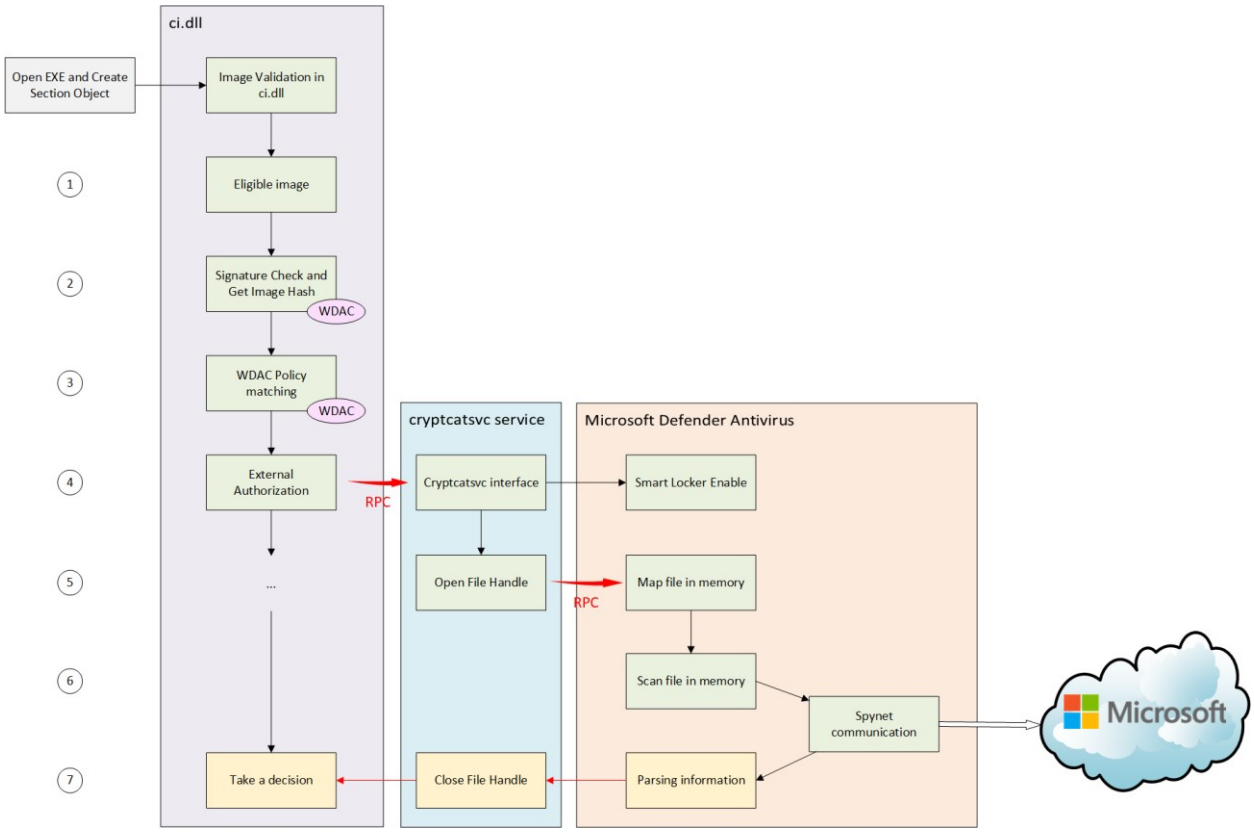

Figure 3. Overview of the main steps of the execution flow of SAC.

4. Part of the WDAC policy evaluation, there is an external evaluation assessment which is requested by ci.dll to user-mode components. The transition from kernel to user-mode is performed with RPC

notification to the cryptcatsvc service. The last is responsible first to enable the "Smart Locker" feature in Microsoft Defender Antivirus, if it has not been done before.

5. The loading executable file is mapped into user-mode memory for further analysis. This procedure is performed in several steps, including an RPC notification to Microsoft Defender Antivirus.
6. The scan of the file and the notification to the Microsoft's cloud-based backend for evaluation is performed within Microsoft Defender Antivirus.
7. Once the cloud-based analysis has been performed and returned to the Microsoft Defender Antivirus, the answer is provided back to the kernel so that a decision is taken based on the analysis run by Microsoft Defender Antivirus and especially the cloud-based analysis.

The procedure described in Figure 3 represents the main steps when a notification from the kernel to Microsoft Defender Antivirus is performed. Once the cloud-based analysis has been provided back to Microsoft Defender Antivirus, the answer follows the trace back of the function call stack until reaching back the kernel in ci.dll. In this context, a decision is taken based on the cloud-based analysis and the SAC current mode (evaluation or enforcement) to decide whether the process must be executed or not.

3.3.1 Eligible image for SAC analysis

In the WDAC, it matters to know which kind of evaluation is given to a specific image. For instance, a driver requires every time to be digitally signed to be loaded while an executable file does not. Based on the file extension and internal flags set at compilation time (such as the /INTEGRITYCHECK compilation option set in Visual Studio (Microsoft)), an executable file (.sys, .dll, .exe, or .msi) or a script (.js, .ps1, .vbs, …) is selected for specific evaluation in WDAC. Based on Windows' debug symbols, this notion is sometimes called "action for image" evaluation, especially in the `CiGetActionsForImage` function.

The Table 1 represents the different kind of files evaluated during the creation of a process. It represents the files provided to the `CiValidateImageHeader` function in a process creation. This is relevant to understand which kind of image related to Code Integrity are susceptible to be eligible for SAC analysis. The Table 1 is the result of an analysis performed with a debugger in Windows 11. The tests were performed with Windows 11 "out-of-the-box" (which means Windows 11 in default configuration) with SAC in evaluation mode and not specific WDAC policy enforcement setup. More information is available in Annex A about the experiment resulting in observations provided in Table 1.

	Signature Type				
Image Type		Microsoft Signed	Trusted third-party signature	Untrusted third-party signature	Unsigned
	Application	X	X	X	X
	Script				
	Driver	X	X	X	X
	Installer	X	X	X	X

Table 1. Matrix of code integrity check notification based on Image and Signature Types.

After an internal procedure of selection to know which kinds of file are eligible for analysis, the final set of files sent to SAC is different. SAC only concerns a small subset of notifications. For short, the code integrity evaluation procedure can take care of a lot of cases by itself. This is specifically the case for drivers which are only evaluated in kernel-mode without requesting any further assistance from Microsoft Defender Antivirus in this context (which does not prevent the regular Microsoft Defender Antivirus filtering mechanism in kernel mode to perform its own notification by itself). In Table 2, we represent the different types of files notified to SAC through ci.dll. The empirical observations are performed by checking which image is notified to the `SSCatDBSmartlockerDefenderCheck2` function. The experimental environment is the same as the one used to make the observations given by the Table 1.

	Signature Type				
Image Type		Microsoft Signed	Trusted third-party signature	Untrusted third-party signature	Unsigned
	Application		X	X	X
	Script				
	Driver				
	Installer	X	X	X	X

Table 2. Matrix of SAC notification based on Image and Signature Types.

Then, an internal cache evaluation is performed. Dealing with image eligibility, the `CipValidateFileInCache` functions is called. This function is mainly based on a Kernel Extended Attribute (see section 3.1.4) called "$Kernel.Purge.ESBCache" (in some specific version of Windows, the name could also be "$Kernel.Purge.CIpCache"). This Kernel EA holds a structure representing different information already checks for a given file, avoiding reevaluating them. We also note there is a specific mechanism concerning an expiration timestamp set in the Kernel EA to reevaluate periodically the content of the cache data stored. The validity period is retrieved in the `CipGetFileCache` function, parsed with the `CipParseFileCache` function, and evaluated in the `CipVerifyFileCache` function, in comparison with the current time when the WDAC evaluation is performed.

Also, part of the WDAC policy matching procedure (in the `CipApplySIPolicyUMCI` function), the Kernel EA called "$Kernel.Purge.TrustClaim" is checked. If this one is correctly setup, the verification is skipped. Further information about this specific Kernel EA (which is automatically removed if the associated file is modified) can be found in (Graeber).

3.3.2 Signature check and image hash generation

This part is a preparation for the WDAC evaluation. It aims to parse the MZ-PE file in memory and to extract the digital signature of the file if there is one. The principle of signature extraction respects the Windows Authenticode Portable Executable Signature Format as explained in section 3.1.3. In case of a digital signature would be present, this one is checked by computing its hash integrity. This procedure is clearly a preparation step for the WDAC evaluation performed thereafter.

In specific circumstances (if the machine supports VBS and VBS is activated), the signature extraction and verification can be performed in an isolated user mode (IUM) (Microsoft) procedure. Based on the VTL provided by the Hyper-V Hypervisor, this check involves HVCI capacities to perform the signature integrity check.

Once the signature extraction has been done and checked, there is a notification to the User Mode Code Integrity (UMCI). The user-mode in this context should be interpreted as a check for user-mode component but not especially performed in the user-mode. This is the kernel-mode `CipApplySIPolicyUMCI` function which is called to start the WDAC policy evaluation.

3.3.3 WDAC Policy Matching

After having extracted signature and other relevant information from the evaluated image, the WDAC applies in a loop all the "signature verification" procedures. The iterate procedure from one signature verification to another one is performed via a direct call to the `CiGetNextSignatureVerification` function.

The signature verification is performed in the `CipApplySiPolicyEx` function. The WDAC engine is notified to validate the policies provided in the context of the signature verification. Part of the verification procedure is driven by the `CipExternalAuthorizationCallback` function which is used as a callback function. This callback function is called part of the policy validation to perform a SAC notification.

3.3.4 External authorization and SAC initialization

The callback function called in the context of the WDAC policy is responsible – in the end – to notify the user-mode component involved in the analysis of SAC. This operation is performed by interfacing with the RPC interface provided by the cryptcatsvc service (RPC UUID {f50aac00-c7f3-428e-a022-a6b71bfb9d43}). This service is a proxy in between ci.dll and Microsoft Defender Antivirus. Since ci.dll is naturally interfaced with cryptcatsvc service for signature verification purpose (especially to query the chain of certificates for a given signature), Microsoft just added a "SmartLocker" interface accessible via RPC to this service, enhancing the original architecture of code integrity check with SAC.

The SmartLocker RPC interface allows to interface with the SAC mode (enforce, evaluation, deactivate) in addition to submit an analysis request to SAC. Such a request is called a "Defender Check". For a Microsoft Defender Antivirus check, the operation is twofold. On the first hand, such an operation starts by interfacing with Microsoft Defender Antivirus and to initialize it if it has not been performed before. Under the wood, the MpClient.dll library is loaded, and then internal functions are called to initiate the SAC feature. This is the `MpSmartLockerEnable` function which is responsible of this action.

The initialization is performed once for all, meaning that a recall to the initialization procedure is a transparent operation. The initialization procedure is complex, and it involves a lot of internal implementation details in Microsoft Defender Antivirus. This procedure checks different points about the current operating system version and if Microsoft Defender Antivirus is up to date, in addition to load many configuration values, including for instance the quarantine location directory path. Then, the procedure ensures that the RPC interface dedicated to SAC is correctly setup and in case of, it instantiates it. That way, it also checks that the calling process is trusted, checking for instance that the calling process is running in the context of the user whose SID is "S-1-5-80-242729624-280608522-2219052887-3187409060-2225943459". For security purpose, SAC initialization ensures that the process trying to interface with Microsoft Defender Antivirus is registered as a Protected Process Light (PPL).

3.3.5 File mapping operation

This is the cryptcatsvc service which opens the file to analyze. Even if the file has already been mapped in kernel-memory, it is necessary to remap it into user-mode memory for analysis purpose in the context of Microsoft Defender Antivirus. The analysis of the file by Microsoft Defender Antivirus will be performed in the memory context of cryptcatsvc service, meaning the parsing operations are performed remotely by Microsoft Defender Antivirus.

The transition between the cryptcatsvc service and Microsoft Defender Antivirus engine for analysis is performed through RPC context, more specifically with the `MpCheckMappedFileTrust` function from MpClient.dll. The RPC interface of Microsoft Defender Antivirus is referenced with the {C503F532-443A-4C69-8300-CC0D10FBDB3839} GUID. This RPC interface proposes a full list of functions allowing to perform different kind of operations on Windows. Among other things, it is possible to request for getting the threat history, getting the quarantine content, updating the signature engine, generating Spynet analysis report, querying the default folder guard list, asking for a network capture, and demanding a scan for analysis (with many possibilities such as fast analysis, in memory, offline, …).

3.3.6 Scan analysis in Microsoft Defender Antivirus for SAC

The analysis is started by a scan notification performed from MpClient.dll to MpSvc.dll through the RPC interface exported by Microsoft Defender Antivirus. The name of the RPC function in MpSvc.dll used to perform a SAC analysis is the `ServerMpRpcMemoryScanStart` function.

The implementation relies on instantiating a "memory scan engine VFZ", with no specific information about the meaning of VFZ. This is the "`MpService::CMpMemScanEngineVfz`" class which rules this engine providing generic capabilities (URL scanning, Amsi, Executable files). That way, the same engine can provide

different capabilities to analyze different kind of potential threats. In SAC, this engine is only used to analyze executable file, showing the "normal analysis flow" taken by SAC.

The analysis relies on a complex system of internal notifications, due principally to a system of thread pool used to manage a request to a "memory scan context". This engine is a generic engine configured to analyze any kind of resource (executable, script, or URL to scan). Based on our observations, with SAC notified from the kernel, the analysis first concerns user-mode executable code. This engine is based on a thread pool used to optimize the scheduling of the different scan operations to be performed across multiple threads.

The internals of the scanning operation are directly related to Microsoft Defender Antivirus analysis procedure, which is out of topic concerning SAC. The most relevant point is the cloud-based analysis notification which provides information to Microsoft about the analyzed file. The pieces of information provided are threefold. Firstly, there is a set of elements extracted or computed from the file to analyze. For instance, this concerns the hash of some sections of the file, headers from MZ-PE format, and signature of the file is there is any (more details in section 4.1.5.2). Secondly, there is set of metadata related to the file such as the file path of the file or modification timestamps. Thirdly, there is a set of information concerning the Microsoft Defender Antivirus's configuration (version number but also specific configuration values set) and the system in general (version of the operating system). All this information is transferred to be used in the context of the cloud-based analysis. This operation is performed through a secure HTTPS connection using "Bond" content type to encapsulate data.

3.3.7 Post-analysis Notification

3.3.7.1 Feedback procedure

Once the Microsoft cloud-based backend has provided back its analysis, this is Microsoft Defender Antivirus which receives the answer. This is the answer to the HTTPS request previously sent. The answer is parsed to extract information, for instance in the `MpService::IMpSpyNetReportContext::HandleResponse` function from MpSvc.dll. Once this parsing operation has been performed, it is possible to differentiate in the answer at least three kinds of answers.

- An answer concerning the result of the cloud-based analysis about the current file submitted. A scoring value used to describe the "reputation" of the file, according to the cloud-based analysis.
- An answer concerning the reference of the submission performed, especially internal values which can be considered as a "tracking index" regarding the submitted file. This ensure that the answer is correctly related to the original submission but
- An optional answer concerning the status of SAC in the client, especially to switch it from evaluation to enforcement or deactivated mode.

The first kind of answer is directly used to provide a hint for the final decision taken in the end by the kernel. The second kind is used for tracking operations. And the last one is used to configure remotely and automatically the status of SAC.

All this information is packed into a specific structure ready to be returned to the kernel which originally performed the RPC call. All these parsing and packing operations are based on different procedures which are common to any analysis in Microsoft Defender Antivirus, especially using thread pool technology for optimization purpose.

The transition from Microsoft Defender Antivirus in user-mode to ci.dll in kernel-mode is quite direct since the original notification for scan has been performed within an RPC function call. Back in kernel-mode, in ci.dll, the procedure extract information from the cloud-based analysis forwarded to the kernel. There is a scoring value associated with the file originally submitted to the scan which is used. This scoring value is composed also by different flag values which will be used in the end to explicit the reason of a potential denying of execution. In conjunction to a set of conditions relative to the extensions files possibly submitted to SAC, a decision is taken regarding the current file. Either the evaluation does not consider the file has dangerous and the execution is allowed (even if further WDAC policies are taken into consideration for the

final choice), either the evaluation negatively consider the file and the result depends on the current mode of SAC. Either SAC is in evaluation mode and in such a case only a WNF toast is displayed to user's eyes (in addition to some logging capabilities and a kernel debug print if a debugger has been previously attached in the `SIPolicyProcessDbgSettingAndReprieve` function in ci.dll). In the end, the `CiValidateImageHeader` function returns a status value resulting in the execution if the returned value is not an error value, otherwise a deny of execution if the returned value is an error value.

In the response returned by the cloud-based analysis and transferred to the kernel, it is possible to switch the mode in which SAC belongs. Part of the post-analysis operation in the kernel, there is a check to know if such a specific order has been provided. In case of, this switching mode operation will trigger the `CiAsyncPolicyRefreshRoutine` function in kernel mode, resulting in an RPC call to the `s_SSCatDBSendSmartQppControlSwitchEnforceToast` function in crytcatsvc service. The transition is based on a call to the `WldpSendSmartAppControlSwitchEnforceToast` function from the Wldp.dll. Behind the stage, this is the "*Windows Push Notification Platform*" API which is used to proceed (Microsoft). This last consideration illustrates the remote control that the cloud-based analysis has on SAC for each computer.

The procedure of the feedback from Microsoft Defender Antivirus to the Code Integrity check module in the kernel is illustrated in Figure 4. It is worth noting here the different kinds of messages sent at once from Microsoft Defender Antivirus to the kernel. In one case, the feedback is used to take a decision about the possible execution of the given image notified to the system. On the other case, the feedback is used to switch the current mode of SAC (from evaluation to enforced or deactivated).

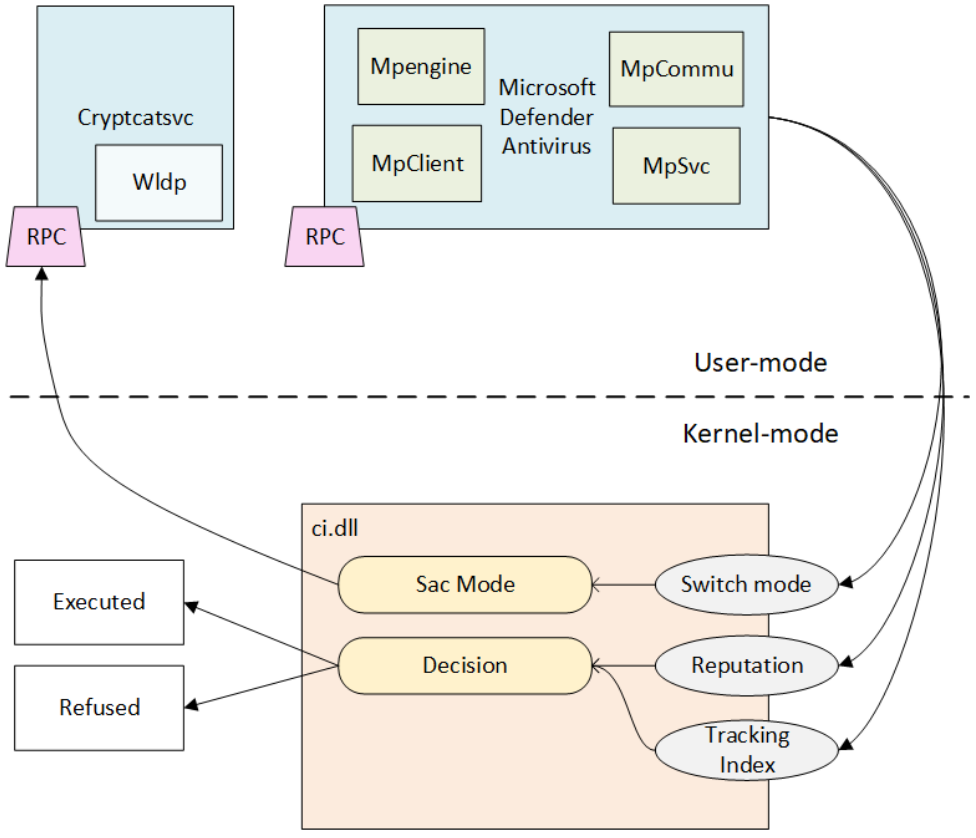

Figure 4. Feedback of the cloud-based analysis loop providing different kind of answers to ci.dll.

3.3.7.2 Enforcement mode consequences

In the case where SAC would be switched to enforcement mode, the main logic of the security feature will be like the one described previously. The only difference relies in the post-notification where a blocking decision may be taken depending on the feedback provided by Microsoft Defender Antivirus which needs Internet to operate. It raises the question of the lack of internet connection. Without Internet, the result is clear: any analyzed file will be blocked. The reason relies in the fact that Microsoft explains to block files whose reputation is known to be malicious or unknown. Without any internet connection, the reputation of the file remains unknown.

But it is also relevant to note that not all the executable files in the machine about to be executed will be blocked without internet connection. The operating system must remain functional, meaning that all executable files signed by Microsoft are always considered as valid (since they are never notified for cloud-based analysis). Same for drivers or scripts. Only MSI installers may be impacted, like executable files non signed by Microsoft. Nevertheless, in this last case, one further important clarification is in order.

In the kernel-code integrity check procedure, there is a cache check which is performed before an executable file is about to be notified. This specific procedure concerns the `CipValidateFileInCache` functions which reads a kernel-EA, looking for a potential former analysis result that would have been performed before by the system. If such a cache is associated with a file, the system directly reuses it, without notifying again Microsoft Defender Antivirus for a cloud-based analysis. This specific optimization mechanism has two direct consequences.

At first, because the optimization avoids a cloud-based analysis, the lack of internet connection is not a problem for already analyzed files. At least, for a certain amount of time[3] since the kernel-EA cache associated to the file is considered as valid.

The second consequence of this optimization mechanism is the smooth transition between the learning mode and the enforcement mode. There is no re-evaluation of already evaluated files when switching to enforcement mode – at least since the cache associated to a file is still considered as valid. Supposing there were few (and possibly no) executable files considered with an insufficient reputation to be executed, it means that all existing applications executed on the system are known to be good by the cloud-based analysis. It could also mean that when the cloud-based analysis switch to enforcement mode for a given machine, there

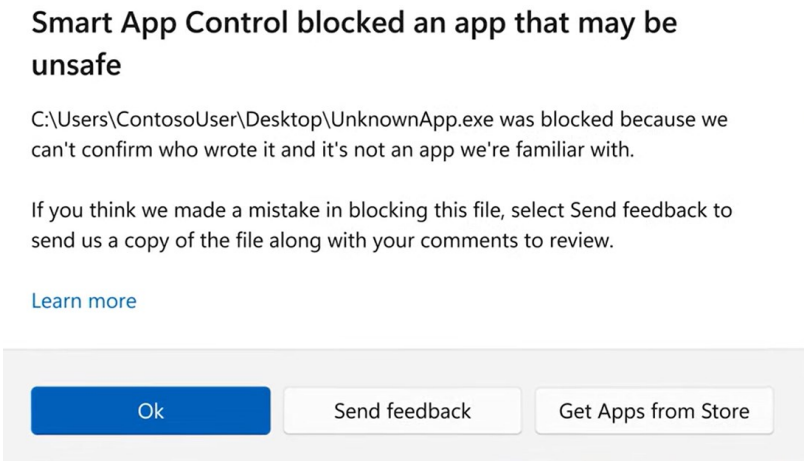

Figure 5. Illustration of the message displayed to the user's eyes in case of SAC blocking operation - extracted from Microsoft's promotion videos (Microsoft).

[3] This validity period is not directly hardcoded in the kernel, but it depends on WDAC's configuration. Details about this point is out of the scope of this study, but a reference to this value is set in the Kernel EA associated to the file.

is a measure at Microsoft's side that the impact of the enforcement mode should be low (or null) for the user experience.

In the case where the user would face the situation of a blocked file when SAC is in enforcement mode, in addition to the execution file being blocked by the kernel, there is a specific GUI Window displayed to user's eyes. This one is provided in Figure 5. In such a case, it is possible to send feedback to Microsoft and try to get an equivalent application from the Microsoft Store of Apps.

4 Technical Analysis of Functionalities

The present section highlights relevant technical points concerning the Smart App Control feature in Windows 11. It is worth noting that only the most relevant specific points of the feature are detailed in this part. The reason is twofold. On the first hand, SAC is a highly fragmented feature, dispatched over different modules and relying on different technology. Behind the wood, it is most a construction of different pieces of technology linked together than a standalone feature. On the other hand, if the internal implementation details of SAC are interesting, they are probably not as important as the key points on which rely SAC. Especially, the implementation details (class instantiation, thread pool management, internal data structures, …) of in between parts of SAC are ignored here. This is the reason why the following subsections are mainly focused on critical points of SAC, trying to provide a technical background to propose a reliable analysis of SAC and relevant questions concerning it.

In this context, the present section relies mainly on the regular SAC notification activity, meaning that a file must be analyzed. To cover the most generic case (and probably the most common one), we propose an analysis which follows the notification by the kernel to an unsigned executable file (.exe) executed by an authenticated user. Other possibilities may have been considered (digitally signed executable, installer), but without any loss of generalities, the general execution flow of the SAC feature remains the same.

4.1 Smart App Control pre-analysis part

4.1.1 Analysis of CipExternalAuthorizationCallback function

The `CipExternalAuthorizationCallback` function is definitively central in the procedure resulting in a SAC notification. The internal logic of the `CipExternalAuthorizationCallback` function is complex since it evolves WDAC policy objects and a lot of global flags in the system. We propose to describe the function with its main steps concerning the SAC notification.

Af first, there is an evaluation of the Smart Locker procedure by calling the `CipCheckSmartlockerEAandProcessToken` function. This function queries on Kernel EA concerning Smart Locker (accessed through `CipSmartlockerGetValidEA` function) if there is some, and Security Attributes Token (SAT) which is an internal artifact of Authz[4] (Microsoft). In the last case, this is the function `SeQuerySecurityAttributesToken` which is used to query the two attributes "SMARTLOCKER://ORIGINCLAIM" and “SMARTLOCKER://SMARTSCREENORIGINCLAIMNOTINHERITED”. Thereafter, there are different checks performed through the `CipCheckForExtensionAgainstList` function to know if the extension in the file name of the targeted file mapped into memory could be relevant. By relevant, it means that the extension could correspond to one defined in the “Dangerous Extensions” list (available in Annex C) or an “Installer Extensions” (.msi). Regarding an unsigned executable file with no pre-existing kernel-EA, only the installer extension is checked. In either case, there is a call to the `CiCatDbSmartlockerDefenderCheck` function. This last call is definitively the most important one since it directly notifies Microsoft Defender Antivirus for direct analysis.

The main point of the `CipExternalAuthorizationCallback` function is the call to the `CiCatDbSmartlockerDefenderCheck` function. This function is directly responsible to notify the Microsoft Defender Antivirus with an RPC for SAC analysis. The remaining of the callback is about managing the post-notification of this last call. Then, there is an evaluation to check if the RPC call has succeeded or not, followed, if necessary, by a cache update associated with the validated file (and only for validated files) thanks to a call to the `CipSetFileCache` function.

[4] From Microsoft documentation (Microsoft), this API allows applications to cache access checks for improved performance.

4.1.2 RPC notification with CiCatDbSmartlockerDefenderCheck function

Whenever it comes from the `CipCheckSmartlockerEAandProcessToken` or directly from the `CiCatDbSmartlockerDefenderCheck` functions, the transition from the kernel-mode to Microsoft Defender Antivirus's service, is performed within the `CiCatDbSmartlockerDefenderCheck` function. In this last function's name taken from the Windows' symbols (Microsoft), we note the transition from the notion of "Smart Locker" to the "Defender" one, highlighting the different internal names of "Smart App Control" from the kernel point of view.

Mainly, there is a call to the `CipCatDbRpcConnect` function. The last one creates a bind RPC endpoint called "keysvc" with the use of the `RpcBindingCreateW` function. The Authentication-Level set in the binding procedure is RPC_C_AUTHN_LEVEL_PKT, meaning that all data received is from the expected client, without validating the data itself. The RPC interface ID targeted is the UUID {f50aac00-c7f3-428e-a022a6b71bfb9d43}. Subsequently, regular operations of synchronization are performed with `KeWaitForSingleObject`, `RpcAsyncCancelCall`, and `RpcAsyncGetCallStatus` functions on the RPC handle, waiting for the answer from Microsoft Defender Antivirus.

After the `CipCatDbRpcConnect` function call, the `CiCatDbSmartlockerDefenderCheck` function uses an endless loop to perform a call to `RpcAsyncInitializeHandle` and `SSCatDBSmartlockerDefenderCheck2` functions. This last function is responsible to directly notify Microsoft Defender Antivirus through the RPC interface. Once the RPC call notification is performed, there is a wait operation for Microsoft Defender Antivirus's answer for a certain amount of time. This time internal is based on the policy whose GUID is {283AC0F-49AE-FFF1-938A-A1ADD6CA3031} and retrieved by the `SIPolicyIsPolicyActive` function. The wait operation is performed with a call to the `KeWaitForSingleObject` function, as a regular kernel-mode wait. If the wait operation is issued before the RPC respond (RPC could be RPC_NT_SERVER_TOO_BUSY), there is a delay issued to wait for a less busy time for RPC analysis (with the use of the `KeDelayExecutionThread` function).

In case of success, the end of the procedure is about to extract the number of "answers" returned by Microsoft Defender Antivirus and to parse the content of these answers. In addition, the total time elapsed for the operation is kept, likely for log and statistical purposes.

4.1.3 From kernel RPC to Microsoft Defender Antivirus

4.1.3.1 Identification of the RPC server and the RPC routine called

The RPC notification from the kernel to the SAC interface in Microsoft Defender Antivirus is done by the `SSCatDBSmartlockerDefenderCheck2` function. This function is nothing but a call to the `Ndr64AsyncClientCall` function. This last one is used to transfer an RPC client call to an RPC server interface.

The targeted RPC server is the one registered under the {f50aac00-c7f3-428e-a022-a6b71bfb9d43} GUID. Using the RpvView tool[5] to find that GUID, it is possible to identify the service housing the RPC server as "C:\windows\system32\svchost.exe -k NetworkService -p -s CryptSvc". The endpoint is identified as

[5] https://github.com/silverf0x/RpcView

"keysvc", confirming the binding procedure observed in section 4.1.2. Observation of the name of the DLL related to the cryptcatsvc is given in Figure 6, identifying C:\Windows\System32\cryptcatsvc.dll.

Pid	Uuid	Ver	Type	Procs	Stub	Callba	Name	Base	Location	Flags	Description
4004	0d72a7d4-6148-11d1-b4aa-00c04fb66ea0	1.0	RPC	1	Inter...	0x00...		0x00007ff8ddb70000	C:\Windows\System32\cryptsvc.dll	0x21	Cryptographic Services
4004	1495a2be-b7a8-4299-9d3b-8825e5bcbfb9	1.0	RPC	1	Inter...	0x00...		0x00007ff8dd730000	C:\Windows\System32\webauthn.dll	0x21	Web Authentication
4004	18f70770-8e64-11cf-9af1-0020af6e72f4	0.0	RPC	5	Inter...	0x00...		0x00007ff8faf40000	C:\Windows\System32\combase.dll	0x21	Microsoft COM for Windows
4004	2579ff35-0ab0-4e5a-88fa-1d88c4e0cb92	2.0	RPC	5	Inter...	0x00...		0x00007ff8dd7f0000	C:\Windows\System32\crypttpmeksvc.dll	0x21	Cryptographic TPM Endorsement Key Services
4004	f50aac00-c7f3-428e-a022-a6b71bfb9d43	2.0	RPC	8	Inter...	0x00...		0x00007ff8dd7c0000	C:\Windows\System32\cryptcatsvc.dll	0x21	Cryptographic Catalog Services

Figure 6. Screenshot from RpcView identifying the RPC server interface used between ci.dll and Microsoft Defender Antivirus in the context of a SAC notification.

The cryptcatsvc is a Dll used as a kind of service to provide an interface (through exported functions of RPC server interface) to manage cryptographic catalog files. According to the Microsoft's documentation (Microsoft), a digitally signed catalog file (*.cat*) contains a collection of cryptographic hashes, or *thumbprints*. Each thumbprint corresponds to a file that is included in the collection. Since ci.dll is vested with the signature validation enforcement, there is a logic to connect it with the Cryptographic Catalog Service.

The RPC function targeted by the `Ndr64AsyncClientCall` function can be identified with the "nProcNum" parameter. This one corresponds to the index referencing the function exported by the RPC server in its Dispatch Table.

4.1.3.2 Initialization of the Crypcatsvc service and RPC server interface setup

In Windows, cryptcatsvc is a library originally used to provide digitally signed catalog files capabilities. During the initialization procedure linked to that DLL, there is in the `CatalogStart` function, a security descriptor setup. This one is built from a string and converted in a security descriptor with the `ConvertStringSecurityDescriptorToSecurityDescriptorW` function. The ACE strings representing this security descriptor is:

"D:(A;;GRGWGX;;;WD)(A;;GRGWGX;;;RC)(A;;GA;;;BA)(A;;GA;;;OW)(A;;GR;;;AC)(A;;GR;;;)"

The previous ACE string can be interpreted as given in Table 3.

Object's Owner.	Associated Rights
Everyone	AccessAllowed (GenericExecute, GenericRead, GenericWrite)
OWNER RIGHTS	AccessAllowed (GenericAll)
NT AUTHORITY\RESTRICTED	AccessAllowed (GenericExecute, GenericRead, GenericWrite)
BUILTIN\Administrators	AccessAllowed (GenericAll)
APPLICATION PACKAGE AUTHORITY\ALL APPLICATION PACKAGES	AccessAllowed (GenericRead)
S-1-15-3-1024-3203351429-2120443784-2872670797-1918958302-2829055647-4275794519-765664414-2751773334	AccessAllowed (GenericRead)

Table 3. Translation of the ACE associated to the RPC in cryptcatsvc.dll.

This security descriptor is open widely open to any client, which makes sense since it is supposed to be used for Catalog validation purposes – a need potentially shared by everyone in the system. This security descriptor is directly associated with the RPC server thanks to the use of `RpcServerRegisterIf3` function. This one takes several parameters, indicating that the RPC accepts as much as possible clients (RPC_C_LISTEN_MAX_CALLS_DEFAULT), it is automatically listening (RPC_IF_AUTOLISTEN) and the RPC

runtime rejects calls made by remote clients (RPC_IF_ALLOW_LOCAL_ONLY). In addition, the security-callback function `CryptSvcSecurityCallback` provided with the `RpcServerRegisterIf3` function only accepts local clients with the use of the `I_RpcBindingIsClientLocal` function. But the most interesting parameter provided to the `RpcServerRegisterIf3` concerns its RPC_SERVER_INTERFACE, illustrated in Figure 7.

The Figure 7 illustrates this RPC interface where it is possible to observe (indicated in red) the GUID of the

```
; RPC_SERVER_INTERFACE RPC_IfSpec
RPC_IfSpec      dd 60h                  ; Length
                                        ; DATA XREF: HEADER:000000018000023C↑o
                                        ; CatalogStart+F9↑o ...
                dd 0F50AAC00h           ; InterfaceId.SyntaxGUID.Data1
                dw 0C7F3h               ; InterfaceId.SyntaxGUID.Data2
                dw 428Eh                ; InterfaceId.SyntaxGUID.Data3
                db 0A0h, 22h, 0A6h, 0B7h, 1Bh, 0FBh, 9Dh, 43h; InterfaceId.SyntaxGUID.Data4
                dw 2                    ; InterfaceId.SyntaxVersion.MajorVersion
                dw 0                    ; InterfaceId.SyntaxVersion.MinorVersion
                dd 8A885D04h            ; TransferSyntax.SyntaxGUID.Data1
                dw 1CEBh                ; TransferSyntax.SyntaxGUID.Data2
                dw 11C9h                ; TransferSyntax.SyntaxGUID.Data3
                db 9Fh, 0E8h, 8, 0, 2Bh, 10h, 48h, 60h; TransferSyntax.SyntaxGUID.Data4
                dw 2                    ; TransferSyntax.SyntaxVersion.MajorVersion
                dw 0                    ; TransferSyntax.SyntaxVersion.MinorVersion
                db 4 dup(0)
                dq offset DispatchTable_CatalogStart; DispatchTable
                dd 0                    ; RpcProtseqEndpointCount
                db 4 dup(0)
                dq 0                    ; RpcProtseqEndpoint
                dq 0                    ; DefaultManagerEpv
                dq offset InterpreterInfo_CatalogStart; InterpreterInfo
                dd 6000000h             ; Flags
                db 4 dup(0)
```

Figure 7. Dump of the RPC_SERVER_INTERFACE provided by cryptcatsvc.dll.

server RPC (corresponding to {f50aac00-c7f3-428e-a022-a6b71bfb9d43}, as targeted by ci.dll) and the "InterpreterInfo" table (indicated in blue), providing an access to a MIDL_SERVER_INFO structure. The second entry of this structure corresponds to a Dispatch Table of server routines (as illustrated in Figure 8). This table (given in Figure 8) references the whole list of RPC functions exported for clients.

```
ServerRoutineTable_CatalogStart dq offset s_SSCatDBAddCatalog
                                        ; DATA XREF: .rdata:InterpreterInfo_CatalogStart↑o
                dq offset s_SSCatDBDeleteCatalog
                dq offset s_SSCatDBEnumCatalogs
                dq offset s_SSCatDBPauseResumeService
                dq offset s_SSCatDBRebuildDatabase
                dq offset s_SSCatDBPrepareForCall
                dq offset s_SSCatDBAddCatalog2
                dq offset s_SSCatDBSmartlockerDefenderCheck
                dq offset s_SSCatDBSmartlockerDefenderCheck2
                dq offset s_SSCatDBSendSmartAppControlBlockToast
                dq offset s_SSCatDBSendSmartAppControlSwitchEnforceToast
                dq offset s_SSCatDBQueryChainByCerts
                dq offset s_SSCatDBMpSmartLockerEnable
                align 10h
```

Figure 8. List of routines exported by the RPC server in the Server RPC Dispatch Table Routines.

In the kernel, `SSCatDBSmartlockerDefenderCheck2` function calls the `Ndr64AsyncClientCall` function with the parameter "nProcNum" equals to 8. This means that the 8th entry of the RPC server dispatch

routine table will be called. In our case, it means that `s_SSCatDBSmartlockerDefenderCheck2` function will be RPC called.

4.1.4 Scan analysis Overview

The notification procedure is a three steps operation. Starting from the `MpCheckMappedFileTrust` function in the MpClient.dll, there is first the initialization and the launch of the scan operation with the call to the `ClientMpRpcMemoryScanStart` function. In the MpSvc.dll, after having initialized a "memory scan engine VFZ" in the `MpService::CMpMemScanEngineVfz::Initialize` function, there is a thread pool notification performed via the `CommonUtil::CMpSimpleThreadPool::Submit` function to register a scan work to be performed by the Microsoft Defender Antivirus's engine. Previously during the initialization of Microsoft Defender Antivirus, this thread pool has been initialized to be driven with the `CommonUtil::CMpThreadPoolProviderVista::WorkCallback` function callback. This one dequeue a work item (representing a scan operation in our case) already pushed in the working queue to perform a scan operation. This scan operation has a short transition by the engine of Microsoft Defender Antivirus, routing an "internal signal" to the `ScanStreamBuffer` function in mpengine.dll. The most relevant part of this scan procedure (for SAC) is the generation of a Spynet report with and its submission to the Microsoft's cloud-based backend. This last operation is performed by the `MpService::CMpSpyNetManager::SubmitReport` function from MpSvc.dll.

Once the report has been generated and sent to Microsoft, the notification procedure waits for the answer from the cloud-based analysis. The wait operation is driven by some "queued events" linked to the request. Once the answer from the cloud-based analysis has been retrieved, there is a parsing operation and an extraction of information, the last being performed by the `ServerMpRpcMemoryScanQueryNotification` function from MpSvc.dll.

Many kinds of answers are returned by the cloud-based analysis, in a queue of elements extracted before by the parsing operation. The identification of the different kinds of answers is based on the header value of each element represented in the answer. This is a value whose hexadecimal value must be either 0x31001, 0x31002, or 0x4005. Depending on the header value hence returned, the structure representing the element in the answer will be different (since it does not hold the same kind of information). The goal of this procedure is to get an access to each element of the answer for further proceedings, including taking the decision to know if the submitted application must be executed or not (if SAC is in enforcement mode). Otherwise, if SAC is in evaluation mode, it is just about warning the user with a WNF toast notification that the application would have been refused if SAC would have been in enforcement mode (in such a case, SAC is in evaluation mode).

4.1.5 From Microsoft Defender Antivirus to the analysis provided by Microsoft's cloud-based backend

Part of the scan procedure, there is a notification to the Microsoft's cloud-based backend for an analysis. This one is central since it is responsible to collect information from Microsoft to know if a given file could be executed but also because of the potential disclosure of information it may represent. These two axes are the focus given to this subsection about the network analysis of SAC feature. The goal is not to document here internal implementation details but instead to provide a clear view about the data transiting on the network in SAC.

It is worth noting that the internal functions used in Microsoft Defender Antivirus to communicate over the network are not dedicated to SAC. This is a reuse of a code which can be notified in other context, mainly for the purpose of obtaining a cloud-based analysis.

The data exchanged over the network is principally between the Microsoft Defender Antivirus and Microsoft's backend infrastructure. The primary focus is on the network interface allowing communication between the Microsoft Defender Antivirus and the Microsoft backend infrastructure.

As a first step to analyze the network interface, a network capture has been performed using the Wireshark utility[6]. This utility enables the analysis of network data that is being transmitted between the Microsoft Defender Antivirus and the Microsoft's cloud-based backend infrastructure. An illustration of a section of the output generated by WireShark is given in Figure 9. The output shows the communication between Microsoft Defender Antivirus and Microsoft occurs over a network connection that is secured using Transport Layer Security (TLS). Within this secure connection, the exchange involves messages formatted using the Hypertext Transfer Protocol (HTTP), as evidenced by the information presented in the *'SrcPort'*, *'DstPort'*, and *'Protocol'* columns within Figure 9, but also with further analysis in the module implementing this exchange of information.

Source	SrcPort	Destination	DstPort	Protocol	Info
172.28.11.176		172.28.8.3		DNS	Standard query 0x1695 A wdcp.microsoft.com
172.28.11.176		172.28.8.3		DNS	Standard query 0x1695 A wdcp.microsoft.com
172.28.8.3		172.28.11.176		DNS	Standard query response 0x1695 A wdcp.microsoft.com CNAME wd-prod-cp.trafficmanager.net CNAME wd-prod-cp-eu-north-2-fe.northeurope.cloudapp.azure.com A 20.82.207.122
172.28.11.176	48334	20.82.207.122	443	TCP	48334 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=1845013442 TSecr=0 WS=128
172.28.8.3		172.28.11.176		DNS	Standard query response 0x1695 A wdcp.microsoft.com CNAME wd-prod-cp.trafficmanager.net CNAME wd-prod-cp-eu-north-2-fe.northeurope.cloudapp.azure.com A 20.82.207.122
20.82.207.122	443	172.28.11.176	48334	TCP	443 → 48334 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1440 WS=256 SACK_PERM
172.28.11.176	48334	20.82.207.122	443	TCP	48334 → 443 [ACK] Seq=1 Ack=1 Win=64256 Len=0
172.28.11.176	48334	20.82.207.122	443	TLSv1.2	Client Hello
20.82.207.122	443	172.28.11.176	48334	TLSv1.2	Server Hello, Certificate, Server Key Exchange, Server Hello Done
172.28.11.176	48334	20.82.207.122	443	TCP	48334 → 443 [ACK] Seq=234 Ack=3868 Win=60416 Len=0
172.28.11.176	48334	20.82.207.122	443	TLSv1.2	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
20.82.207.122	443	172.28.11.176	48334	TLSv1.2	Change Cipher Spec, Encrypted Handshake Message
172.28.11.176	48334	20.82.207.122	443	TCP	48334 → 443 [ACK] Seq=392 Ack=3919 Win=64128 Len=0
172.28.11.176	48334	20.82.207.122	443	TLSv1.2	Application Data
20.82.207.122	443	172.28.11.176	48334	TCP	443 → 48334 [ACK] Seq=3919 Ack=800 Win=524800 Len=0
172.28.11.176	48334	20.82.207.122	443	TCP	48334 → 443 [ACK] Seq=800 Ack=3919 Win=64128 Len=1440 [TCP segment of a reassembled PDU]
172.28.11.176	48334	20.82.207.122	443	TCP	48334 → 443 [PSH, ACK] Seq=2240 Ack=3919 Win=64128 Len=20 [TCP segment of a reassembled PDU]
172.28.11.176	48334	20.82.207.122	443	TCP	48334 → 443 [ACK] Seq=2260 Ack=3919 Win=64128 Len=1440 [TCP segment of a reassembled PDU]
172.28.11.176	48334	20.82.207.122	443	TLSv1.2	Application Data

Figure 9: A snippet of the output of WireShark

[6] https://www.wireshark.org/

Before any data exchange occurs over a TLS-secured network interface, a communication session must be established. This procedure typically involves server authentication, negotiation of a cipher suite designated for data encryption, as well as the generation of a session encryption key. During session establishment, Microsoft Defender Antivirus sends a 'client hello' message to the server, along with the supported cipher suites (see *'Cipher Suites'* in Figure 11).

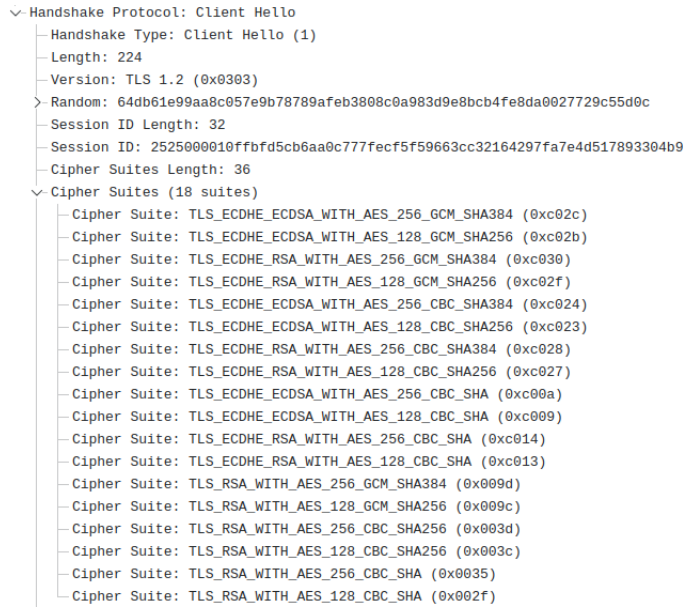

Figure 11. A piece of a 'client hello' message.

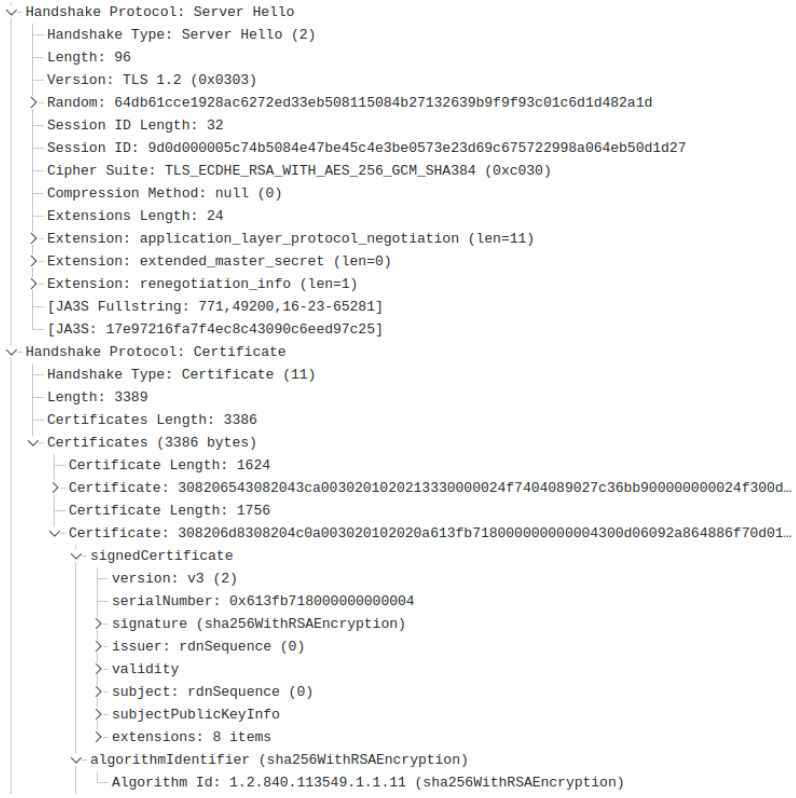

Figure 10. A portion of a 'server hello' message.

Microsoft Defender Antivirus directly communicates information with the “https://wdcp.microsoft.com” URL which is resolved in our case by the 20.82.207.122 IP address. The TLS certificate associated with the HTTPS connection is provided in Figure 10.

The API used for the communication is based on the Microsoft Windows HTTP Service (see section 3.1.12) framework. Microsoft Defender Antivirus operates this framework through the mpcommu.dll library. Together, this library and the WinHTTP framework are the core of the network capabilities used by the service to communicate with the Microsoft’s cloud-based backend.

4.1.5.1 TLS Server Certificate Validation Policy

The WinHTTP framework is responsible to handle the procedure of setting up a HTTP connection over a TLS protected tunnel. Such a procedure is achieved through the invocation of a predefined sequence of functions, each playing a distinct role in the establishment of the TLS protected connection (as outlined in (Microsoft)).

When implementing an HTTPS connection, the state of the art to secure the implementation involves the use of an event driven callback mechanism to check the certificate of the targeted server once the connection has been established. This is the `WinHttpSetStatusCallback` function which allows to set this callback mechanism. This function serves to establish a callback function, thereby enabling WinHTTP to invoke the specified callback function as the sequence unfolds (see (Microsoft)).

Among other things, this callback mechanism is used to verify the authenticity of the server providing the cloud-based analysis. It ensures the legitimacy of the server with whom the connection is established. When the message is received, the established callback function is triggered, directing the invocation towards the `winHttp::CRequest::OnSendingRequest` function. This function invokes the `winHttp::CRequest::CheckCertForMicrosoftRoot` function, which is responsible for verifying the certificate chain's integrity and confirming its authenticity. Code Block 1 illustrates a portion of the `winHttp::CRequest::CheckCertForMicrosoftRoot` function, enlighten the criteria upon which the verification of the certificate chain's integrity and authenticity is carried out.

```
winHttp::CRequest::CheckCertForMicrosoftRoot(HINTERNET hInternet)
{
[...]
  if ( WinHttpQueryOption(hInternet, WINHTTP_OPTION_SERVER_CERT_CONTEXT, &pCertContext, &dwPtrLength)
) {
    [...]
    if ( CertGetCertificateChain(
            0,
            pCertContext,
            0,
            pCertContext->hCertStore,
            &pChainPara,
            0,
            0,
            &pChainContext) )
    {
      [...]
      pPolicyPara.dwFlags = MICROSOFT_ROOT_CERT_CHAIN_POLICY_CHECK_APPLICATION_ROOT_FLAG;
          if   (   CertVerifyCertificateChainPolicy(CERT_CHAIN_POLICY_MICROSOFT_ROOT,   pChainContext,
&pPolicyPara, &pPolicyStatus) ) {
        [...]
        if ( pPolicyStatus.dwError ) {
          dwError = LOWORD(pPolicyStatus.dwError) | 0x80070000;
          if ( (int)pPolicyStatus.dwError <= 0 )
            dwError = pPolicyStatus.dwError;
        }
        else { dwError = 0; }
      }
  [...]

  return dwError;
}
```

Code Block 1: Portion of CheckCertForMicrosoftRoot function.

The function assesses the validity of a server's certificate chain in relation to Microsoft's root certificate authorities. The key steps are as follows:

1. Certificate Context Retrieval: The `WinHttpQueryOption` function acquires the certificate context of the server associated with the provided `hInternet` handle representing the HTTPS connection. This certificate is the one associated with the 'server hello' message. Stored in the `pCertContext` variable, this context becomes the foundation of subsequent evaluations.
2. Certificate Chain Retrieval: The `CertGetCertificateChain` function initializes a certificate chain context, beginning from the provided certificate and potentially extending backward to a trusted root certificate, based on the previously acquired `pCertContext` variable. Upon successful completion, the resulting chain context is stored in the `pChainContext` variable.
3. Certificate Chain Policy Verification: The function `CertVerifyCertificateChainPolicy` is used to perform a verification of the chain context stored within the `pChainContext` variable. The first parameter `CERT_CHAIN_POLICY_MICROSOFT_ROOT` indicates that the last element of the chain must corresponds to a Microsoft root public key. In addition, the `dwFlags` attribute of the `CERT_CHAIN_POLICY_PARA` structure indicated by the `pPolicyPara` parameter includes the `MICROSOFT_ROOT_CERT_CHAIN_POLICY_CHECK_APPLICATION_ROOT_FLAG` flag. By setting this flag, the function does an additional verification by checking if the last element of the chain corresponds to a "Microsoft Root Certificate Authority 2011" public key. This means that the hash of the public key is validated against a predefined instance of a public key hash associated with the "Microsoft Root Certificate Authority 2011". This hash is statically predefined within the crypt32.dll file. This procedure is called "certificate pinning" in secure programming area.
4. Error Handling: In the considered scenario, if the last element of the chain does not correspond to a public key associated with "Microsoft Root Certificate Authority 2011", the `dwError` attribute of the `CERT_CHAIN_POLICY_STATUS` structure referenced by the `pPolicyStatus` parameter is set to `CERT_E_UNTRUSTEDROOT` (`0x800B0109`). Accessed through the `dwError` variable of the `winHttp::CRequest::CheckCertForMicrosoftRoot` function, the returned error code depends on the value set in the `dwError` attribute of the `CERT_CHAIN_POLICY_STATUS` structure. If there is an error, the low-order word value of the `dwError` attribute is extended with the value `0x80070000`. In addition, if the `dwError` attribute value is negative, it is assigned directly to `dwError` variable.

***Note**: Relevant events related to certificate operations (e.g., certificate verification) are logged by the ETW provider with GUID 5bbca4a8-b209-48dc-a8c7-b23d3e5216fb. This provider is registered under the name `Microsoft-Windows-CAPI2`. It logs data related to the certificate verification process described in this section.*

When the certificate verification process implemented in the `winHttp::CRequest::CheckCertForMicrosoftRoot` takes place, the following events are logged in a sequential order: Event ID 10, Event ID 11, Event ID 30, and Event ID 90. The events with IDs 10, 11, and 90 are generated in the `CertGetCertificateChain` function. The event with ID 10 indicates that the extraction of the certificate chain from the 'server hello' message has started. The event with ID 90 shows that actual certificates are part of this chain. The event with ID 11 indicates the validity of the certificate chain. In the end, the event with ID 30 is generated in the `CertVerifyCertificateChainPolicy` function to indicate the validity of the root certificate.

Note 1: ETW information related to certificate operations.

4.1.5.2 Microsoft Defender Antivirus and Cloud-Based Security Service Data Exchange

To get a confident prediction for each of the file to be analyzed, a report holding information about the file is generated for each of them by Microsoft Defender Antivirus, more specifically by the different modules composing the MsMpEng service, i.e., MpSvc, Mpengine and MpCommu DLLs. The goal is to transfer these reports over the network to use the capabilities of the cloud-based analysis. This operation is performed through a HTTPS POST request as explained in section 4.1.5.1.

Based on the rich information contained within these reports, the cloud-based analysis provides an answer concerning the potential trustworthy assessment regarding the file submitted. The decision arising from the cloud-based analysis is subsequently communicated back through a HTTPS response to Microsoft Defender Antivirus. This exchange of information empowers Microsoft Defender Antivirus with the authority to make cloud-based informed decisions regarding whether to allow or deny the execution of the specific file originally requested by the kernel.

For a better overview of the data exchanged over the network, it matters to describe the content of the requests exchanged over the network. That way, we propose to consider first the request sent to the Microsoft's cloud-backend for analysis and then the answer. To make the distinction between the two requests, we call the initial request the "initial request" and the return from the server the "answering request".

Initial Request:

The initial HTTPS POST request is used to send a report to the Microsoft's cloud-based backend. As any HTTPS request, it consists of two main components: the request header and the request body. Together, the header and the body define the intent and the content of a regular HTTPS request.

On the one hand, the request's header contains specific information to properly communicate with Microsoft Defender Antivirus. In particular, the header of the request holds of a specific user-agent ("MpCommunication") and specific Microsoft defined non-standard request fields. This informs us how to communicate with the servers of the Microsoft's cloud-based backend but also how to potentially filter requests to monitor the traffic related to cloud-based analysis performed via Microsoft Defender Antivirus.

Code Block 2 shows the information provided through the HTTPS request' header to the URL "https://wdcp.microsoft.com/wdcp.svc/bond/submitreport" when submitting a report. We propose to detail the most relevant fields of this request's header.

```
POST /wdcp.svc/bond/submitreport HTTP/1.1
Connection: Keep-Alive
Content-Type: application/bond
Accept: application/bond
Accept-Charset: utf-8
User-Agent: MpCommunication
X-MS-MAPS-CUSTOMERTYPE: Consumer
X-MS-MAPS-OSVERSION: a00000000585d
X-MS-MAPS-PLATFORMVERSION: 400125a1e03ec
X-MS-MAPS-ENGINEVERSION: 100015a1e03ed
Content-Length: 2890
Host: wdcp.microsoft.com
```

Code Block 2: Http request header.

POST /wdcp.svc/bond/submitreport HTTP/1.1: This is the request line indicating that it is a POST request to the /wdcp.svc/bond/submitreport endpoint using the HTTP protocol version 1.1.

Connection: Keep-Alive: It specifies that the connection should be kept alive for potential reuse, which is a performance optimization.

Content-Type: application/bond: It informs that the content of the request body is in the application/bond format. This header tells the server how to interpret the data in the request body.

Accept: application/bond: It specifies the expected response format, indicating in our case that the client can accept a response in the application/bond format.

Accept-Charset: utf-8: It specifies that the client can accept characters encoded in UTF-8, indicating the character encoding used in the response.

User-Agent: MpCommunication: It provides information about the user agent or client making the request. In this case, it identifies itself as "MpCommunication."

X-MS-MAPS-CUSTOMERTYPE: Consumer, X-MS-MAPS-OSVERSION: a00000000585d, X-MS-MAPS-PLATFORMVERSION: 400125a1e03ec, X-MS-MAPS-ENGINEVERSION: 100015a1e03ed: These are custom headers that provide additional information to the server. They are not standard HTTP headers but are used by the service for additional context or version information. We recognize in this context the version of the current operating system and the version of Microsoft Defender Antivirus.

Content-Length: 2890: It specifies the length of the request body in bytes. This is important for the server to know how much data to expect in the request body.

Host: wdcp.microsoft.com: It specifies the domain name or the IP address of the server to which the request is being sent.

On the other hand, the request's body carries the actual data being transmitted to the cloud-based security service. The format and the content of the request body depend on the specified *Content-Type* in the request header. In our context, the expected format is "Bond"[7], an open-source and cross-platform framework[8] used to work with schematized data about to be serialized and deserialized[9]. According to Microsoft, Bond is broadly used at Microsoft in high scale services, and it is claimed to be particularly well-suited for scenarios where data needs to be exchanged between different systems or components written in different languages and running on different platforms. Code Block 3 illustrates a portion of the request body that carries the actual data being transmitted where we can observe – as expected – that the data is serialized as a binary "blob" (hexadecimal values) where some can be interpreted as text.

```
00000224`a76d2a1d  43 42 01 00 a9 3c 4d 69-63 72 6f 73 6f 66 74 2e  CB...<Microsoft.
00000224`a76d2a2d  50 72 6f 74 65 63 74 69-6f 6e 53 65 72 76 69 63  ProtectionServic
00000224`a76d2a3d  65 73 2e 45 6e 74 69 74-69 65 73 2e 52 61 77 2e  es.Entities.Raw.
00000224`a76d2a4d  53 70 79 6e 65 74 52 65-70 6f 72 74 45 6e 74 69  SpynetReportEnti
00000224`a76d2a5d  74 79 01 01 cb 0a 0b 01-0e 00 ca 14 a9 3c 4d 69  ty...........<Mi
00000224`a76d2a6d  63 72 6f 73 6f 66 74 2e-50 72 6f 74 65 63 74 69  crosoft.Protecti
00000224`a76d2a7d  6f 6e 53 65 72 76 69 63-65 73 2e 45 6e 74 69 74  onServices.Entit
00000224`a76d2a8d  69 65 73 2e 52 61 77 2e-53 70 79 6e 65 74 52 65  ies.Raw.SpynetRe
00000224`a76d2a9d  70 6f 72 74 45 6e 74 69-74 79 01 01 cb 14 0f 01  portEntity......
00000224`a76d2aad  06 c9 1e 24 66 63 66 30-34 31 30 36 2d 61 38 63  ...$fcf04106-a8c
00000224`a76d2abd  66 2d 34 63 32 34 2d 39-33 33 31 2d 33 30 38 38  f-4c24-9331-3088
00000224`a76d2acd  31 30 66 32 65 30 62 34-c9 28 0b 31 2e 33 39 35  10f2e0b4.(.1.395
00000224`a76d2add  2e 34 39 38 2e 30 c9 2b-0b 31 2e 33 39 35 2e 34  .498.0.+.1.395.4
00000224`a76d2aed  39 38 2e 30 c9 32 0b 31-2e 33 39 35 2e 34 39 38  98.0.2.1.395.498
00000224`a76d2afd  2e 30 c9 3c 0e 31 2e 31-2e 32 33 30 37 30 2e 31  .0.<.1.1.23070.1
00000224`a76d2b0d  30 30 35 c9 46 0b 31 2e-33 39 35 2e 34 39 38 2e  005.F.1.395.498.
00000224`a76d2b1d  30 c9 50 0e 31 2e 31 2e-32 33 30 37 30 2e 31 30  0.P.1.1.23070.10
00000224`a76d2b2d  30 35 c9 96 08 31 30 2e-30 2e 30 2e 30 cb aa 05  05...10.0.0.0...
00000224`a76d2b3d  01 dd b0 01 cb b4 10 01-80 04 cb be 0f 01 02 cb  ................
00000224`a76d2b4d  c8 10 01 bc 01 cb d2 05-01 80 40 cb dc 04 01 09  ..........@.....
00000224`a76d2b5d  e9 18 01 24 37 37 62 64-61 66 37 33 2d 62 33 39  ...$77bdaf73-b39
00000224`a76d2b6d  36 2d 34 38 31 66 2d 39-30 34 32 2d 61 64 33 35  6-481f-9042-ad35
00000224`a76d2b7d  38 38 34 33 65 63 32 34-e9 22 01 01 32 e9 2c 01  8843ec24."..2.,.
00000224`a76d2b8d  0f 34 2e 31 38 2e 32 33-30 37 30 2e 31 30 30 34  .4.18.23070.1004
00000224`a76d2b9d  eb 90 01 0b 01 0a 01 cb-14 02 01 01 cb 28 0b 01  .............(..
00000224`a76d2bad  0a 01 cb 0a 10 01 02 c9-14 28 36 62 38 61 34 36  .........(6b8a46
00000224`a76d2bbd  35 66 61 61 39 36 34 30-39 33 32 36 33 31 31 37  5faa964093263117
00000224`a76d2bcd  63 63 38 64 36 31 66 36-66 32 37 37 62 62 65 35  cc8d61f6f277bbe5
00000224`a76d2bdd  32 30 c9 1e 40 63 31 62-38 39 65 61 61 65 31 65  20..@c1b89eaae1e
00000224`a76d2bed  36 38 31 32 63 39 65 30-34 37 37 32 35 64 66 62  6812c9e047725dfb
00000224`a76d2bfd  65 62 38 36 61 32 38 39-38 37 65 31 65 64 30 62  eb86a28987e1ed0b
00000224`a76d2c0d  62 32 31 63 38 64 32 36-34 32 64 34 38 33 33 65  b21c8d2642d4833e
00000224`a76d2c1d  32 61 61 37 66 c9 3c 63-31 2c 35 2c 32 31 2c 32  2aa7f.<c1,5,21,2
00000224`a76d2c2d  33 2c 33 31 2c 33 37 2c-34 35 2c 38 34 2c 39 34  3,31,37,45,84,94
00000224`a76d2c3d  2c 39 37 2c 31 30 36 2c-31 30 39 2c 31 31 30 2c  ,97,106,109,110,
00000224`a76d2c4d  31 31 33 2c 31 31 34 2c-31 31 35 2c 31 31 36 2c  113,114,115,116,
00000224`a76d2c5d  31 31 37 2c 31 31 39 2c-31 32 32 2c 31 32 36 2c  117,119,122,126,
00000224`a76d2c6d  31 34 31 2c 31 35 37 2c-31 35 38 2c 31 39 35 2c  141,157,158,195,
00000224`a76d2c7d  32 30 32 2c 32 31 34 2c-32 32 35 c9 46 93 05 30  202,214,225.F..0
```

[7] https://github.com/microsoft/bond

[8] https://microsoft.github.io/bond/manual/bond_cs.html

[9] *Serialization is the procedure of converting data into a format, which is efficient in terms of both space and speed. This makes it suitable for scenarios where minimizing data size and reducing network or storage I/O are important. Deserialization is the procedure of reconstructing data from its serialized form.*

```
00000224`a76d2c8d  78 38 35 33 61 39 37 62-35 3a 30 78 30 30 30 30  x853a97b5:0x0000
00000224`a76d2c9d  31 38 65 37 63 62 61 61-34 61 30 39 2c 30 78 65  18e7cbaa4a09,0xe
00000224`a76d2cad  33 66 36 34 38 34 37 3a-30 78 30 30 30 30 31 66  3f64847:0x00001f
00000224`a76d2cbd  65 37 34 32 64 31 65 39-33 35 2c 30 78 36 39 66  e742d1e935,0x69f
[...]
```

Code Block 3: Serialized HTTP request body

Once deserialization is done, the report is a schematized data structure, that can conceptually be interpreted as a property tree structure (i.e., hierarchical structure). This schematized data structure uses “elements” and “attributes” to describe data within the report. Conceptually, this is not far from an XML-based property tree structure. An explanation of elements and attributes in a Microsoft Defender Antivirus generated file report is presented below:

Elements: They serve as the fundamental building blocks that represent containers for data in a report. They play a central role in defining the hierarchical structure of the data within the report. Each element has a distinct name, and it may be accompanied by attributes. Additionally, elements can be embedded inside other elements as child elements (hence the tree structure), and they can also contain data content, representing the values associated with the element.

Attributes: They serve as metadata linked to an element, offering supplementary details regarding the element itself, rather than the data encapsulated within it. These attributes are always tied to a specific element, and they are composed of a name-value pair. They are commonly employed to impart further traits or properties to elements.

Code Block 4 illustrates the parsed content of the report property tree structure, presenting both the names and values of elements along with their associated attributes.

```
<SpynetReport>
    <enginereportguid=AC25A8A8-E1E9-4F0B-A52B-17A83123E455>
    <revision=33>
    <xmlns=AntiMalwareServices.Components.SpynetReport>
    <machineguid=FCF04106-A8CF-4C24-9331-308810F2E0B4>
    <productguid=77BDAF73-B396-481F-9042-AD358843EC24>
    <appversion=4.18.23070.1004>
    <assigversion=1.395.498.0>
    <avsigversion=1.395.498.0>
    <sigversion=1.395.498.0>
    <osver=10.0.0.0>
    <osbuild=22621>
    <ossuite=256>
    <ostype=31>
    <studyid=GR-OfficeRingNotSet>
    <cioptions=132>
    <engineversion=1.1.23070.1005>
    <computernetbiosname=DESKTOP-D647BKN>
    <osplatform=31>
    <officeconfigurationvalue=4294967295>
    <passivemoderemediation=30>
    <servicestartstates=30>
    <tamperprotectionstate=31>
    <tamperprotectionexclusionsstate=30>
    <cloudbadlistversion=35>
    <pvpring=4294967295>
    <computerdnsnamehash=0bc1e87a>
    <membership=32>
    <nrienginеversion=1.1.23070.1005>
    <nrisigversion=1.395.498.0>
    <engineloadtime=133365711362038044>
    <osproducttype=34>
    <osrevision=1702>
    <geoid=94>
    <lcid=8192>
    <processor=39>
    <supportedcompressions=Xor,DeflateLevel1Xor>
    <autosampleoptinvalue=31>
    <puamode=32>
    <wuuniqueid=4117f91d-2707-46c2-a744-e204a93d8a8a>
    <machinesqmid={44E12583-63E7-4102-8AC0-588135985892}>
    <vditype=31>
    <ismeterednetwork=30>
    <isverifiedandreputabletrustmode=31>
```

```
    <isverifiedandreputableperfmode=30>
    <SignatureRequest>
        <dsssource=31>
        <averagerttime=37737>
        <rttimestddev=27270>
        <rttimehistory=5068,3618,60000,60000,60000>
        <lowficount=31>
        <isvfz=31>
        <rtsd=31>
    <FileQuery>
        <realpath=C:\Users\test\AppData\Local\Temp\Temp1_QSlice.zip\QSlice\QSlice.exe>
        <hashedfullpath=7751e6b401a4cee531e7b48b2d9ec91e0a402982>
        <filename=QSlice.exe>
        <type=31>
        <sha1=6b8a465faa964093263117cc8d61f6f277bbe520>
        <sha256=c1b89eaae1e6812c9e047725dfbeb86a28987e1ed0bb21c8d2642d4833e2aa7f>
        <md5=a6ec35b8d4289ddb5d0486d3d6745ce9>
        <partialcrc1=07331507>
        <partialcrc2=9d6ce0e4>
        <partialcrc3=07ec90ed>
        <crc16=ab6ea4b5>
        <filesize=40960>
        <scenario=32>
        <agent=771>
        <namedattributes=0x853a97b5:0x000018e7cbaa4a09,0xe3f64847:0x00001fe742d1e935,0x69fa3b04:0x00
0065e719e6bf60,0x0237c5a9:0x000090e7598832e2,0xea5e5ef0:0x00000e965423df74,0x3dcfccd7:0x000030e74569
fcbf,0xcb0224c1:0x000062bd35849f17,0xaafe9f57:0x000098e770c9c9b8,0x02335526:0x000024e74bd1dcb7,0x677
23404:0x0001ccbdd4aed7de,0xb52f98d7:0x0000c1e779917b93,0x23ebf6a6:0x000062788d314f8a,0xb1eb3f24:0x00
001d61e0587711,0x99c32ee3:0x00002c78457f7afc,0xbb89c891:0x0000517876bd2f81,0xea4fb480:0x00005878c84c
8ef3,0xa8113bde:0x00006e7892221938,0x6a98a299:0x000038e7f286fca8,0x53ccbda9:0x000201bd3ab7c237,0x419
40c02:0x000018e78ecbe76a,0xf331e554:0x002ab0bdce726dc0,0x2920d030:0x00019ebdd6fddd48>
        <peattributes=1,5,21,23,31,37,45,84,94,97,106,109,110,113,114,115,116,117,119,122,126,141,15
7,158,195,202,214,225>
        <sigattrevents=12362>
        <filetypestring=BM MZ FILE>
        <imphash=f34d5f2d4577ed6d9ceec516c1f5a744>
        <pdbproject={4240609C-A95A-4412-8F20-1C610AFB046C}-1-
C:\Dev\QSlice\QSlice\obj\Release\QSlice.pdb>
        <extendedkcrcs=KFWD:8c5affbb:a40d8a23:03b0f146:80001000,KREV:8c5affbb:c6fb44bb:fe8f7320:8000
1000,KSTD:8c5affbb:c6fb44bb:e2b4bfab:80001000,KHEA:9cc88bce:35ccb4b3:c709eefb:80000400,KFOO:e6f5aa52
:8a729cc9:8acfb54c:80000400>
        <name=QSlice>
        <originalname=QSlice.exe>
        <version=1.0.0.0>
        <ctph=768:PWwywT++ofkLqhf9v6xetXqGh34WyrBjyXntYcFOKc6K:PpJlnWhf9Lt335MBjQ/OKc1>
        <lshashs=ff7cad5849f02d1696ad47801bfa6eb061c5f42b1319e8ac18cef21f3043b53c>
        <lshash=efffe67ff7dd9a7aeb7569fbbb5e795aba97f7a97f7655a5af57dabfad6955ef567d77f9b6fedf699f9a
9756aafdead9e596aeb5adfaff9a955f9e65a79bf95a>
        <threattrackingid=1988DFFB-62C8-4F84-9983-2BDE6E3462D9>
        <isvfz=31>
        <motwreferrerurl=C:\Users\test\Downloads\AllTools1\AllTools-master\QSlice.zip>
        <markoftheweb=33>
        <filedevicecharacteristics=131104>
        <samplesubmissionineligiblereason=31>
        <rtpscanreason=16>
        <index=31>
    <ResearchDataEx>
    <MpKeyValuePair>
        <key=MSILMVID>
        <value=ea22081b95d32c458f099dff9f1db12900000000>
    <MpKeyValuePair>
        <key=SECTHDR>
        <value=94428f077a51b67e0d4cce707ae43d0d9fdc98a4>
    <MpKeyValuePair>
        <key=PESTRUCT>
        <value=0db6f885ed4092d91e754ce22d59caca5bdfc828>
    <Audit>
        <sigseq=0001ccbdd4aed7de>
        <sigsha=94e38f7de08ae29cca09bdd5f24225dc44c7fb98>
        <isprimary=31>
```

Code Block 4: Deserialized HTTP request body.

The interesting point about the data sent relies in the diversity of information sent to Microsoft. We can make the distinction between different three different kinds of information. At first, there are all the data concerning the Microsoft Defender Antivirus metadata, especially the versions of the antivirus' engine or the signature database. Then, we have the data related to the operating system itself. It includes the Windows'

version number, but also machine's identification numbers. Finally, we have information about the file submitted, mainly metadata concerning the file (file path for instance, size of the file) and different kinds of hashes (SHA1, SHA256, MD5) of the file. We also note the presence of an index GUID representing the report submitted, used as a tracking number, especially in the answering request.

Answering request:

The answer returned to the HTTPS request represents the cloud-based analysis answer to the initial Microsoft Defender Antivirus's analysis request. It contains information regarding the outcome of the request, allowing Microsoft Defender Antivirus to understand the reputation of the file submitted for analysis.

The content of the answer consists of two main components: the response header and the response body. In the header of the answer, besides the HTTP status code indicating whether the request was successful or not, there is a mention of the server's name, "Kestrel" in our case (as shown in Code Block 5).

```
HTTP/1.1 200 OK
Content-Length: 448
Content-Type: application/bond
Date: Tue, 15 Aug 2023 11:32:53 GMT
Server: Kestrel
```

Code Block 5: HTTP response header.

The body of the response holds all the relevant information convoyed by the cloud-based analysis. It is used to provide feedback about the analyzed file but also potential information about the current mode in which SAC is running. We observe the answer is "personalized" for SAC, meaning that the generic SpyNet communication system – potentially used by different elements of Microsoft Defender Antivirus – is identified with the input parameters provided by the initial request.

The content of the answer is formatted the same way the initial request was, with the "application/bond" content-type. After having parsed the content of the answer (Code Block 6), it is possible to observe different kinds of elements. At first, there is a notion of "signature", as given in (Code Block 6) and explained in section 3.1.3.

```
<SpynetReportResponse>
    <Revision=5>
    <SampleRate=1>
    <SignaturePatches=0>
    <EnableBlob=
        'ec 00 01 00 98 e1 d4 05 bc ae 79 3c 5e 0e e2 34'
        '58 cb 5a a0 e8 ce c2 63 33 02 d6 9f 65 54 4d 40'
        'fe ba 21 8e 6c fa 2a 78 f1 21 ed c2 c0 0d c4 4c'
        '21 9d 47 10 4b 7c 1e e3 86 e3 c5 63 e5 a6 36 2c'
        'ee 72 16 97 3f 1d 69 e5 e5 67 ff 4f 2d 4b 64 03'
        'c0 9e 92 01 9a bd 63 a6 4f 90 bf 11 80 dd 9b 29'
        'ad fd bc 16 c4 f1 eb a0 22 05 68 37 ea f2 5a 1f'
        '65 87 fd 12 ea 44 c4 99 69 57 bc 47 c4 76 83 34'
        'd6 ae a1 b3 2b c2 be f6 e3 ae b4 99 65 84 21 c1'
        '3d 7a 98 87 83 0b 77 2d b8 a1 f3 f7 b2 12 cc c3'
        '0e 5d 12 eb f5 ff b7 cc bd 0f 7d 43 43 b8 66 c6'
        'ee b7 82 bf 04 8e 96 51 0e bc 2e c0 d1 3f 94 ac'
        '06 18 2c 82 65 c4 57 bc c8 1d 24 61 2d 69 4d 5c'
        '45 0d b4 37 8a 0f 30 5a 48 a3 a0 be a8 55 f5 e4'
        '67 e9 97 66 97 12 b2 5d b3 c4 d2 03 d6 90 91 c2'
        '1c 35 34 7d a7 ec 7f 19 59 83 6f 66 32 3a 6b 58'
        '3a 55 2e 41 aa 14 00 00 08 30 01 01 a0 86 01 00'
        '00 00 00 00 e2 c0 36 3b 6c cf d9 01 67 36 00 00'
        '07 15 33 07 ed 90 ec 07 e4 e0 6c 9d 00 a0 00 00'
        '00 10 6b 8a 46 5f aa 96 40 93 26 31 17 cc 8d 61'
        'f6 f2 77 bb e5 20 0f 23 43 6c 6e 46 69 6c 65 49'
        '6e 4d 69 6e 3a 31 '>
    <SignatureMatches=0>
    <Index=1>
```

Code Block 6: Deserialized HTTP response body

Another highly meaningful element in the response returned to SAC is the "*EnableBlob*" field. The last one contains the dynamic signature previously introduced in section 3.1.3. The Code Block 7 shows a dynamic signature generated based on the answer returned by the Microsoft's cloud-based backend.

```
'ec 00 01 00 98 e1 d4 05 bc ae 79 3c 5e 0e e2 34' 16
'58 cb 5a a0 e8 ce c2 63 33 02 d6 9f 65 54 4d 40' 32
'fe ba 21 8e 6c fa 2a 78 f1 21 ed c2 c0 0d c4 4c' 48
'21 9d 47 10 4b 7c 1e e3 86 e3 c5 63 e5 a6 36 2c' 64
'ee 72 16 97 3f 1d 69 e5 e5 67 ff 4f 2d 4b 64 03' 80
'c0 9e 92 01 9a bd 63 a6 4f 90 bf 11 80 dd 9b 29' 96
'ad fd bc 16 c4 f1 eb a0 22 05 68 37 ea f2 5a 1f' 112
'65 87 fd 12 ea 44 c4 99 69 57 bc 47 c4 76 83 34' 128
'd6 ae a1 b3 2b c2 be f6 e3 ae b4 99 65 84 21 c1' 144
'3d 7a 98 87 83 0b 77 2d b8 a1 f3 f7 b2 12 cc c3' 160
'0e 5d 12 eb f5 ff b7 cc bd 0f 7d 43 43 b8 66 c6' 176
'ee b7 82 bf 04 8e 96 51 0e bc 2e c0 d1 3f 94 ac' 192
'06 18 2c 82 65 c4 57 bc c8 1d 24 61 2d 69 4d 5c' 208
'45 0d b4 37 8a 0f 30 5a 48 a3 a0 be a8 55 f5 e4' 224
'67 e9 97 66 97 12 b2 5d b3 c4 d2 03 d6 90 91 c2' 240
'1c 35 34 7d a7 ec 7f 19 59 83 6f 66 32 3a 6b 58' 256
'3a 55 2e 41 aa 14 00 00 08 30 01 01 a0 86 01 00' 272
'00 00 00 00 e2 c0 36 3b 6c cf d9 01 67 36 00 00' 288
'07 15 33 07 ed 90 ec 07 e4 e0 6c 9d 00 a0 00 00' 304
'00 10 6b 8a 46 5f aa 96 40 93 26 31 17 cc 8d 61' 320
'f6 f2 77 bb e5 20 0f 23 43 6c 6e 46 69 6c 65 49' 336
'6e 4d 69 6e 3a 31'
```

Code Block 7: Dynamic signature based on the cloud-based analysis.

The general structure of the dynamic signature is organized in a sequence of chunks of bytes, where each chunk consists of a header part and a body part.

Header: The structure of the header is defined over a 4-byte binary value that holds crucial information. The first byte of this value serves as a magic number, acting as a unique identifier to specify the body type. Following this one, the following 3 bytes specify the length of the body binary data part[10] of the current chunk. The length represented on 3 bytes is casted with a specific arithmetic procedure to fit on a regular 4-byte value.

Body: The body is a binary data blob that matches a wide range of data types and formats. The format and the meaning of the content of the body depend on the first byte of the header (magic number). It can indifferently hold various data elements, including signatures, hashes, strings, numerical values, and more. Moreover, the body can represent one or many elements, either serving as a single value holder or as an array of values. For instance, it might include a hash value, followed by a string, and then followed by an integer.

The "*EnableBlob*" presented in Code Block 7 consists of three distinct chunks, documented as follows:

- Chunk-1: In the first chunk, the header specifies a magic number whose value is '0xEC' and a body length of 256 bytes. It means that the body of the chunk is an RSA signature. This signature will be used to validate the authenticity and the integrity of the data within the dynamic signature, for authenticity reasons. The body length suggests that it is a 2048-bit RSA key (hence encoded on 256 bytes), which is a regular design for security purposes.

Chunk-2: In the second chunk, the header specifies a magic number whose value is '0xAA' and a body length of 20 bytes. Within the body of this chunk, we can find information related versioning of Microsoft Defender Antivirus.

Chunk-3: In the third chunk, the header specifies a magic number whose value is '0x67' and a body length of 55 bytes. Its body contains a combination of data properties, including 'partialcrc,' 'sha1,' and 'sig properties,' notably marked by the absence of names but represented as structured elements like '(ClnFileInMin:1).' This chunk appears to provide a compact representation of properties and attributes, aligning with the file report send to the Microsoft's cloud-based backend.

[10] *This value represents a total size of 4 bytes plus the body length.*

***Note**: From the processing logic of dynamic signatures, it can be inferred that the provided answer directly controls the logic sustaining the current SAC mode. For instance, two specific chunks are directly associated with SAC, especially to control its current mode:*

1. *The chunk with a header magic number value of '0xCE'. When a chunk begins with this specific magic number, it means a transmission of a control order related to SAC. More specifically, where there is at the offset 12 relative to the beginning of the chunk, the magic number '0x9B' set followed by another data value, therefore there is an update of the remaining duration during which SAC remains in its evaluation state.*

Note 2: Additional information about the logic sustaining the current SAC mode.

2. *The chunk with a header magic number value of '0x55'. When a chunk begins with this specific magic number, it means a transmission of control order related to SAC. More specifically, when there is at the offset 8 relative to the beginning of the chunk the magic number '0x9A' set followed by a data value, therefore there is an update of the current mode where SAC belongs (from evaluation state to enforcement or deactivation).*

***Note**: It is important to emphasize that the magic numbers and offsets presented in this section are potentially subject to frequent changes. However, the overall logic and mechanisms behind these values to remotely drive SAC or for file analysis should remain valid. These magic numbers and offsets were observed in mpengine.dll version 1.1.23070.1005.*

Note 3: Addition information about magic numbers.

4.1.6 SAC Operating Mode

SAC offers the flexibility to operate in three distinct operating modes, which are deactivated, evaluation, and enforcement. These modes can be configured and managed through manual or automatic means. Manual control involves the user interacting with the system settings, typically accessible through the Smart App Control setting in Windows Security settings, as detailed in the configuration section (see section 5). Automatic control confers authority to Microsoft Defender Antivirus, enabling it to make autonomous decisions regarding the configuration of SAC. By default, SAC is set in the evaluation mode, and the automatic control retrieved from cloud-based analysis rules a possible decision to switch SAC either to deactivated mode or to enforcement mode. The dynamic decision-making procedure is dependent on real-time evaluation issued from Microsoft Defender Antivirus and various contextual factors exclusive to the cloud-based analysis. This procedure plays a central role in the SAC feature since it directly controls the status of the feature, especially with the possibility to deactivate it without any possibility to turn it back (except by reinstalling Windows 11).

It is important to emphasize how the configuration of SAC is designed. This one is split in two sides. On the kernel-mode side, the configuration of ci.dll is mainly based on its own registry values and WDAC policies. On the user-mode side, the configuration of Microsoft Defender Antivirus is independent and mainly based on its own internal configuration elements. Then, once the two entities of SAC are running (in user-mode and in kernel-mode), both configurations can be “synchronized”. More specifically, when there is no human interaction in the loop (what we call “automatic mode"), this is the Microsoft Defender Antivirus’s configuration which has authority on the system, and which is used to rule the one concerning SAC in ci.dll.

4.1.6.1 Different configurations for different elements of SAC

4.1.6.1.1 Kernel-mode configuration

The SAC feature in Windows operates at two levels: kernel-mode and user-mode. Similarly, the configuration of SAC applies at these two levels also. On the one hand, there is the ci.dll library related to the enforcement of WDAC policies. This includes:

1. SAC is enabled by loading a specifically tailored SAC related WDAC policy (storer under the System32\CodeIntegrity\CiPolicies).
2. Unique WDAC policies are maintained for the evaluation and enforcement operating modes, each explicitly defining the parameters and behaviors of SAC within its respective mode. It is worth noting that there is no dedicated WDAC policy for the deactivated mode, as deactivated implies that no SAC-related WDAC policy is loaded.
    a. PolicyGUID: {0283AC0F-FFF1-49AE-ADA1-8A933130CAD6} - Enforce SAC policy, activated when SAC is loaded to enforcement mode.
    b. PolicyGUID: {1283AC0F-FFF1-49AE-ADA1-8A933130CAD6} - Evaluate SAC policy, activated when SAC is loaded to evaluation mode.

Notably, ci.dll also considers the values stored under the "HKLM\SYSTEM\CurrentControlSet\Control\CI\Policy" registry key. In particular, the `VerifiedAndReputablePolicyState` value is used to define the current mode of SAC (`0` deactivated, `1` enforcement, and `2` evaluation, see also Table 4) according the Microsoft's documentation (Microsoft). The second value called `VerifiedAndReputablePolicyStateMinValueSeen` is used for checking purpose of the first value.

The primary responsibility of the `VerifiedAndReputablePolicyStateMinValueSeen` value is to prevent the `VerifiedAndReputablePolicyState` from surpassing a specific value. By doing so, it maintains the `VerifiedAndReputablePolicyState` within well-defined limits, ensuring that the value represents transitions applicable in a single direction, moving from a higher value (e.g., `2` for evaluation mode) to a lower one (e.g., `0` for deactivated mode). This control mechanism safeguards the integrity and the consistency of SAC's operation mode, preventing undesirable shifts in its configuration.

> ***Note**: It is important to emphasize that the keys can only be influenced from different system state contexts (i.e., during boot, kernel initialization, etc.). For example, the* `VerifiedAndReputablePolicyStateMinValueSeen` *value can only be influenced when the system is booting. When the system is running, the key is directly protected by the kernel of Windows (with a dedicated procedure registered in ci.dll) to ensure that it is not possible to arbitrary change the value across reboots. At the opposite, the* `VerifiedAndReputablePolicyState` value *can be changed at any time considering the change is performed with appropriate rights (i.e., in ci.dll).*

Note 4: Additional information regarding when SAC registry keys can be modified.

Since WDAC policies are initially loaded by the Winload component (as described in (ERNW GmbH), section 2.2.1) during the system boot, this registry values are also taken into consideration. There are many points where these two values are evaluated during the initialization of Windows (Winload and ci.dll) to finally activate the SAC feature in the kernel.

For instance, when SAC is configured in evaluation mode during system boot, both the `VerifiedAndReputablePolicyState` and `VerifiedAndReputablePolicyStateMinValueSeen` registry values are initially set to the same value, which is `2`. This configuration triggers the loading of the evaluation mode WDAC policy (PolicyGUID: {`1283AC0F-FFF1-49AE-ADA1-8A933130CAD6`}).

However, when the evaluation mode is altered during a subsequent reboot, such as transitioning to enforcement mode, Winload synchronizes the value of `VerifiedAndReputablePolicyStateMinValueSeen` with `VerifiedAndReputablePolicyState` before the loading of the enforcement WDAC policy. As a result, both keys have the same value again, (namely `1`).

Considering the case where there would be an attempt to revert to evaluation mode following a reboot, `VerifiedAndReputablePolicyState` would have a higher value than `VerifiedAndReputablePolicyStateMinValueSeen`, which is not allowed (moving from a lower value (e.g., `1` for enforcement mode) to a higher one (i.e., `2` for evaluation mode)). In such a configuration,

only the value of `VerifiedAndReputablePolicyStateMinValueSeen` would be considered, and the `VerifiedAndReputablePolicyState` value would be aligned with the previous value. This procedure helps maintain the consistency of SAC operating modes. This procedure matches the 'buddy' principle, where one component safeguards the configuration values of the other.

4.1.6.1.2 User-mode configuration

On the other side, we have the "user-mode component of SAC", namely Microsoft Defender Antivirus if we take abstraction of the crytpcatsvc service used as an RPC proxy. The configuration of Microsoft Defender Antivirus is initialized by the `MpConfigInitialize` function in mpclient.dll.

Practically, this means invoking the `MpConfig::ConstructConfigKeySchemes` function to set up the internal structures representing the configuration loaded in memory. This function is responsible to initialize the Microsoft Defender Antivirus configuration based on the Windows' registry, system environment variables, values hardcoded within Microsoft Defender Antivirus binaries and with the msmplics.dll[11].

With all this different sources of information, Microsoft Defender Antivirus is about to load its own configuration in memory. The procedure is complex, and it covers different kinds of elements in an antivirus, such as the quarantine directory to use or the list of allowed applications registered for a given protected folder. From a technical point of view, the structure used to store the Microsoft Defender Antivirus's configuration can be considered as a "tree". There is a hierarchical organization of the data to regroup configuration elements belonging to the same concept (for instance, configurations concerning IP addresses or folder paths can be regrouped under the "Exclusions" concept to allow exception in analysis). By abuse of terminology, one can see such a configuration structure as structure of "folders" and "subfolders" where the different elements belonging to the same configuration topic would be regrouped. As a side note, it is worth noting that access to sub-configurations is internally done with backslash separators, as with directories in Windows explorer (for instance, the "Windows Defender Exploit Guard\Network Protection" configuration). An element of configuration can be anything, from a numerical value or a string of characters to a full structure composed of different kinds of data.

This internal configuration structure could be anecdotic by itself, but Microsoft Defender Antivirus's uses it extensively to calibrate its actions, especially to know which kind of operation it must (or must not) run and specifically when running an analysis, how and with which means the analysis must be run. Access to this configuration tree is performed with a set of "setter" and "getter" (i.e., `MpConfigSetValue` and `MpConfigGetValue`) functions used to respectively write and read to the configuration tree.

The Figure 12 represents a part of the configuration tree concerning different elements of the configuration of the SAC feature. Other configuration elements may be involved in the analysis of SAC since a great part of the analysis is common to a lot of analysis submission in Microsoft Defender Antivirus. In the Figure 12, we represent some of the elements composing the configuration of Microsoft Defender Antivirus. The configuration tree is much bigger than the one presented in Figure 12, and it contains many more elements. All these elements are linked to the "root" configuration called "." at the top of the general configuration tree. From this point, there are two kinds of configuration entries. On the one hand, the sub-folders which are literally sub-branches in the configuration tree. On the other hand, relative to sub-folders (considering the root "." is also a sub-folder), there are configuration elements providing a real configuration content, such as a numerical value or a string of characters, for instance.

But the elements which matters the most in SAC are provided in the Figure 12, linked to the root of the tree. It matters to mention `SacEvalModeExpirationTime`, `SacLearningModeSwitch`,

[11] *The rationale behind the utilization of this library for certain configuration parameters is to exercise control over the features made available within Microsoft Defender Antivirus, particularly in conjunction with the Windows licensing model. Once this library file has been loaded into memory, configuration information is extracted from the file. For instance, the* `ProductGUID` *is* `77BDAF73-B396-481F-9042-AD358843EC24` *in our case.*

`SmartLockerMode`, `VerifiedAndReputableTrustModeEnabled`, and `HybridModeEnabled`. We propose to detail them as follows:

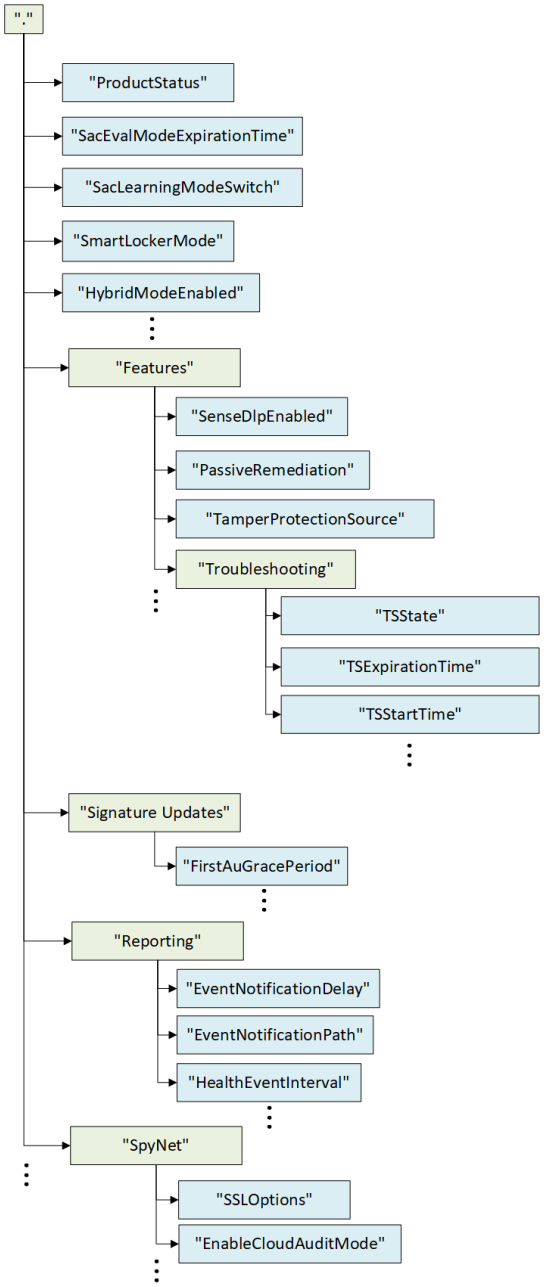

Figure 12. Illustration of the internal structure used by Microsoft Defender Antivirus to store its configuration. The tree structure has main entries (in green) and sub-entries (in blue) containing configuration values.

- `SmartLockerMode`: The value of this configuration element is either true or false, and its purpose is to indicate whether SAC is currently in enabled or deactivated state.
- `VerifiedAndReputableTrustModeEnabled` & `HybridModeEnabled`: These two configuration entries are both Boolean values used to inform Microsoft Defender Antivirus about the active operating mode of SAC. They are retrieved through the wldp.dll library, asking policy configuration to the kernel. When `VerifiedAndReputableTrustModeEnabled` is set to true, it indicates that a SAC-specific WDAC policy is active and loaded, with a setting called `VerifiedAndReputableTrustMode` set to `true`. This informs Microsoft Defender Antivirus that SAC is in evaluation mode because only the evaluation policy has this value set (see section 4.1.6.1.1 policy). At the opposite, if both `VerifiedAndReputableTrustModeEnabled` and `HybridModeEnabled` configuration values are

set to true, it informs Microsoft Defender Antivirus that SAC is operating in enforcement mode. In this case, it means that a SAC-specific WDAC policy is active and loaded, with both `VerifiedAndReputableTrustMode` and `VerifiedAndReputablePerfMode` settings set to `true`.

Among the different possible values linked to these two configuration variables [`VerifiedAndReputableTrustModeEnabled`, `HybridModeEnabled`], we have:

- [true, true]: SAC is enabled, and the enforcement mode WDAC policy is loaded.
- [true, false]: SAC is enabled, and the evaluation mode WDAC policy is loaded.
- [false, true]: Not defined/not used.
- [false, false]: SAC is deactivated, meaning no WDAC policy is loaded.

- `SacLearningModeSwitch`: This configuration value holds the most important information to control SAC's behavior. It operates on a simple principle: `0` signifies no action, `1` implies the disabling of SAC, and `2` triggers the transition of SAC into the enforcement mode. In essence, it serves as a control switch for SAC, allowing Microsoft Defender Antivirus to make specific choices. A value of `0` means SAC remains unaffected, `1` disengages the SAC functionality, and `2` shifts SAC into enforcement mode, enhancing security measures.
- `SacEvalModeExpirationTime`: The purpose of this configuration value is to regulate the remaining duration time during which SAC remains in evaluation state.

4.1.6.2 Dynamic Decision-Making Procedure

Automatic control of SAC's mode confers authority to Microsoft Defender Antivirus to take autonomous decisions regarding the configuration of SAC. The configuration values listed above exhibit dynamic changes throughout the operational lifecycle of Microsoft Defender Antivirus. Specifically, these changes begin when Microsoft Defender Antivirus is initialized for the first time (i.e., the system is started after the installation of Windows), and SAC is configured in learning mode. Subsequently, considerations are given to deactivate SAC or its transition into enforcement mode, whereby these deterministic elements have a direct impact on the configuration.

After the installation of the Windows operating system and the initial launch of Microsoft Defender Antivirus, SAC is configured to operate in learning mode within the system. In practical terms, this initiation unfolds as follows:

- `SmartLockerMode` configured to [`1`]: By default, set to `1` at the initiation of the Microsoft Defender Antivirus service.
- `VerifiedAndReputableTrustModeEnabled` & `HybridModeEnabled` configured to [`1`; `0`]: Configured to `1` and `0` in accordance with the loaded dedicated SAC WDAC policy.
- `SacLearningModeSwitch` configured to [`0`]: Initially set to `0` during the initiation of Microsoft Defender Antivirus when the tree-like configuration structure is initialized.
- `SacEvalModeExpirationTime` configured to [`0`]: Initially set to `0` during the initiation of Microsoft Defender Antivirus when the tree-like configuration structure is initialized.

As previously mentioned, these initial values are subject to adjustments throughout the operation of Microsoft Defender Antivirus. In practical terms, alongside the operation of learning mode, there is a possibility to transition from evaluation mode to enforcement or deactivated mode. This transition reflects Microsoft Defender Antivirus's adaptive nature, where it initially acquires knowledge and then it employs that knowledge to implement more rigorous security measures, as needed. However, the change in values regarding the transition from evaluation mode to enforcement or deactivated modes can be further narrowed down through two different cases:

Modifications occur in response to a time interval: In Microsoft Defender Antivirus's configuration, a crucial aspect matters for the timing of evaluation. This temporal orchestration is achieved through the utilization of the `CreateTimerQueueTimer` function, which plays a central role in managing the scheduling and the execution of the `MpService::CHeartbeatManager::OnHeartbeat` callback function. The function is invoked after a predetermined interval of time has elapsed. This interval is fully configurable and it depends on the value stored under the path "./Reporting/HealthEventInterval" in the Microsoft Defender Antivirus's configuration tree.

The "./Reporting/HealthEventInterval" is expressed as a DWORD value, typically initialized by default to 60. This value is essential in determining the expected duration time before the callback function is called. To derive the time in milliseconds, the value configured for this entry is multiplied by 60000, subsequently translating into the elapsed time required for the execution of the callback function. Consequently, this mechanism enables Microsoft Defender Antivirus to effectively time its operations, thereby ensuring timely evaluations and responses based on the configured "./Reporting/HealthEventInterval".

In SAC, this procedure ultimately leads to the execution of the `MpService::CHeartbeatManager::SACEvalModeOnHeartbeat` function. This function is directly involved in the management of the `SacLearningModeSwitch` configuration element. Its principal goal is to dynamically adjust this configuration element based on a specific time interval beginning when Microsoft Defender Antivirus is initialized for the first time.

The assessment of whether the designated time interval elapsed begins after a duration of 1.25 days and concludes after 45 days. It means that before 1.25 days after the initial installation of the operating system, it is not necessary for Microsoft Defender Antivirus to expect feedback concerning a potential switch mode update for SAC. The same way, the by default upper limit of the evaluation time is fixed to 45 days. It means that if there is no feedback concerning the SAC status coming for the cloud-based analysis, the SAC feature will be automatically deactivated. If it is considered as necessary by the Microsoft's cloud-based backend, the duration of the evaluation can be extended through the remote update of the `SacEvalModeExpirationTime` configuration element. This mechanism ensures the adaptive control of SAC's learning mode over time.

Modifications are instigated by the Microsoft cloud-based backend: The Microsoft cloud-based backend has the capability to modify configuration elements, as necessary (see Note 2). This control is exercised through dynamic signatures in SAC (see section 4.1.5.2). This modification has a dual effect:

On the one hand, it can impact the configuration entry `SacLearningModeSwitch`. This adjustment occurs during the parsing of the dynamic signature. If a chunk within the dynamic signature contains the magic number `0x55`, it triggers the execution of the function `SendSacLearningModeNotification`, which invokes the function `MpHandleSacLearningMode` through a callback mechanism. In this end, this results in the alteration of the `SacLearningModeSwitch` data value to match the value transmitted from the Microsoft's cloud-based backend.

On the other hand, it can also affect the configuration entry `SacEvalModeExpirationTime`. Like the previous scenario, this modification takes place during the parsing of the dynamic signature. When a chunk bearing the magic number `0xCE` is present, the `SendSacRemainInEvalNotification` function is triggered, subsequently invoking the `MpHandleSacRemainInEval` function using the same callback mechanism. This leads to the adjustment of the `SacEvalModeExpirationTime` data value to match the value transmitted from the Microsoft's cloud-based backend.

The synchronization between Microsoft Defender Antivirus, as the entity wielding decision-making regarding changes in SAC's operational mode, and the ci.dll component, which manages the kernel-mode configuration side, is a crucial element in the automated control of SAC's mode. This collaborative procedure requires a streamlined communication channel, ensuring that Microsoft Defender Antivirus effectively communicates its decisions regarding configuration alterations. Once receiving this communication, ci.dll interprets the directives and it implements the required modifications according to preestablished procedures. Following the implementation of the configuration changes, ci.dll provides feedback to confirm the actions taken. This ensures that the new configuration is accurately acknowledged and duly considered by Microsoft Defender Antivirus. This synchronization mechanism is vital for maintaining an automated and coherent control over SAC's operational modes. Figure 14 illustrates the procedural flow, wherein Microsoft Defender Antivirus makes a configuration decision and where it communicates this decision to ci.dll. Subsequently, ci.dll actively implements the configuration changes and provides feedback back to Microsoft Defender Antivirus. This illustration shows the sequential steps involved in the communication and configuration process between Microsoft Defender Antivirus and ci.dll.

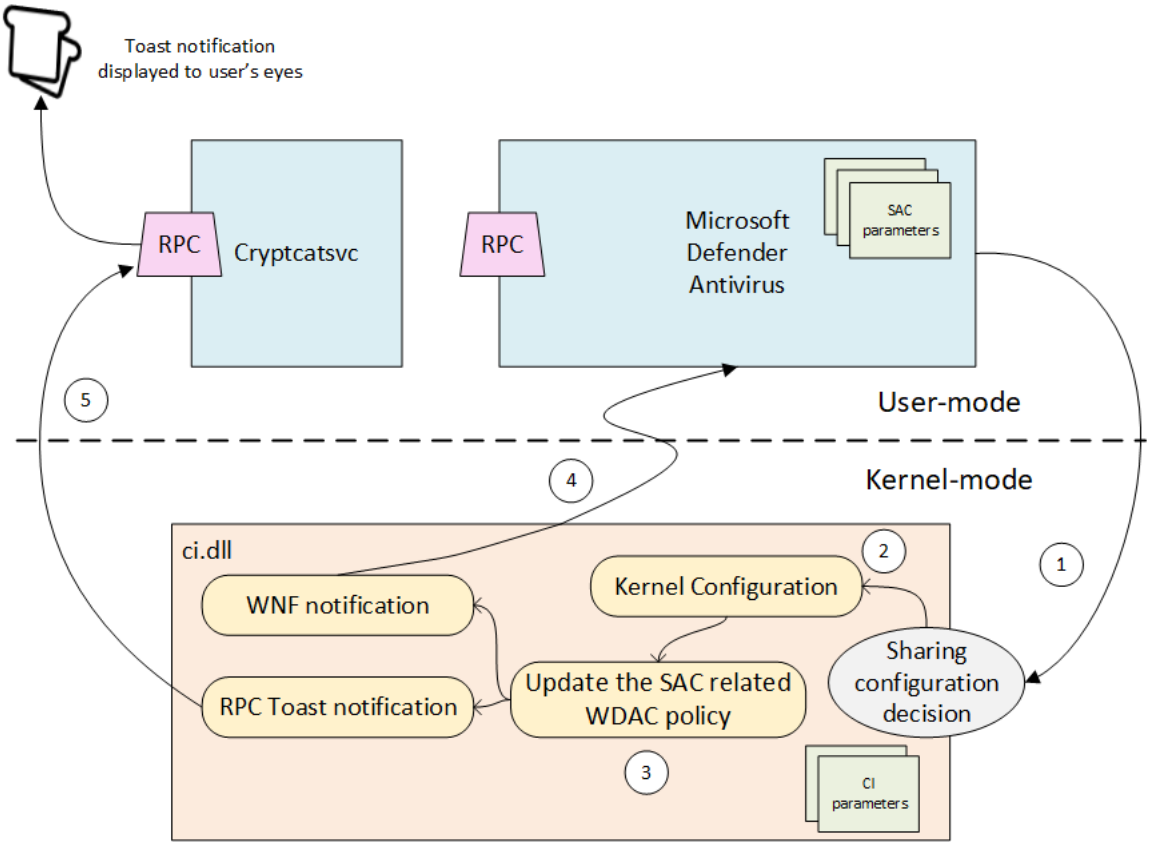

Figure 13: Illustration of the dynamic decision-making procedure

The following discusses the fundamental activities illustrated in Figure 14, outlining their sequential progression in the transition from evaluation to enforcement mode. It must be emphasized that the procedure for transitioning to deactivated mode is conceptually identical, and for this reason, it will not be explicitly covered in the following discussion.

1. Sharing the decision: Microsoft Defender Antivirus bases its decisions on two main scenarios: either triggered after the deadline of SAC evaluation or at the initiative of the Microsoft's cloud-based backend in the response of an analysis request. The critical factor here is the ultimate value of `SacLearningModeSwitch`, as it controls SAC's behavior concerning its configuration. To reiterate, the value of `0` maintains the status quo for SAC, while the value `1` switches SAC into deactivated mode, and the value `2` switches SAC into enforcement mode. The outcome of the

decision-making process is conveyed to ci.dll whenever an analysis is triggered by ci.dll through the `CiCatDbSmartlockerDefenderCheck` function (see section 4.1.1). This implies that the value of `SacLearningModeSwitch` must be correctly considered, and the relevant information is passed to ci.dll as an output parameter when the function concludes and returns.

2. Configuring the kernel side: After the successful return of the `CiCatDbSmartlockerDefenderCheck` function, the output parameter reflecting information about `SacLearningModeSwitch` is assessed within the `CiHandleDefenderSignals` function. In SAC's configuration, this function encompasses two core responsibilities: adjusting registry values and instigating the update of the WDAC policy in alignment with the determined decisions. For example, when the transition to enforcement mode is required, it configures the registry value `VerifiedAndReputablePolicyState` to 1 and it initiates the loading of the WDAC enforcement policy.
3. Update the SAC related WDAC policy: To update the SAC-related WDAC policy, the function `CiAsyncPolicyRefreshRoutine` is invoked. This function calls the `CiUpdatePolicies` function, responsible for transitioning the existing active SAC WDAC policy (i.e., the evaluation policy, as outlined in section 4.1.6.1.1) to the SAC enforcement WDAC policy.
4. Update the Microsoft Defender Antivirus configuration's parameters: Once the policy has been successfully updated, the `CiAsyncPolicyRefreshRoutine` function signals Microsoft Defender Antivirus to update its configuration as well, ensuring synchronization with the kernel configuration side. This update occurs through an indirect communication method utilizing the Windows Notification Facility (WNF) API. Whenever the `CiUpdatePolicies` function updates a WDAC policy, a WNF notification is triggered (specifically, `WNF_CI_CODEINTEGRITY_MODE_CHANGE`). Microsoft Defender Antivirus actively monitors this WNF notification, allowing it to update the two configuration values `VerifiedAndReputableTrustModeEnabled` and `HybridModeEnabled`. For instance, when the SAC related WDAC policy is updated to the enforcement policy, Microsoft Defender Antivirus retrieves the values through the wldp.dll library (see section 4.1.6.1.2) and it adjusts them accordingly. In this last case, it would mean that `VerifiedAndReputableTrustModeEnabled` and `HybridModeEnabled` are finally set to 1.
5. Informing the user (in parallel to the 4th action): Only for the purpose of displaying information to the user, the `CiAsyncPolicyRefreshRoutine` function notifies the user of a configuration change by using the RPC interface provided by the cryptcatsvc service (see section 3.3.4). For example, when transitioning to enforcement mode, the kernel calls the `CiCatDbSendSmartAppControlSwitchEnforceToast` function to finally notifies the RPC `s_SSCatDBSendSmartAppControlSwitchEnforceToast` function. The same way, this is the `CiCatDbSendSmartAppControlBlockToast` function, which is called from kernel mode, notifying in the end the RPC's `s_SSCatDBSendSmartAppControlBlockToast` function in cryptcatsvc service. Consequently, this function triggers the display of a notification containing pertinent information about the configuration update.

4.1.7 SAC Dynamic Signature Persistence

Microsoft Defender Antivirus makes the distinction between static and dynamic signatures. As the term implies, static signatures, are preloaded into the Microsoft Defender Antivirus's signature database[12]. These signatures are the main object used for identifying known malware threats. In contrast, dynamic signatures are generated or acquired on-the-fly during runtime to address emerging or specialized threats. By

[12] *The Microsoft Defender Antivirus's signature database (located at 'C:\ProgramData\Microsoft\Windows Defender\Definition Updates') contains vital threat detection resources. Within this directory, files with the '.vdm' extension house a collection of multiple files named 'modules'. These modules serve as repositories for essential malware signatures and patterns.*

categorizing signatures in this manner, Microsoft Defender Antivirus service efficiently manages protection against well-known but evolving threats.

The signatures obtained through the assessment conducted by the cloud-based security service are categorized as dynamic signatures. These dynamic signatures diverge from the traditional approach used for managing static signatures, as they are not stored within the Microsoft Defender Antivirus's signature database. Instead, each dynamic signature resides in the file system as single file, typically located at C:\ProgramData\Microsoft\Windows Defender\Scans\RtSigs.[13]

This means that whenever a signature has been received and evaluated as valid, it is then stored in a file using the `FpPersistBlob` function. Doing so, a custom algorithm is used for encoding the dynamic signature. The core within the algorithm is the use of a lookup table. This table likely maps input values (as bytes) to specific output values (as bytes also). For example, the byte at the current position pointed to by the `pByte` pointer is used as an index into this table, and the value stored at that index is retrieved. The value retrieved from the lookup table is then written back to the memory location pointed to by the `pByte` pointer. This effectively replaces the original byte pointed to by the `pByte` pointer with the value from the lookup table. In summary, this algorithm is used to encode or decode (see Code Block 8 and Code Block 9) the signature data blob by looking up each byte of the data blob in a table, replacing it with the corresponding value from the table, and repeating this process for all bytes in the data blob.

```
000000075AEE46B0  C9 69 9D A8 21 63 5E B0  5A 22 53 C5 5C 25 9F A9  Éi.¨!c^°Z"SÅ\%Ÿ©
000000075AEE46C0  B9 1C 8A 59 3D 4B A4 0C  BD 27 D2 92 6B 79 66 90  ¹.ŠY=K¤.½'Ò'kyf.
000000075AEE46D0  F9 12 4F DC 65 7C 8F B5  C0 E0 5B 76 6F FB FE 78  ù.OÜe|.µÀà[voûþx
000000075AEE46E0  33 9A 74 88 E5 29 D8 41  BF 4A EC F7 71 5D 4E 0A  3štˆå)ØA¿Jì÷q]N.
000000075AEE46F0  35 E3 14 05 D7 FC 04 77  A3 17 F6 00 E7 AD F8 6A  5ã..×ü.w£.ö.ç-øj
000000075AEE4700  67 46 93 C7 0F 89 44 ED  82 DE E2 C8 FA 38 72 C2  gF"Ç.‰Dí‚ÞâÈú8rÂ
000000075AEE4710  60 BC 99 31 13 80 C3 3E  18 F2 CE E4 EA 73 3F AC  `¼™1.€Ã>.òÎäês?¬
000000075AEE4720  E6 91 0B D1 1E DA 19 1A  D0 A7 52 AE 36 55 2A CB  æ'.Ñ.Ú..Ð§R®6U*Ë
000000075AEE4730  EE CC 5F A2 06 49 97 EB  FD D4 6D 87 16 4D 8B 75  îÌ_¢.I—ëýÔm‡.M‹u
000000075AEE4740  FF 81 85 57 34 C6 23 09  6E F0 B1 61 56 2E 4C 2B  ÿ.…W4Æ#.nð±aV.L+
000000075AEE4750  B3 B6 42 02 D5 7A 45 EF  08 C1 3A 8D D3 0D 98 9C  ³¶B.ÕzEï.Á:.Ó.˜œ
000000075AEE4760  43 7F 1D 58 D9 AB E9 A0  7D 48 01 7B DD 62 84 BA  C..XÙ«é }H.{Ýb„º
000000075AEE4770  03 54 DB B8 64 C4 F5 39  B4 83 E1 94 2F 70 50 A5  .TÛ¸dÄõ9´ƒá"/pP¥
000000075AEE4780  B7 2C 9E 1F 24 A6 CF 7E  40 B2 AA 30 20 10 AF 47  ·,ž.$¦Ï~@²ª0 .¯G
000000075AEE4790  F4 2D 95 A1 3B 68 3C 96  86 DF 8C BE F1 07 E8 0E  ô-•¡;h<-†ßŒ¾ñ.è.
000000075AEE47A0  CA 15 CD 37 6C 32 26 BB  8E 11 D6 28 1B 9B 51 F3  Ê.Í7l2&»Ž.Ö(.›Qó
```

Code Block 8: Encoding lookup table (g_PatternEncodingTable).

```
000000075ADCAE10  4B BA A3 C0 46 43 84 ED  A8 97 3F 72 17 AD EF 54  K°£ÀFC„í¨—?r.ïT
000000075ADCAE20  DD F9 21 64 42 F1 8C 49  68 76 77 FC 11 B2 74 D3  Ýù!dBñŒIhvwü.²tÓ
000000075ADCAE30  DC 04 09 96 D4 0D F6 19  FB 35 7E 9F D1 E1 9D CC  Ü..–Ô.ö.û5~ŸÑá.Ì
000000075ADCAE40  DB 63 F5 30 94 40 7C F3  5D C7 AA E4 E6 14 67 6E  Ûcõ0"@|ó]Çªäæ.gn
000000075ADCAE50  D8 37 A2 B0 56 A6 51 DF  B9 85 39 15 9E 8D 3E 22  Ø7¢°V¦Qß¹…9.ž.>"
000000075ADCAE60  CE FE 7A 0A C1 7D 9C 93  B3 13 08 2A 0C 3D 06 82  Îþz.Á}œ"³..*.=.‚
000000075ADCAE70  60 9B BD 05 C4 24 1E 50  E5 01 4F 1C F4 8A 98 2C  `›½.Ä$.På.O.ôŠ˜,
000000075ADCAE80  CD 3C 5E 6D 32 8F 2B 47  2F 1D A5 BB 25 B8 D7 B1  Í<^m2.+G/.¥»%¸×±
000000075ADCAE90  65 91 58 C9 BE 92 E8 8B  33 55 12 8E EA AB F8 26  e'XÉ¾'è‹3U.Žê«ø&
000000075ADCAEA0  1F 71 1B 52 CB E2 E7 86  AE 62 31 FD AF 02 D2 0E  .q.RËâç†®b1ý¯.Ò.
000000075ADCAEB0  B7 E3 83 48 16 CF D5 79  03 0F DA B5 6F 4D 7B DE  ·ãƒH.ÏÕy..ÚµoM{Þ
000000075ADCAEC0  07 9A D9 A0 C8 27 A1 D0  C3 10 BF F7 61 18 EB 38  .šÙ È'¡ÐÃ.¿÷a.ë8
000000075ADCAED0  28 A9 5F 66 C5 0B 95 53  5B 00 F0 7F 81 F2 6A D6  (©_fÅ.•S[.ð..òjÖ
000000075ADCAEE0  78 73 1A AC 89 A4 FA 44  36 B4 75 C2 23 BC 59 E9  xs.¬‰¤úD6´uÂ#¼Yé
000000075ADCAEF0  29 CA 5A 41 6B 34 70 4C  EE B6 6C 87 3A 57 80 A7  )ÊZAk4pLî¶l‡:W€§
000000075ADCAF00  99 EC 69 FF E0 C6 4A 3B  4E 20 5C 2D 45 88 2E 90  ™ìiÿàÆJ;N \-Eˆ..
```

Code Block 9: Decoding lookup table (g_PatternDecodingTable).

[13] *Storing frequently used data in a persistent cache can significantly improve application performance by reducing the need to retrieve data from slower sources, such as cloud-based security service.*

5 Configuration and Logging Capabilities

5.1 Configuration Capabilities

> ***Note**: The functionality of SAC is based on a variety of technologies that may not be immediately identifiable as integral components. Consequently, the configuration of the set of technologies involved in SAC can each have a direct influence on SAC. This section exclusively provides configurations that have a direct correlation. Configurations that may have an impact on SAC but are not identifiable as such due to their association with another technology are outside the scope of this study. For instance, other WDAC policies definition in the context of Code Integrity.*

This section discusses the available configurations for SAC. It must be emphasized, that unlike certain features within Microsoft Defender Antivirus, the centralized management of SAC through group policies is not possible. The available settings reside within the *Smart App Control* settings, accessible via the *Windows Start button → Settings → Update & Security → Windows Security → App & browser control*. Figure 15 illustrates the configuration options that are available via the *Smart App Control* settings.

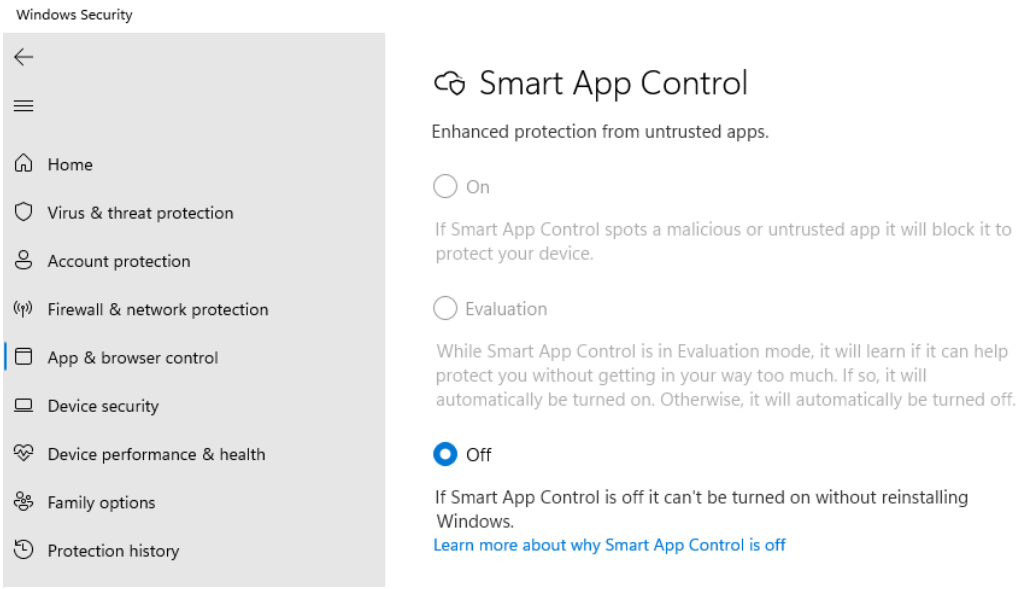

Figure 15: Smart App Control setting

Enforcement (On): This option signifies the evaluation phase has approved the full activation of SAC. SAC no longer operates in its evaluation mode, wherein the execution of processes is only audited without any actively blocking. There is no mandatory prerequisite for going through the evaluation phase prior to transitioning into the active mode, even if it is recommended to keep the evaluation phase until it finishes. Administrators have the autonomy to manually switch from evaluation mode to the active mode, bypassing the possibility to complete the evaluation phase[14].

Evaluation: During the evaluation phase, SAC enters in a mode dedicated to the assessment of the suitability of the Windows system for SAC adoption. After successfully passing this assessment, the system automatically switches to active mode. It is important to note that a system under evaluation does not block processes, but it only audits them. In scenarios where the evaluation phase identifies processes that are unsuitable for SAC or it anticipates potential issues arising from implementation, SAC will be deactivated.

Deactivated (Off): SAC will be switched off (deactivated) if it would cause disruptions of applications used by the user or the system is not suitable for SAC protection (i.e., when evaluation phase is not completed

[14] After the evaluation phase concludes, or if SAC is manually switched on or off, reverting to the evaluation mode is not possible unless Windows is reinstalled or reset to zero.

successfully). Additionally, if SAC is about to run on a Windows not issued from a fresh installation ("out of the box"), SAC is also turned off.

Registry implementation: Table 4 presents the registry implementation of the *Smart App Control settings* configuration options.

Name	Value
Registry hive	HKEY_LOCAL_MACHINE
Registry path	SYSTEM\CurrentControlSet\Control\CI\Policy
Value name	VerifiedAndReputablePolicyState
Value type	REG_DWORD
Evaluation mode	2
Enforcement mode	1
Deactivated mode	0

Table 4: Registry implementation of Smart App Control settings.

5.2 Logging Capabilities

5.2.1 Event Tracing for Windows

Event Tracing for Windows (Microsoft) can effectively capture and log activities associated with SAC. Table 5 lists the manifest based ETW providers, whereas Table 6 provides a list of trace logging ETW providers integrated into Microsoft Defender Antivirus.

ETW Provider	GUID
0A002690-3839-4E3A-B3B6-96D8DF868D99	Microsoft-Antimalware-Engine
E4B70372-261F-4C54-8FA6-A5A7914D73DA	Microsoft-Antimalware-Protection
11CD958A-C507-4EF3-B3F2-5FD9DFBD2C78	Microsoft-Windows-Windows Defender
8E92DEEF-5E17-413B-B927-59B2F06A3CFC	Microsoft-Antimalware-RTP
751EF305-6C6E-4FED-B847-02EF79D26AEF	Microsoft-Antimalware-Service
CFEB0608-330E-4410-B00D-56D8DA9986E6	Microsoft-Antimalware-AMFilter

Table 5: Manifest based ETW providers of Microsoft Defender Antivirus.

ETW Provider	GUID
05eec406-6e71-5f61-e1ea-0f2a6d7e78ba	Microsoft.Defender.EndpointDlp
0be29c0b-0729-534d-0c1d-5fad98cc6118	Microsoft.Defender.EndpointDlp.Tracelog
39bd9805-3945-4878-aecc-096a23369f68	Microsoft.Windows.SmartScreen
65a1b6fc-4c24-59c9-e3f3-ad11ac510b41	Microsoft.Windows.Sense.Client
6d1b249d-131b-468a-899b-fb0ad9551772	TelemetryAssert
703fcc13-b66f-5868-ddd9-e2db7f381ffb	Microsoft.Windows.TlgAggregateInternal
7af898d7-7e0e-518d-5f96-b1e79239484c	Microsoft.Windows.Defender
af2ae1c8-cf6d-4268-8159-fcce3c2e67db	TelemetryAssertDiagTrack
bf94eeee-f654-4baa-9f8a-7f9cb446e18e	Microsoft.Windows.Defender.TeTest

Table 6: Trace logging based ETW providers of Microsoft Defender Antivirus.

5.2.2 File logging capabilities

In the initialization of Microsoft Defender Antivirus, there is a procedure to initialize text files containing logs providing a detailed view over the actions performed by Microsoft Defender Antivirus. In particular, the `MiscConfig::InternalMiscConfigInitialize` function (in MpClient.dll) is responsible to hold the entry point of this procedure.

At the beginning of this function, there is a call to the `MpLoggingOpen` function calling itself the `MpLog::CMpLogging::CmpLogging` function. This functions first starts by crafting the log path holding all log files used by Microsoft Defender Antivirus. The generated path depends on the string stored in the "ProductAppDataPath" configuration value (it corresponds to the folder location where Microsoft Defender

Antivirus is currently executed) concatenated with "\Support" as a subfolder name. Once the path is crafted, the directory access is guaranteed. Then, the `MpLogginOpen` function search for a file whose name is starting by “MPLog-”. This function iterates over files whose extension is “*.log” in a directory, checking for outdated ones based on their last modification date. By default, it represents 2.592.000 seconds, corresponding to 30 days. Also, there is a check about the size of the log file fixing the maximum size based on the "`SupportLogMaxSize`" configuration value, which is by default set to 0x2000000 bytes (32 Mb).

Once the log file has been found, the filename of the log file is generated. This one follows the follows template: “C:\ProgramData\Microsoft\Windows Defender\Support\<PrefixLogFile>YYYYMMDD-HHMMSS.log” where “<PrefixLogFile>” corresponds to the prefix log file name (“MPLog-” in our case) and “YYYYMMDD-HHMMSS” describe the date and time where the file has been generated. Once the file name is generated, this one is created on the disk. The same applies for "MPDetection-" and "MPDeviceControl-" log files.

Once the `MpLoggingOpen` function has finished, the `MpLogServiceStart` function is executed. This one initializes first an internal buffer (called a “session”) flushed (with the `MpLogWriteSession` function) from time to time to the log file, for performance reasons. The function used to write inside a log file is the `MpLogMessage` function. After the initialization of this log procedure, we observe the use of the `MpLogMessageWithTime` function to log the content of some configuration values, among other things.

6 Final Remarks

6.1 Main considerations about SAC

More than the technical considerations, the internal documentation of SAC performed in this project raises two important questions. On the one hand, there is a real concern about the privacy of user's information. On the other hand, there is a real concern about the position of Microsoft Defender Antivirus in the Windows 11 operating system.

At first, concerning data disclosure, SAC is nothing but an automatic notification mechanism to the Microsoft cloud-based backend. It discloses metadata about the executable files executed on the user's system but also information on the system itself (from Microsoft Defender Antivirus' version and configuration to the operating system). This is not something new that Microsoft has telemetry capacities (ERNW GmbH) but this is another means for information disclosure. In the case where another antivirus would be present in the system (side-by-side mode) or Microsoft Defender Antivirus would have been set in passive mode, SAC is still active, and it conserves some cloud-based analysis capabilities. In Microsoft's documentation concerning SAC (Microsoft), it is directly written SAC works even if there is already an antivirus in the system.

> *"Smart App Control works alongside your other security software, such as Microsoft Defender or non-Microsoft antivirus tools, for added protection."*

It means that even though the user may have chosen another antivirus software (for any reason) or to turn Microsoft Defender Antivirus into passive mode, this last one is still present in the system, and it still has notification from the kernel for analysis. We also note the necessity to turn on the optional diagnostic data (ERNW GmbH) in Windows 11 to activate SAC. This raises a question of control over the security tools available in the Windows environment, and the choice left to the organization or the respective user.

Then, concerning the position of SAC in the system, this one is far from being trivial. An antivirus is usually a standalone component in the system. It means it can be installed and removed from the system without impacted the general behavior of the last. An antivirus takes its information from notification performed by a dedicated driver (usually a mini-filter driver registering a callback via the `PsSetLoadImageNotifyRoutine` function to be notified for each new executable image loaded into memory). It means this driver is a standalone component that can be added to and removed from the system. In the case of SAC, this one is directly embedded in the kernel, more specifically in ci.dll and not in one of the Microsoft Defender Antivirus's drivers which however uses which uses the callback registration API).

If the internal RPC notification from the kernel to Microsoft Defender Antivirus is triggered in ci.dll by the `CipExternalAuthorizationCallback` function, the "external" word in the function's name means Microsoft Defender Antivirus and no one else. There is no way to substitute Microsoft Defender Antivirus for another antivirus since the callback is directly hardcoded into the kernel. That way, SAC is a security feature which forces to use Microsoft Defender Antivirus, not as a standalone software as any other antivirus but as a key core component of Windows. Of course, it remains possible to deactivate SAC and Microsoft Defender Antivirus totally and manually, but this is a choice which is more and more "everything" or "nothing" for the organization or the respective user.

Considering the two questions together, SAC literally questions the position of Microsoft Defender Antivirus in the system. By linking it directly to the kernel without any substitution interface and by collecting a lot of information on the user's system, Microsoft Defender Antivirus may be considered as an unavoidable component in the Windows' environment. By deeply embedding SAC, that way, to the kernel and to Microsoft Defender Antivirus, this feature definitively raises the question of the current impossibility for potential alternatives proposed by competitors.

6.2 Further work

As a further work, it could be interesting to evaluate deeper different interactions of SAC concerning different scenarios, especially to measure with different notifications the reputation of diverse files submitted to Microsoft's cloud-based backend for analysis. The notion of digital signature would also deserve a deep evaluation in SAC, especially to know more precisely in which situation a file is considered with a good reputation. The cloud-based architecture prevents us from directly understanding the algorithm behind the scenes used to perform this evaluation.

Another point concerns the assessment of the automatic switch to enforcement mode, especially when the cloud-based analysis decides it. In such a context, it could be interesting to test different use cases scenario looking for the situations where a deactivation is required and when an enforcement is activated.

Finally, the security of the feature should be regarded, especially concerning potential fake notification performed from a rogue machine to Microsoft's cloud-based backend to deactivate SAC on another machine. After all, the notification is a single HTTPS request and if the security is correctly assessed on client side with a certificate pinning, nothing prevent the last to forge fake notification about random or untrusted applications in order deactivate SAC remotely. Such a work would definitively be interesting in a context of a security assessment.

Bibliography

Bar, Tomer und Omer Attias. „Defender - Pretender, When Windows Defender Updates Become a Security Risk.“ Las Vegas: Black Hat USA, 2023. <https://i.blackhat.com/BH-US-23/Presentations/US-23-Tomer-Defender-Pretender-final.pdf>.

ERNW GmbH. „SiSyPHuS Win10: Analyse von Device Guard.“ Work Package 7. Federal Office for Information Security, kein Datum. <https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Cyber-Sicherheit/SiSyPHus/Workpackage7_Device_Guard.pdf?__blob=publicationFile&v=1>.

—. „Telemetry (Version 1.0).“ Work Package 4. Federal Office for Information Security, kein Datum. <https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Cyber-Sicherheit/SiSyPHus/Workpackage4_Telemetry.html?nn=1011494>.

Forshaw, James. „The Definitive Guide on Win32 to NT Path Conversion.“ Google Project Zero, 29. 02 2016. <https://googleprojectzero.blogspot.com/2016/02/the-definitive-guide-on-win32-to-nt.html>.

Graeber, Matt. „Documenting and Attacking a Windows Defender Application Control Feature the Hard Way.“ *A Case Study in Security Research Methodology*. SpecterOps, 22. 06 2018. <https://posts.specterops.io/documenting-and-attacking-a-windows-defender-application-control-feature-the-hard-way-a-case-73dd1e11be3a>.

Luc Reginato, @_YouB_. „Updated Analysis of PatchGuard on Microsoft Windows 10 RS4.“ 2019. <https://blog.tetrane.com/downloads/Tetrane_PatchGuard_Analysis_RS4_v1.01.pdf>.

Microsoft. „/INTEGRITYCHECK (Require signature check).“ 14. 08 2023. <https://learn.microsoft.com/en-us/cpp/build/reference/integritycheck-require-signature-check?view=msvc-170>.

—. „5.1 NTFS Streams.“ 14. 12 2021. <https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-fscc/c54dec26-1551-4d3a-a0ea-4fa40f848eb3>.

—. „About Event Tracing.“ 01. 07 2021. <https://learn.microsoft.com/en-us/windows/win32/etw/about-event-tracing>.

—. „About Extensible Storage Engine.“ 07. 01 2021. <https://learn.microsoft.com/en-us/windows/win32/extensible-storage-engine/about-extensible-storage-engine>.

—. „About WinHTTP.“ 07. 01 2021. <https://learn.microsoft.com/en-us/windows/win32/winhttp/about-winhttp>.

—. „Antimalware Scan Interface (AMSI).“ 23. 08 2019. <https://learn.microsoft.com/en-us/windows/win32/amsi/antimalware-scan-interface-portal>.

—. „Antivirus API.“ 15. 08 2017. <https://learn.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/platform-apis/ms537365(v=vs.85)>.

—. „App capability declarations.“ 18. 08 2023. <https://learn.microsoft.com/en-us/windows/uwp/packaging/app-capability-declarations>.

—. „Application Control for Windows.“ 31. 08 2023. <https://learn.microsoft.com/en-us/windows/security/application-security/application-control/windows-defender-application-control/wdac>.

—. „Application User Model IDs (AppUserModelIDs).“ 07. 01 2021. <https://learn.microsoft.com/en-us/windows/win32/shell/appids?redirectedfrom=MSDN>.

—. „AppLocker.“ 26. 07 2023. <https://learn.microsoft.com/en-us/windows/security/application-security/application-control/windows-defender-application-control/applocker/applocker-overview>.

—. „Authorize reputable apps with the Intelligent Security Graph (ISG).“ 26. 07 2023. <https://learn.microsoft.com/en-us/windows/security/application-security/application-control/windows-defender-application-control/design/use-wdac-with-intelligent-security-graph>.

—. „Automatically allow apps deployed by a managed installer with Windows Defender Application Control.“ 26. 07 2023. <https://learn.microsoft.com/en-us/windows/security/application-security/application-control/windows-defender-application-control/design/configure-authorized-apps-deployed-with-a-managed-installer>.

—. „Catalog Files and Digital Signatures.“ 05. 03 2022. <https://learn.microsoft.com/en-us/windows-hardware/drivers/install/catalog-files>.

—. „Consumer antivirus software providers for Windows.“ 01. 05 2023. <https://support.microsoft.com/en-US/windows-antivirus-software-providers>.

—. „CreateTimerQueueTimer function (threadpoollegacyapiset.h).“ 13. 10 2021. <https://learn.microsoft.com/en-us/windows/win32/api/threadpoollegacyapiset/nf-threadpoollegacyapiset-createtimerqueuetimer>.

—. „cscript.“ 03. 02 2023. <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/cscript>.

—. „Diagnostics, feedback, and privacy in Windows.“ 03. 09 2021. <https://support.microsoft.com/en-us/windows/diagnostics-feedback-and-privacy-in-windows-28808a2b-a31b-dd73-dcd3-4559a5199319>.

—. „Digital Signatures.“ 07. 01 2021. <https://learn.microsoft.com/en-us/windows/win32/seccrypto/digital-signatures>.

—. „Digital Signatures for Kernel Modules on Systems Running Windows Vista.“ June 2007. <https://learn.microsoft.com/en-us/previous-versions/dotnet/articles/bb530195(v=msdn.10)>.

—. „Driver Signing.“ 08. 05 2023. <https://learn.microsoft.com/en-us/windows-hardware/drivers/install/driver-signing>.

—. „File Streams (Local File Systems).“ 07. 01 2021. <https://learn.microsoft.com/en-us/windows/win32/fileio/file-streams>.

—. „Impersonation.“ 21. 08 2020. <https://learn.microsoft.com/en-us/windows/win32/com/impersonation>.

—. „InstallELAMCertificateInfo function (sysinfoapi.h).“ 13. 10 2021. <https://learn.microsoft.com/en-us/windows/win32/api/sysinfoapi/nf-sysinfoapi-installelamcertificateinfo>.

—. „Isolated User Mode (IUM) Processes.“ 07. 01 2021. <https://learn.microsoft.com/en-us/windows/win32/procthread/isolated-user-mode--ium--processes>.

—. „Kernel Extended Attributes.“ 09. 08 2023. <https://learn.microsoft.com/en-us/windows-hardware/drivers/ifs/kernel-extended-attributes>.

—. „LoadLibraryExA function (libloaderapi.h).“ 09. 02 2023. <https://learn.microsoft.com/en-us/windows/win32/api/libloaderapi/nf-libloaderapi-loadlibraryexa>.

—. „Memory integrity and VBS enablement.“ 07. 04 2023. <https://learn.microsoft.com/en-us/windows-hardware/design/device-experiences/oem-hvci-enablement>.

—. „Microsoft Defender for Endpoint.“ 14. 06 2023. <https://learn.microsoft.com/en-us/microsoft-365/security/defender-endpoint/microsoft-defender-endpoint?view=o365-worldwide>.

—. „Microsoft public symbol server.“ 23. 12 2023. <https://learn.microsoft.com/en-us/windows-hardware/drivers/debugger/microsoft-public-symbols>.

—. „Microsoft Security Advisory 932596.“ *Update to Improve Kernel Patch Protection*. 14. 08 2007. <https://learn.microsoft.com/en-us/security-updates/securityadvisories/2007/932596>.

—. „Naming Files, Paths, and Namespaces.“ 15. 12 2022. <https://learn.microsoft.com/en-us/windows/win32/fileio/naming-a-file#nt-namespaces>.

—. „Privileges.“ 07. 01 2021. <https://learn.microsoft.com/en-us/windows/win32/secauthz/privileges>.

—. „Protect security settings with tamper protection.“ 26. 06 2023. <https://learn.microsoft.com/en-us/microsoft-365/security/defender-endpoint/prevent-changes-to-security-settings-with-tamper-protection?view=o365-worldwide>.

—. „Protecting anti-malware services.“ 08. 08 2022. <https://learn.microsoft.com/en-us/windows/win32/services/protecting-anti-malware-services->.

—. „Remote procedure call (RPC).“ 08. 02 2022. <https://learn.microsoft.com/en-us/windows/win32/rpc/rpc-start-page>.

—. „Set-ExecutionPolicy.“ 13. 12 2022. <https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.security/set-executionpolicy?view=powershell-7.3>.

—. „Setting Up Kernel-Mode Debugging.“ 15. 12 2021. <https://learn.microsoft.com/en-us/windows-hardware/drivers/debugger/setting-up-kernel-mode-debugging-in-windbg--cdb--or-ntsd>.

—. „Test Signing.“ 08. 05 2023. <https://learn.microsoft.com/en-us/windows-hardware/drivers/install/test-signing>.

—. „Test your app's signature with Smart App Control.“ 15. 12 2022. <https://learn.microsoft.com/en-us/windows/apps/develop/smart-app-control/test-your-app-with-smart-app-control>.

—. „Understand Windows Defender Application Control (WDAC) policy rules and file rules.“ 26. 07 2023. <https://learn.microsoft.com/en-us/windows/security/application-security/application-control/windows-defender-application-control/design/select-types-of-rules-to-create>.

—. „Understanding Application Control event tags.“ 26. 07 2023. <https://learn.microsoft.com/en-us/windows/security/application-security/application-control/windows-defender-application-control/operations/event-tag-explanations#validatedsigninglevel>.

—. „Using Authz API.“ 07. 01 2021. <https://learn.microsoft.com/en-us/windows/win32/secauthz/using-authz-api>.

—. „Virtualization-based Security (VBS).“ 20. 03 2023. <https://learn.microsoft.com/en-us/windows-hardware/design/device-experiences/oem-vbs>.

—. „What is Smart App Control?“ 24. 03 2023. <https://support.microsoft.com/en-us/topic/what-is-smart-app-control-285ea03d-fa88-4d56-882e-6698afdb7003>.

—. „Windows 11 Security - Smart App Control, enhanced phishing protection and memory integrity features.“ 01. 11 2022. <https://www.youtube.com/watch?v=auBmX4X7PCA>.

—. „Windows Authenticode Portable Executable Signature Format.“ 21. 03 2008. <https://download.microsoft.com/download/9/c/5/9c5b2167-8017-4bae-9fde-d599bac8184a/Authenticode_PE.docx>.

—. „Windows Defender Application Control and AppLocker Overview.“ 26. 07 2023. <https://learn.microsoft.com/en-us/windows/security/application-security/application-control/windows-defender-application-control/wdac-and-applocker-overview>.

—. „Windows Internet.“ 19. 08 2020. <https://learn.microsoft.com/en-us/windows/win32/wininet/portal>.

—. „Windows Pro in S mode.“ 27. 04 2023. <https://learn.microsoft.com/en-us/windows/deployment/s-mode>.

—. „Windows Push Notification Services (WNS) overview.“ 17. 03 2023. <https://learn.microsoft.com/en-us/windows/apps/design/shell/tiles-and-notifications/windows-push-notification-services--wns--overview>.

—. „WinHTTP Sessions Overview.“ 01. 07 2021. <https://learn.microsoft.com/en-us/windows/win32/winhttp/winhttp-sessions-overview>.

—. „WinHttpSetStatusCallback function.“ 30. 06 2022. <https://learn.microsoft.com/en-us/windows/win32/api/winhttp/nf-winhttp-winhttpsetstatuscallback>.

„MSASCui.exe und MSASCuiL.exe in der Windows 10 1809 nicht mehr vorhanden – Erklärung.“ *deskmodder.de*. 12. 08 2018. <https://www.deskmodder.de/blog/2018/08/12/msascui-exe-und-msascuil-exe-in-der-windows-10-1809-nicht-mehr-vorhanden-erklaerung/>.

n4r1b. „Smart App Control Internals (Part 1).“ *Deep dive into the internals of the latest Windows Security feature: "Smart App Control"*. 29. 08 2022. <https://n4r1b.com/posts/2022/08/smart-app-control-internals-part-1/>.

—. „Smart App Control Internals (Part 2).“ 08. 09 2022. <https://n4r1b.com/posts/2022/09/smart-app-control-internals-part-2/>.

Shafir, Yarden. „HyperGuard – Secure Kernel Patch Guard: Part 1 – SKPG Initialization.“ Winsider Seminars & Solutions Inc. , 01. 01 2022. <https://windows-internals.com/hyperguard-secure-kernel-patch-guard-part-1-skpg-initialization/>.

Yosifovich, Pavel and Russinovich, Mark E. and Solomon, David A. and Ionescu, Alex. *Windows Internals, Part 1: System Architecture, Processes, Threads, Memory Management, and More (7th Edition)*. USA: Microsoft Press, 2017.

Glossary

ALPC
Advanced Local Procedure Call, 15
API
Application Programming Interface, 9, 15, 16, 18, 21, 26, 29, 36, 51, 56, 58, 60
cryptcatsvc
Cryptographic Catalog Service, 8, 22, 24, 31, 51
DLL
Dynamic Link Library, 12, 18, 31
EA
Extended Attributes, 16, 23, 27, 29, 52
FAT
File Allocation Table, 16
GUID
Globally Unique Identifier, 15, 24, 30, 32, 37, 42, 54
HTTP
Hypertext Transfer Protocol, 18, 34, 36, 38, 39, 40, 41, 42
HTTPS
Hypertext Transfer Protocol Secure, 8, 9, 12, 25, 36, 37, 38, 42, 57
HVCI
Hypervisor-Protected Code Integrity, 14, 20, 23
ISG
Intelligent Security Graph, 19, 20, 59
KPP
Kernel Patch Protection, 14, 19
MAPS
Microsoft Active Protection Service, 16, 38, 39
MSRC
Microsoft Security Response Center, 19
NTFS
New Technology File System, 16, 58
RPC
Remote Procedure Call, 4, 8, 14, 15, 21, 22, 24, 25, 26, 29, 30, 31, 32, 33, 46, 51, 56, 60
SAC
Smart App Control, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 33, 34, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 53, 54, 56, 57, 64, 65
TLS
Transport Layer Security, 34, 35, 36
UMCI
User Mode Code Integrity, 23
VBS
Virtualization-based Security, 14, 20, 23, 59, 60, 65, 66
WDAC
Windows Defender Application Control, 3, 7, 8, 12, 13, 14, 15, 19, 20, 21, 22, 23, 24, 25, 27, 29, 44, 45, 47, 48, 51, 53, 60, 65
WinHTTP
Microsoft Windows HTTP Services, 3, 9, 18, 36, 58, 61
WinINet
Windows Internet, 18

Annexes

Annex A

Description of the experiment:

It is desirable to understand which kind of file are analyzed by SAC. To proceed, it matters to know which kind of file are specifically oriented to be analyzed by Microsoft Defender Antivirus during a memory scan, the internal name of the procedure involved.

Why performing an experiment? Even if a large part of the kernel and Microsoft Defender Antivirus has been reverse engineered for this study, it matters to check what is really happening during execution. On the first hand, it confirms by the experiment what has been observed during the analysis of SAC. On the other hand, it helps to illustrates the diversity of possible situations where SAC is or could have been involved, especially in the context of the eligibility of an image to be notified to SAC.

To proceed, we propose to make the difference between the files originally rooted to the code integrity analysis and the one finally notified to Microsoft Defender Antivirus when dealing with SAC. From the kernel point of view, it matters to check which kind of files are provided to the `CiValidateImageHeader` function (the entry point of the code integrity analysis) and the `CiCatDbSmartlockerDefenderCheck` function (the notification entry point from the kernel to Microsoft Defender Antivirus). In another way, it would also be interesting to check when it may be relevant the `ServerMpRpcMemoryScanStart` function relative to the SAC analysis in Microsoft Defender Antivirus.

In Windows, many elements are susceptible to be executed. Nevertheless, the system can only **natively** execute the executable files following the MZ-PE format. It means that regular application (.exe) and (.dll) are considered, in addition to drivers (.sys). We also consider the notion of script files, specifically PowerShell ones. For SAC feature, the digital signature of a file matters. By consequence, we evaluate the different files executed in conjunction with the different levels of digital signature they may have.

Conditions of the experiment:

- The evaluation is performed for different kinds of files (application, library, driver, and script).
- The evaluation is performed for different levels of digital signatures.
- The executable file is launched on a Windows 11 machine in debug mode with an attached debugger.
- There are two breakpoints set in the system (the `CiValidateImageHeader` function and the `CiCatDbSmartlockerDefenderCheck` function) to check which files are forwarded to SAC when there is an execution.
- No specific modification has been performed on the Windows 11 virtual machine.

Experimental protocol:

The experiment has been conducted with the Windbg debugger attached in kernel-mode to a virtual machine running Windows 11. The experimental protocol aims to launch an application, a script, an installer, or a driver in the virtual machine, looking for potential breakpoints hit during the experimental session. The breakpoints have been set before the experimental session and couple of checks have been performed in a context where SAC is notified to check there are perfectly functionals.

Results and detailed analysis:

The results are provided in section 3.3.1 in Table 1and in Table 2. Many observations can be withdrawn from the experiment.

The first observation is that only applications are notified to SAC. By applications, we mean the regular executable files whose extension is ".exe" but also the installers. In the last case, there are two kinds of installers. The first kind is the usual "setup.exe" which is just a regular application in the end. The other kind concerns Microsoft Software Installer (".msi" files) whose file format is undocumented by Microsoft. But in both situations, the executable image is mapped into memory as any other user-mode application.

It is worth noting here regular applications digitally signed by Microsoft are not notified to SAC. No matters the digital signature of the application is legitimately trusted by Windows as a third party-party legitimate certificate, this one is notified to SAC. Also, in the context of installer, even when they are signed by Microsoft, these ones are notified to SAC.

The second main observation concerns the scripts. In our experiment, we tested two types of scripts, namely PowerShell and Visual Basic Script (VBS). The first kind of script is a little complex to consider since PowerShellscript execution is naturally restricted by default on Windows out-of-the-box (Microsoft). Different tests have been performed to ensure the highest possible degree of reliability in the experiments we carry out. On the first hand, the "AllSigned" PowerShell execution policy makes sense in the context of digitally signed scripts, this configuration does not include unsigned script. On the other hand, the "ByPass" or the "Unrestricted" PowerShell execution policy allows any kind of script to be executed but it might prevent WDAC to perform any check on scripts, meaning that SAC (which relies on WDAC) could be impacted. To solve this issue, we decided to test in both situations. Whatever is the PowerShell execution policy set, the result is the same. No script is notified to SAC and even more unnotified to the `CiValidateImageHeader` function. In Microsoft Defender Antivirus, checking for the `ServerMpRpcMemoryScanStart` function in MpSvc.dll, there is no further notification.

The same observation applies for VBS scripts. If VBS does not have an execution policy embedded in Windows as specific as PowerShell, any VBS script can be directly executed with the "cscript" interpreter (Microsoft). In each context evaluated, there is no notification observed to SAC in Microsoft Defender Antivirus, including to the `ServerMpRpcMemoryScanStart` function in MpSvc.dll.

The last observation refers to drivers. None of them are notified to SAC even if they are directly taken into consideration by the Code Integrity check (through the `CiValidateImageHeader` function). The first analysis related to the Code Integrity check makes sense since drivers are expected to be signed since Windows Vista on 64-bit version (Microsoft). The lack of SAC notification could be explained by two arguments that are not mutually exclusive. On the first hand, since the signature of the driver is trusted, there is no necessity to go further in the analysis procedure. Also, there is a technical issue analyzing boot drivers since at that time Microsoft Defender Antivirus and Internet are not yet operational in the system. On the other hand, it may be the experiment environment that polluted the measurement, especially when the Windows 11 is run in a virtual machine set to debug mode (Microsoft). Therefore, test signing mode (Microsoft) is automatically enabled, especially for drivers. In such a case, any driver signed or unsigned will be authorized to be loaded. This consequence may have impacted the measured results, especially for drivers' notification to SAC since – in the end – the kernel *knows* it is going to load the driver, no matter would be the analysis provided by SAC.

In the end, the SAC feature is first a security mechanism dedicated to user-mode applications, including MSI installers which are especially taken into consideration. We observe that Microsoft's applications are exempted to by notified to SAC, based on their signature. This exception may be explained because of performance optimization, avoiding analyzing well-known and Microsoft considered safe applications. We note an exception concerning .NET libraries. Even though there are part of Windows out of the box, the last ones are systematically notified to SAC. The signature is probably the most relevant explanation in this context.

Annex B

The following gives an overview of the different files tested in the context of the experiment provided in Annex A.

- Executable application files:
  - Unsigned: HelloWorld.exe – executable file written in C and compiled with Visual Studio, only supposing to display “Hello World” on screen.
  - Untrusted third-party Signature: HelloWorld.exe self-signed.
  - Trusted third-party Signature: processhacker.exe, signed with a Digicet EV certificate.
  - Microsoft Signed: calc.exe, notepad.exe, paint.exe.
- Script:
  - PowerShell:
    - Unsigned: HelloWorld.ps1 – a simple PowerShell script resulting in display “Hello World” on screen.
    - Untrusted third-party Signature: HelloWorld.ps1 self-signed.
    - Trusted third-party Signature: SpeculationControlCompliance.ps1 written by Merlin from Belgium (@merlin_with_a_j) and stored on the PowerShell Gallery.
    - Microsoft Signed: The Loaded of DdcHelper library, part of the Powershell Scripts provided with Microsoft Defender Antivirus.
  - VBS:
    - Unsigned: HelloWorld.vbs – a simple VBS script displaying display “Hello World” on screen if it is executed with cscript.
    - Untrusted third-party Signature: HelloWorld.vbs self-signed.
    - Trusted third-party Signature: guitools.vbs from the kilibri-installer-windows project (signed with Symantec certificate issued by VeriSign to “Foundation for Learning Equality, Inc.”).
    - Microsoft Signed: adoquery.vbs from Microsoft’s github depot.
- Driver:
  - Unsigned: HelloWorld.sys driver bases on WPP display.
  - Untrusted third-party Signature: HelloWorld.sys self-signed.
  - Trusted third-party Signature: kprocesshacker.sys, signed with a Digicet EV certificate.
  - Microsoft Signed: CmBatt.sys, the default driver provided by Microsoft for ACPI Control Method Battery Driver (not run by default, only on demand).
- Installer:
  - Unsigned: installer from Perl strawberry project (.msi file).
  - Untrusted third-party Signature: Perl strawberry project self-signed.
  - Trusted third-party Signature: OpenVPN 2.5.7.
  - Microsoft Signed: Windows Driver Kit setup.

Annex C

- ".appref-ms"
- ".appx"
- ".appxbundle"
- ".bat"
- ".chm"
- ".cmd"
- ".com"
- ".cpl"
- ".dll"
- ".drv"
- ".gadget"
- ".hta"
- ".iso"
- ".js"
- ".jse"
- ".lnk"
- ".msc"
- ".msp"
- ".ocx"
- ".pif"
- ".ppkg"
- ".printerexport"
- ".ps1"
- ".rdp"
- ".reg"
- ".scf"
- ".scr"
- ".settingcontent-ms"
- ".sys"
- ".url"
- ".vb"
- ".vbe"
- ".vbs"
- ".vhd"
- ".vhdx"
- ".vxd"
- ".wcx"
- ".website"
- ".wsf"
- ".wsh"