Sandboxing Antimalware Products for Fun and Profit - Elastic Security Research

Gabriel Landau · @gabriellandau 2022-02-02 ⋮

Windows Internals Protected Processes Security Vulnerability

This article demonstrates a flaw that allows attackers to bypass a Windows security mechanism which protects anti-malware products from various forms of attack. This is of particular interest because we build and maintain two anti-malware products that benefit from this protection.

Protected Anti-Malware Services¶

Windows 8.1 introduced a concept of Protected Antimalware Services. This enables specially-signed programs to run such that they are immune from tampering and termination, even by administrative users. Microsoft's documentation (archived) describes this as:

> In Windows 8.1, a new concept of protected service has been introduced to allow anti-malware user-mode services to be launched as a protected service. After the service is launched as protected, Windows uses code integrity to only allow trusted code to load into the protected service. Windows also protects these processes from code injection and other attacks from admin processes.

The goal is to prevent malware from instantly disabling your antivirus and then running amok. For the rest of this article, we call them Protected Process Light (PPL). For more depth, Alex Ionescu goes into great detail on protected processes in his talk at NoSuchCon 2014.

To be able to run as a PPL, an anti-malware vendor must apply to Microsoft, prove their identity, sign binding legal documents, implement an Early Launch Anti-Malware (ELAM) driver, run it through a test suite, and submit it to Microsoft for a special Authenticode signature. It is not a trivial process. Once this process is complete, the vendor can use this ELAM driver to have Windows protect their anti-malware service by running it as a PPL.

You can see PPL in action yourself by running the following from an elevated administrative command prompt on a default Windows 10 install:

Protected Process Light in Action

```
C:\WINDOWS\system32>whoami
nt authority\system

C:\WINDOWS\system32>whoami /priv | findstr "Debug"
SeDebugPrivilege                        Debug programs                        Enabled
```

```
C:\WINDOWS\system32>taskkill /f /im MsMpEng.exe
ERROR: The process "MsMpEng.exe" with PID 2236 could not be terminated.
Reason: Access is denied.
```

As you can see here, even a user running as SYSTEM (or an elevated administrator) with `SeDebugPrivilege` cannot terminate the PPL Windows Defender anti-malware Service (`MsMpEng.exe`). This is because non-PPL processes like `taskkill.exe` cannot obtain handles with the `PROCESS_TERMINATE` access right to PPL processes using APIs such as OpenProcess.

In summary, Windows attempts to protect PPL processes from non-PPL processes, even those with administrative rights. This is both documented and implemented. That being said, with `PROCESS_TERMINATE` blocked, let's see if there are other ways we can interfere with it instead.

Windows Tokens¶

A Windows token can be thought of as a security credential. It says who you are and what you're allowed to do. Typically when a user runs a process, that process runs with their token and can do anything the user can do. Some of the most important data within a token include:

- User identity
- Group membership (e.g. Administrators)
- Privileges (e.g. SeDebugPrivilege)
- Integrity level

Tokens are a critical part of Windows authorization. Any time a Windows thread accesses a securable object, the OS performs a security check. It compares the thread's effective token against the security descriptor of the object being accessed. You can read more about tokens in the Microsoft access token documentation and the Elastic blog post that introduces Windows tokens.

Sandboxing Tokens¶

Some applications, such as web browsers, have been repeated targets of exploitation. Once an attacker successfully exploits a browser process, the exploit payload can perform any action that the browser process can perform. This is because it shares the browser's token.

To mitigate the damage from such attacks, web browsers have moved much of their code into lower-privilege worker processes. This is typically done by creating a restricted security context called a sandbox. When a sandboxed worker needs to perform a privileged action on the system, such as saving a downloaded file, it can ask a non-sandboxed "broker" process to perform the action on its behalf. If the sandboxed process is exploited, the goal is to limit the payload's ability to cause harm to only resources accessible by the sandbox.

While modern sandboxing involves several components of OS security, one of the most important is a low-privilege, or restricted, token. New sandbox tokens can be created with APIs such as `CreateRestrictedToken`. Sometimes a sandboxed process needs to lock itself down after performing some initialization. The `AdjustTokenPrivileges` and `AdjustTokenGroups` APIs allow this adjustment. These APIs enable privileges and groups to be “forfeit” from an existing process’s token in such a way that they cannot be restored without creating a new token outside the sandbox.

One commonly used sandbox today is part of Google Chrome. Even some security products are getting into sandboxing these days.

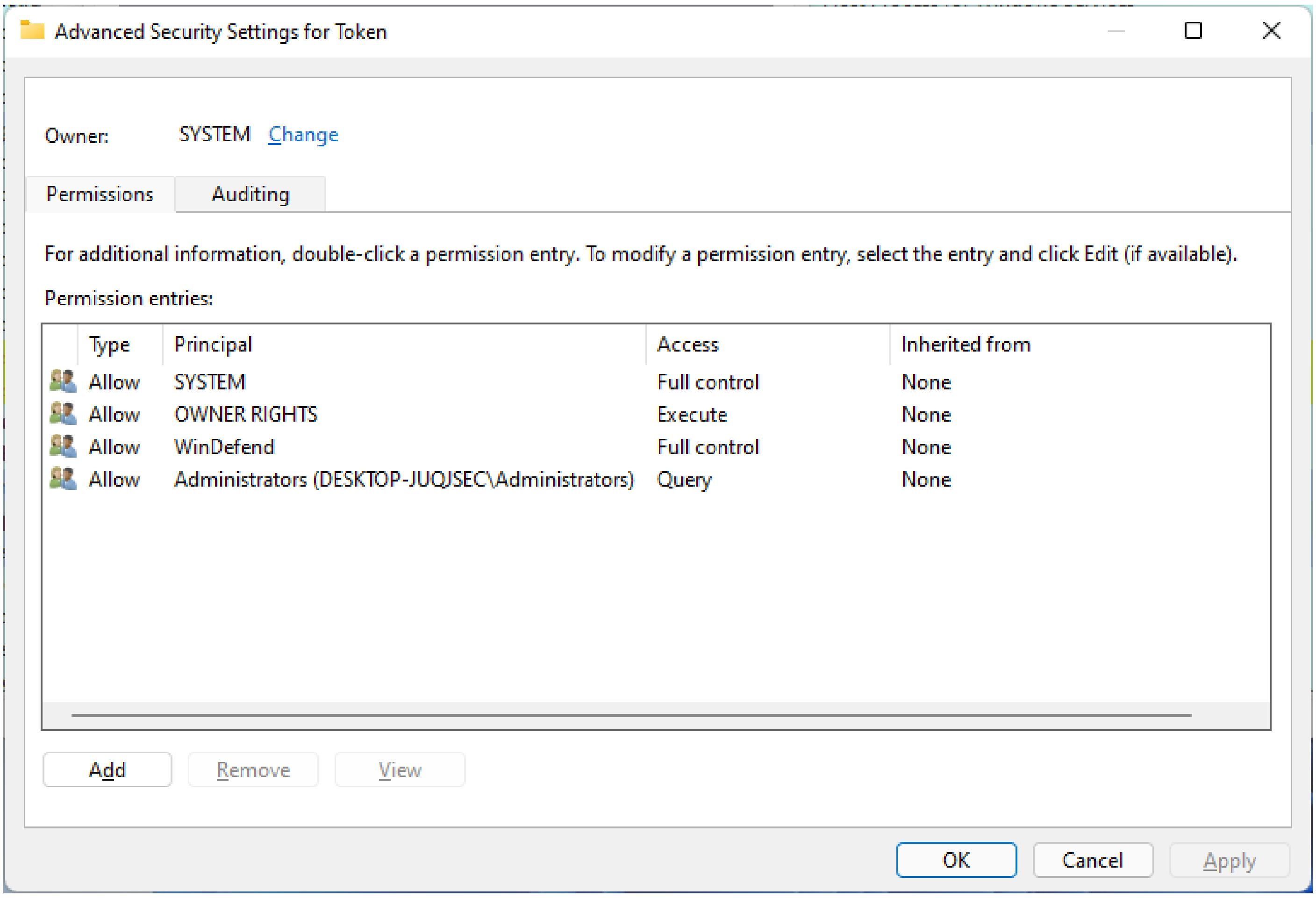
Accessing Tokens¶

Windows provides the `OpenProcessToken` API to enable interaction with process tokens. MSDN states that one must have the `PROCESS_QUERY_INFORMATION` right to use `OpenProcessToken`. Since a non-protected process can only get `PROCESS_QUERY_LIMITED_INFORMATION` access to a PPL process (note the `LIMITED`), it is seemingly impossible to get a handle to a PPL process’s token. However, MSDN is incorrect in this case. With only `PROCESS_QUERY_LIMITED_INFORMATION`, we can successfully open the token of a protected process. James Forshaw explains this documentation discrepancy in more depth, showing the underlying de-compiled kernel code.

Tokens are themselves securable objects. As such, regular access checks still apply. The effective token of the thread attempting to access the token is checked against the security descriptor of the token being accessed for the requested access rights (`TOKEN_QUERY`, `TOKEN_WRITE`, `TOKEN_IMPERSONATE`, etc). For more detail about access checks, see the Microsoft article, “How Access Checks Work.”

The Attack¶

Process Hacker provides a nice visualization of token security descriptors. Taking a look at Windows Defender’s (`MsMpEng.exe`) token, we see the following Discretionary Access Control List (DACL):


Advanced Security Settings for Token

Owner: SYSTEM Change

Permissions | Auditing

For additional information, double-click a permission entry. To modify a permission entry, select the entry and click Edit (if available).

Permission entries:

Type	Principal	Access	Inherited from
Allow	SYSTEM	Full control	None
Allow	OWNER RIGHTS	Execute	None
Allow	WinDefend	Full control	None
Allow	Administrators (DESKTOP-JUQJSEC\Administrators)	Query	None

Note that the SYSTEM user has full control over the token. This means, unless some other mechanism is protecting the token, a thread running as SYSTEM can modify the token. When such modification is possible, it violates the desired "PPL is protected from administrators" design goal.

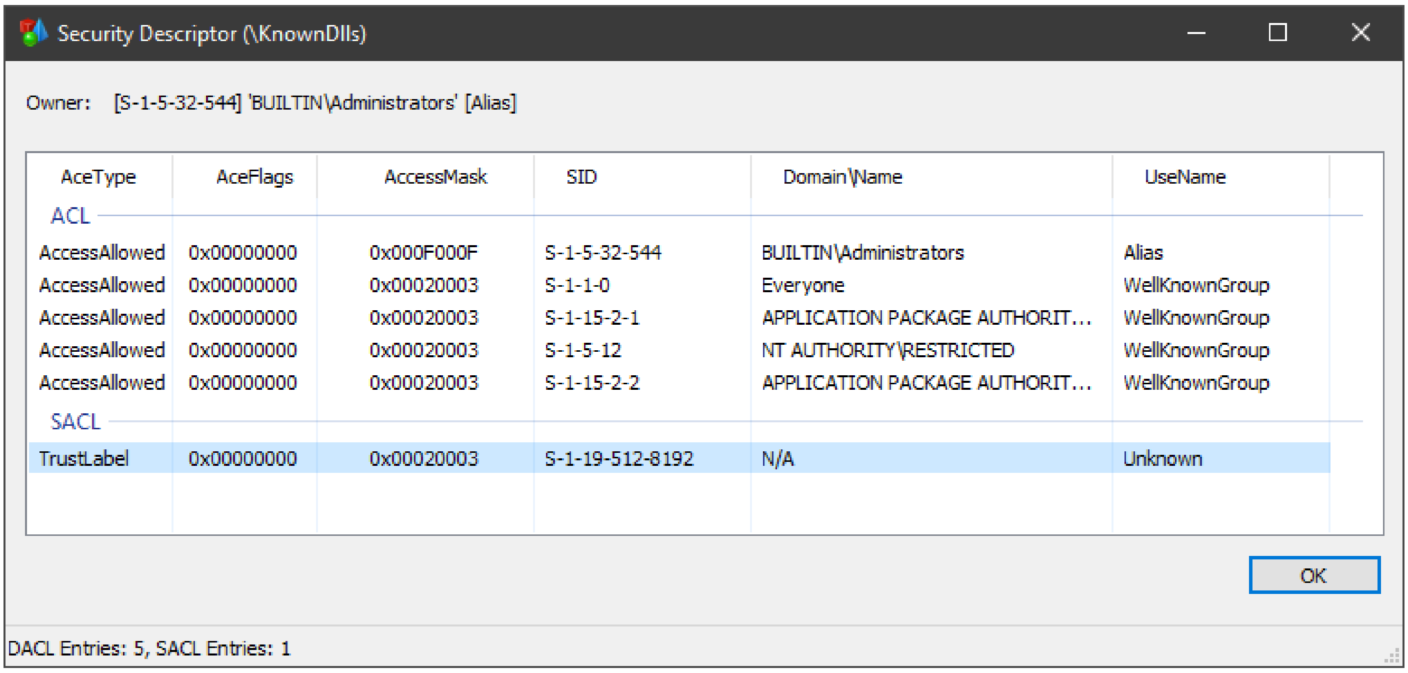
Demo¶

Alas, there is no other mechanism protecting the token. Using this technique, an attacker can forcefully remove all privileges from the `MsMpEng.exe` token and reduce it from system to untrusted integrity. Being nerfed to untrusted integrity prevents the victim process from accessing most securable resources on the system, quietly incapacitating the process without terminating it.

In this video, the attacker could have further restricted the token, but the privilege and integrity changes were sufficient to prevent `MsMpEng.exe` from detecting and blocking a Mimikatz execution. We felt this illustrated a valid proof of concept.

Defense¶

Newer versions of Windows include an undocumented feature called "trust labels." Trust labels are part of the System Access Control List (SACL), an optional component of every security descriptor. Trust labels allow Windows to restrict specific access rights to certain types of protected processes. For example, Windows protects the `\KnownDlls` object directory from modification by malicious administrators using a trust label. We can see this with WinObjEx64:


Security Descriptor (\KnownDlls)

Owner: [S-1-5-32-544] 'BUILTIN\Administrators' [Alias]

AceType	AceFlags	AccessMask	SID	Domain\Name	UseName
ACL					
AccessAllowed	0x00000000	0x000F000F	S-1-5-32-544	BUILTIN\Administrators	Alias
AccessAllowed	0x00000000	0x00020003	S-1-1-0	Everyone	WellKnownGroup
AccessAllowed	0x00000000	0x00020003	S-1-15-2-1	APPLICATION PACKAGE AUTHORIT...	WellKnownGroup
AccessAllowed	0x00000000	0x00020003	S-1-5-12	NT AUTHORITY\RESTRICTED	WellKnownGroup
AccessAllowed	0x00000000	0x00020003	S-1-15-2-2	APPLICATION PACKAGE AUTHORIT...	WellKnownGroup
SACL					
TrustLabel	0x00000000	0x00020003	S-1-19-512-8192	N/A	Unknown

DACL Entries: 5, SACL Entries: 1

Like `\KnownDlls`, tokens are securable objects, and thus it is possible to protect them against modification by malicious administrators. Elastic Security does this, in fact, and is immune to this attack, by denying `TOKEN_WRITE` access to processes with a trust label below “Anti-Malware Light.” Because this protection is applied at runtime, however, there is still a brief window of vulnerability until it can apply the trust label.

Ideally, Windows would apply such a trust label to each PPL process’s token as it is created. This would eliminate the race condition and fix the vulnerability in the PPL mechanism. There is precedent. With a kernel debugger, we can see that Windows is already protecting the System process’ token on Windows (21H1 shown below) with a trust label:

Trust Label on System Process Token

```
1: kd> dx -r1 (((nt!_OBJECT_HEADER*)((@$cursession.Processes[0x4]-
>KernelObject->Token->Object - sizeof(nt!_OBJECT_HEADER))  & ~0xf))-
>SecurityDescriptor & ~0xf)
(((nt!_OBJECT_HEADER*)((@$cursession.Processes[0x4]->KernelObject->Token-
>Object - sizeof(nt!_OBJECT_HEADER))  & ~0xf))->SecurityDescriptor & ~0xf)
: 0xffffe00649c46c20
1: kd> !sd 0xffffe00649c46c20
->Revision: 0x1
->Sbz1    : 0x0
->Control : 0x8814
            SE_DACL_PRESENT
            SE_SACL_PRESENT
            SE_SACL_AUTO_INHERITED
            SE_SELF_RELATIVE
->Owner   : S-1-5-32-544
->Group   : S-1-5-32-544
->Dacl    :
```

```
->Dacl     : ->AclRevision: 0x2
->Dacl     : ->Sbz1       : 0x0
->Dacl     : ->AclSize    : 0x1c
->Dacl     : ->AceCount   : 0x1
->Dacl     : ->Sbz2       : 0x0
->Dacl     : ->Ace[0]: ->AceType: ACCESS_ALLOWED_ACE_TYPE
->Dacl     : ->Ace[0]: ->AceFlags: 0x0
->Dacl     : ->Ace[0]: ->AceSize: 0x14
->Dacl     : ->Ace[0]: ->Mask : 0x000f01ff
->Dacl     : ->Ace[0]: ->SID: S-1-5-18

->Sacl     :
->Sacl     : ->AclRevision: 0x2
->Sacl     : ->Sbz1       : 0x0
->Sacl     : ->AclSize    : 0x34
->Sacl     : ->AceCount   : 0x2
->Sacl     : ->Sbz2       : 0x0
->Sacl     : ->Ace[0]: ->AceType: SYSTEM_MANDATORY_LABEL_ACE_TYPE
->Sacl     : ->Ace[0]: ->AceFlags: 0x0
->Sacl     : ->Ace[0]: ->AceSize: 0x14
->Sacl     : ->Ace[0]: ->Mask : 0x00000001
->Sacl     : ->Ace[0]: ->SID: S-1-16-16384

->Sacl     : ->Ace[1]: ->AceType: SYSTEM_PROCESS_TRUST_LABEL_ACE_TYPE
->Sacl     : ->Ace[1]: ->AceFlags: 0x0
->Sacl     : ->Ace[1]: ->AceSize: 0x18
->Sacl     : ->Ace[1]: ->Mask : 0x00020018
->Sacl     : ->Ace[1]: ->SID: S-1-19-1024-8192
```

The `SYSTEM_PROCESS_TRUST_LABEL_ACE_TYPE` access control entry limits access to `READ_CONTROL`, `TOKEN_QUERY`, and `TOKEN_QUERY_SOURCE` (`0x00020018`) unless the caller is a `WinTcb` protected process (SID `S-1-19-1024-8192`). That SID can be interpreted as follows:

- 1: Revision 1
- 19: `SECURITY_PROCESS_TRUST_AUTHORITY`
- 1024: `SECURITY_PROCESS_PROTECTION_TYPE_FULL_RID`
- 8192: `SECURITY_PROCESS_PROTECTION_LEVEL_WINTCB_RID`

Mitigation¶

Alongside this article, we are releasing an update to the PPLGuard proof-of-concept that protects all running anti-malware PPL processes against this attack. It includes example code that anti-malware products can employ to protect themselves. Here it is in action, protecting Defender:

Disclosure¶

We disclosed this vulnerability and proposed fixes to the Microsoft Security Response Center (MSRC) on 2022-01-05. They responded on 2022-01-24 that they have classified it as moderate severity, and will not address it with a security update. However, they may address it in a future version of Windows.

Conclusion¶

In this article, we disclosed a flaw in the Windows Protected Process Light (PPL) mechanism. We then demonstrated how malware can use this flaw to neutralize PPL anti-malware products. Finally, we showed a simple ACL fix (with sample code) that anti-malware products can employ to defend against this attack. Elastic Security already incorporates this fix, but we hope that Windows implements it (or something equivalent) by default in the near future.

Last update: February 2, 2022

Created: February 2, 2022