

Sandboxing Antimalware Products for Fun and Profit

 elastic.github.io/security-research/whitepapers/2022/02/02.sandboxing-antimalware-products-for-fun-and-profit/article/

Elastic Security Research



Sandboxing Antimalware Products for Fun

Elastic Security identifies Windows security feature bypass

Windows Internals Vulnerability



Gabriel Landau · [@gabriellandau](https://twitter.com/gabriellandau) 2022-02-02

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 (Protected Process Light)). 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 (Protected Process Light), an anti-malware vendor must apply to Microsoft, prove their identity, sign binding legal documents, implement an [Early Launch Anti-Malware \(ELAM \(Early Launch Anti-Malware\)\)](#) 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 \(Early Launch Anti-Malware\) driver](#) to have Windows protect their anti-malware service by running it as a PPL (Protected Process Light).

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

Protected Process Light in Action

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

In summary, Windows attempts to protect PPL (Protected Process Light) processes from non-PPL (Protected Process Light) 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.

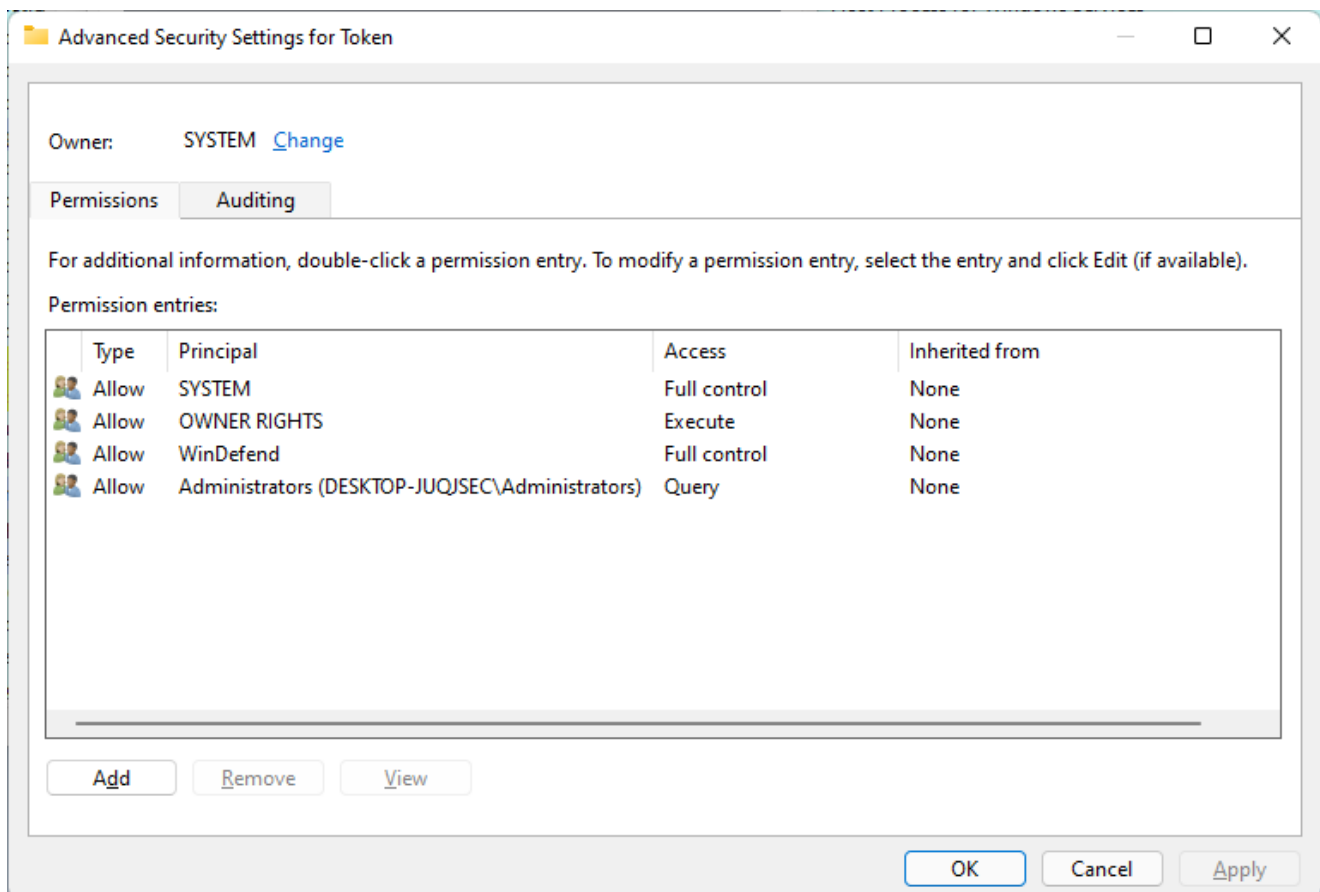
Accessing Tokens¶

Windows provides the `OpenProcessToken` API to enable interaction with process tokens. MSDN (Microsoft Developer Network) 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 (Protected Process Light) 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 (Discretionary Access Control List)):



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 (Protected Process Light) is protected from administrators” design goal.

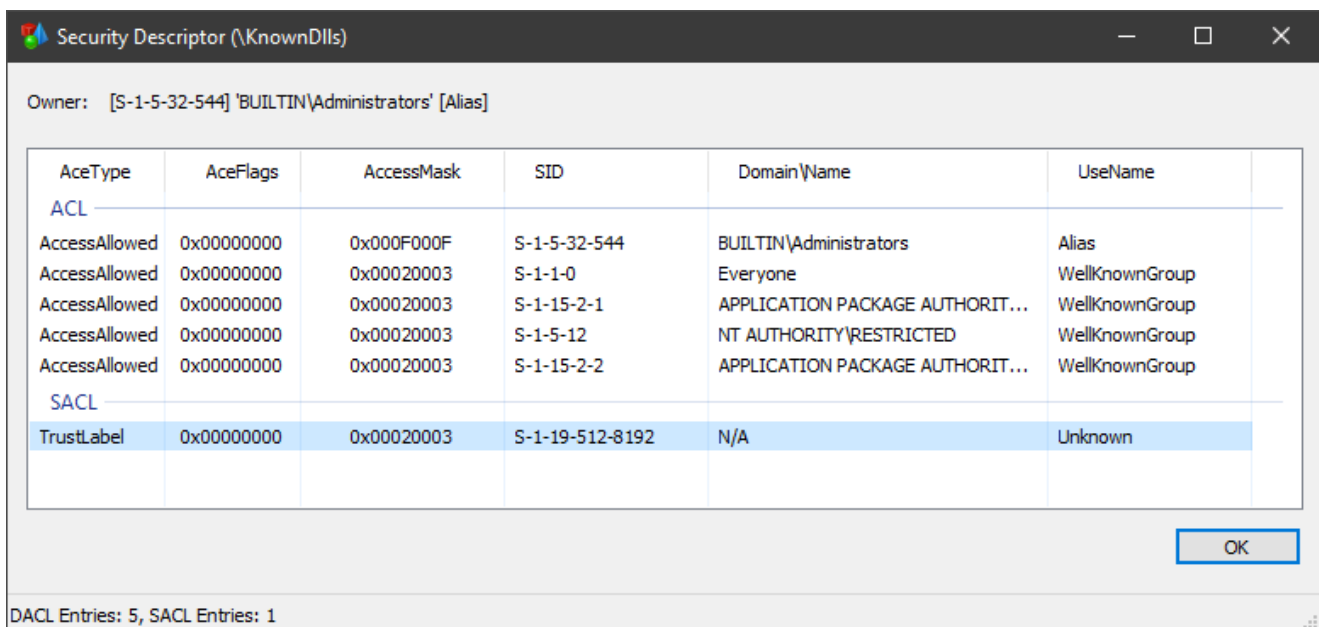
Demo¶

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`:



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 (Protected Process Light) process’s token as it is created. This would eliminate the race condition and fix the vulnerability in the PPL (Protected Process Light) 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:

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 (Protected Process Light) 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 \(Microsoft Security Response Center\)\)](#) 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 (Protected Process Light)) mechanism. We then demonstrated how malware can use this flaw to neutralize PPL (Protected Process Light) 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.