nccgroup
freedom from doubt

An NCC Group Publication

Bypassing Windows AppLocker using a Time of Check Time of Use vulnerability

Prepared by:
Ollie Whitehouse

Contents

1 Introduction and executive summary

1.1 Research findings overview

Windows AppLocker [1] [2] is Microsoft's replacement to Software Restriction Policies [3] [4] [5] in Windows 7, Windows 8, Server 2008 and Server 2012. Windows AppLocker has been promoted by several government agencies such as the National Security Agency [6] [7] and the New Zealand National Cyber Security Center [8] as an effective mechanism to combat the execution of unauthorized code on modern Microsoft Windows based systems. AppLocker has previously been shown to suffer other means of subversion [21] [22], although no out the box universal technique is currently publically known.

This paper presents the findings from research conducted by NCC Group into a way to bypass Windows AppLocker to allow unauthorized code to execute on a system. For the attack outlined in this paper to be effective the following dependencies exist:

- Windows AppLocker is **not** configured to only use Path based restrictions on the local machine, i.e. only allowing files stored on local hard drive to execute. Instead AppLocker should be configured to use some combination of Path, Signature and Hash based restrictions.
- An attacker can execute a file from a network share on a host which they control or can perform a Man-in-the-Middle attack on.

If these conditions can be met then an attacker can bypass Windows AppLocker to execute code of their choice.

1.2 Microsoft's response

"*We appreciate the security researchers at NCC Group for coordinating their disclosure with us. We have investigated this potential scenario and determined this is not a security vulnerability because it requires a third-party to persuade the customer to run a malicious program; a situation that does not cross a security boundary. AppLocker is designed to help organizations with compliance requirements, reduce administrative overhead and help control which applications are in use in their environments. AppLocker can also help prevent the execution of some malware in the enterprise. We encourage customers to configure their AppLocker policy to avoid running executables from remote network shares. For more information about AppLocker, please see the AppLocker Overview.*"

2 An introduction to Time of Check Time of Use race conditions

A Time of Check Time of Use (TOCTOU) race condition [9] [10] describes a class of vulnerability where between the process of 'checking' and 'using' data, that data can change. This change in data, if controllable by an attacker has a number of potential impacts on the security of a system. This class of vulnerability is catalogued by Mitre in the Common Weakness Enumeration under CWE-367 [24].

This class of vulnerability is surprisingly relevant to modern day systems security. A recent example of published work in this area is the paper titled '*Read It Twice! A mass-storage-based TOCTOU attack*' by Mulliner and Michele from 2012 [11].

3 Bypassing Windows AppLocker with TOCTOU

3.1 History behind the attack hypothesis

TOCTOU vulnerabilities have been known about as a class of issue for a significant amount of time. The particulars behind targeting Windows AppLocker build upon previous work by the author exploiting a specific TOCTOU vulnerability in Windows UAC.

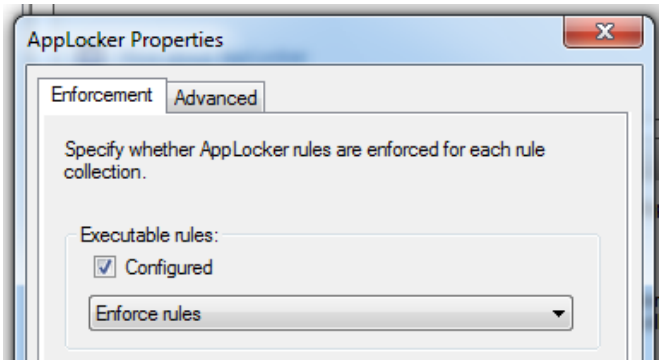
In this previous attack the concept was simple. The secure desktop and the user's execution of a program involved two different file handles. Due to this fact the file could be replaced between the dialogue showing the vendors details being displayed and the process being created in the user's context.

The attack outlined in this paper builds upon this work and applies it to (in the author's opinion) other security critical functionality within the Microsoft Windows operating system. The key difference being the same handle to the file is maintained but due to multiple reads a TOCTOU vulnerability exists that can be exploited when accessing files over the network.

3.2 Research environment configuration

Before describing the attack in detail it's useful to understand the environment configuration that NCC assessed. For the purposes of this research project NCC only focused on executable rule enforcement and not Windows Installer, Script and DLL enforcement as this configuration was thought sufficient for the purposes of the research project.

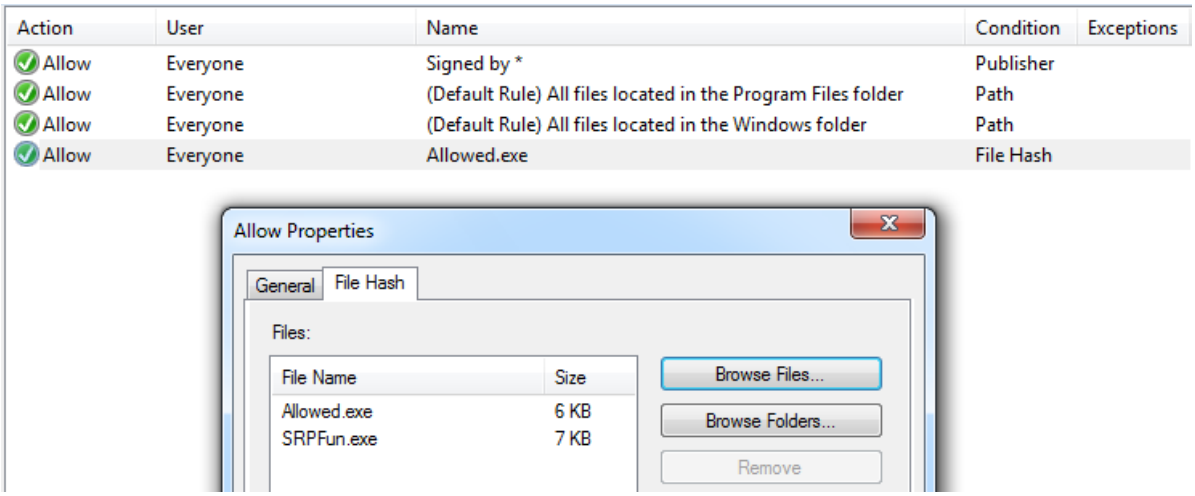
NCC configured a Windows 7 32bit Enterprise edition virtual machine (fully patched as of November 13, 2012) with Executable rule enforcement:



NCC then configured a number of rules:

- Everyone can execute anything signed by anyone (Publisher)
- Everyone can execute anything in the Program Files directory (Path)
- Everyone can execute anything in the Windows folder (Path)
- Everyone can execute two files based on their hash (File Hash)
    - `Allowed.exe`
    - `SRPFun.exe`

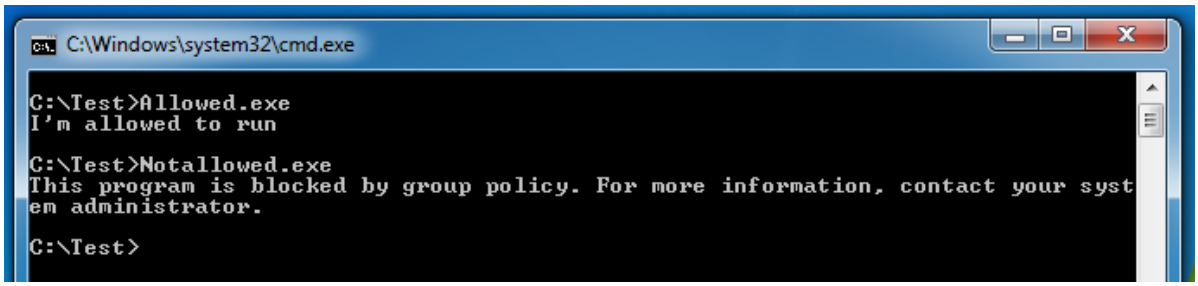
This configuration is show below:



This configuration was then validated by placing two unsigned files in `C:\Test` and executing them. These two files were:

- `Allowed.exe` – allowed to be executed due to its file hash.
- `Notallowed.exe` – not allowed to execute due to its file hash not being included in the rule.

These programs were then executed as a non-administrative user to validate the configuration as shown in the following screenshot.

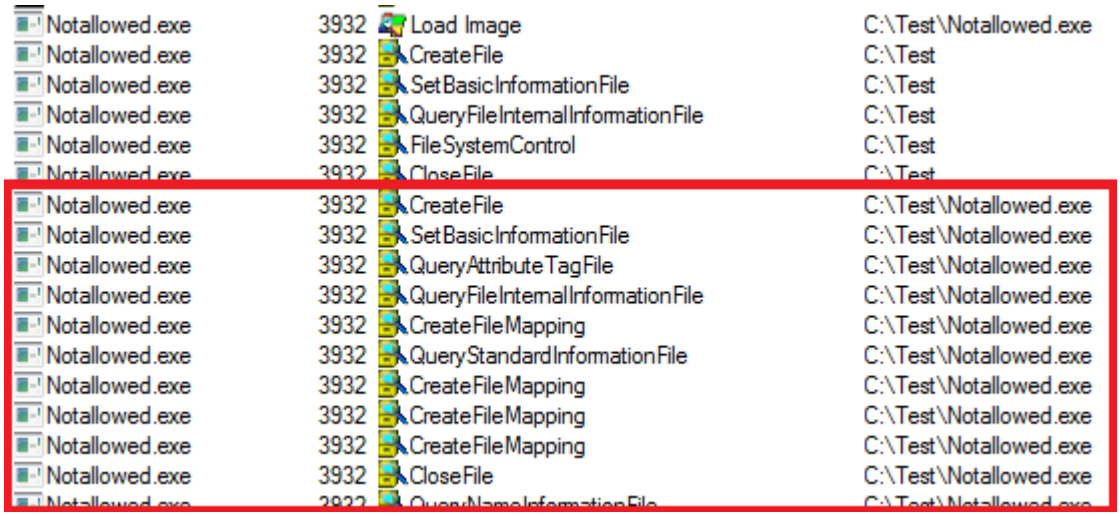


This configuration formed the basis of NCC's research environment.

3.3 Observing program loading behavior

Windows AppLocker is implemented in the `CreateProcess` [12] function call using code located in a variety of DLLs including `KERNEL32.DLL` and `ADVAPI32.DLL`.

If we observe the execution of `Notallowed.exe`, which is blocked, with Microsoft SysInternals Process Monitor [13], we see the following activity:



We can see in the boxed area in the above image that first a file handle is obtained via `CreateFile` [14] and the `CreateFileMapping` [15] is used. The use of `CreateFileMapping` during the loader process is interesting as this function call is typically coupled with `MapViewOfFile` [16] to create memory-mapped files. Microsoft even touches on this fact when discussing a previous non-security related bug in the Software Restriction Policy implementation [17].

"*I reviewed the source code and found that this is the code that performs Software Restriction Policy checking. Specifically, we are attempting to map the executable into memory to perform hash checking on it. Since there isn't 1.8 GB of contiguous available memory, it failed.*"

To validate the fact that multiple reads occur of the source binaries NCC placed the `Allowed.exe` and `Notallowed.exe` executables on a remote SMB file share and observed the loading process with WireShark.

During the loading process of both executables three reads using a variety of different offsets and sizes were observed. These are showing in the following table.

Note: Both files were 6,656 bytes in size

Read	Offset	Size
1	0	4096
1	6114	512
1	3584	1536
1	5632	512
2	0	6656
3	0	4096
3	4096	2560

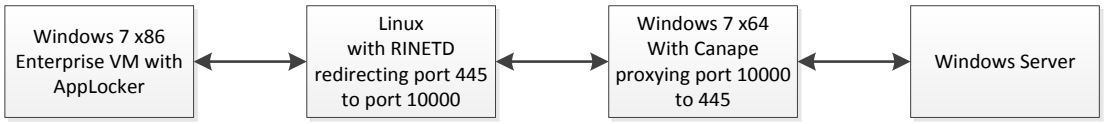
In the case where a binary was allowed to execute there was a final read, which was not evident where execution was blocked:

Read	Offset	Size
4	5120	512

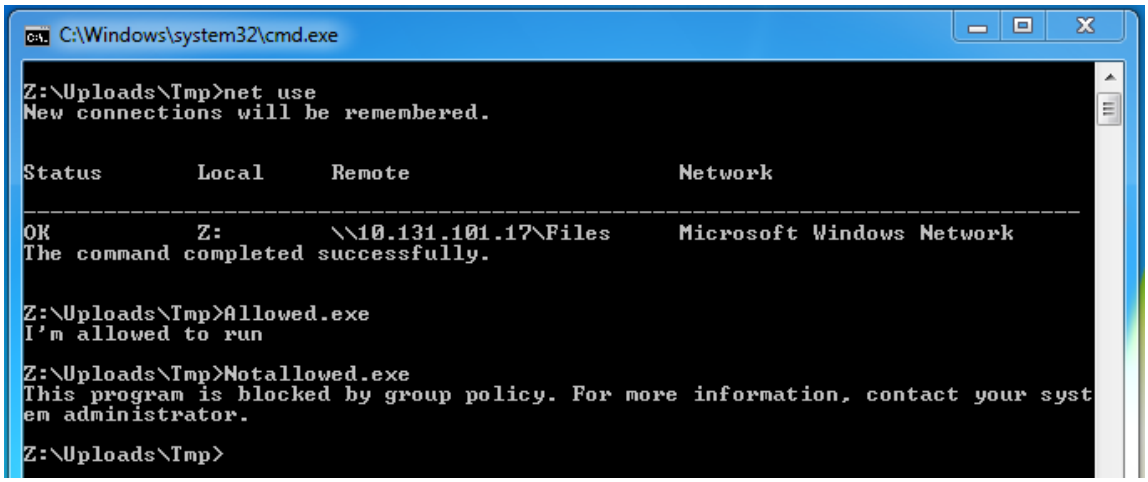
This behavior indicated to NCC that there was a high probability of a TOCTOU vulnerability due to multiple reads in Windows AppLocker allowing the execution of modified unauthorized code. However in order to exploit this vulnerability the handle returned by the original `CreateFile` function call cannot be invalidated due to an underlying file change. In order to successfully demonstrate this vulnerability a Man in the Middle (MitM) style attack was decided as the most convenient method to validate the hypothesis.

3.4 Validating and exploiting

To validate if modification of a binary was possible, pass the hash validation and still have it run the following environment was configured:



The binaries `Allowed.exe` and `Notallowed.exe` were placed on the Windows Server host with Canape [18] configured as a manipulating proxy. The binaries were then run as a non-administrative user to validate the configuration as shown below:

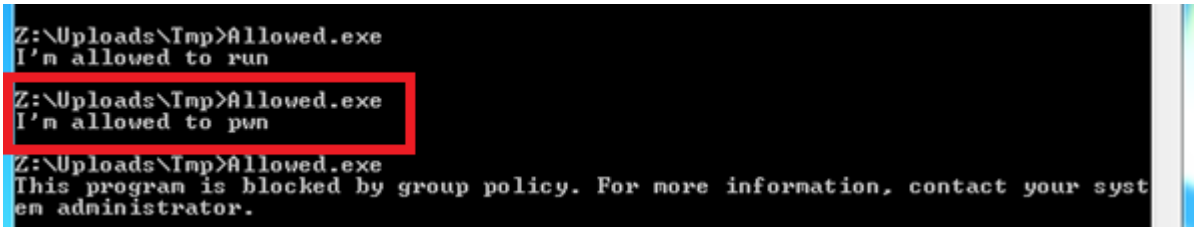


NCC then configured Canape to flag SMB packets for editing those that contained the text '*I'm allowed to run*' as this was contained in the `.text` segment of the binary (i.e. where the binary code exists). As expected this data filtering rule was triggered a total of three times during the execution.

In each case the text was modified from '*I'm allowed to run*' to '*I'm allowed to pwn*' to test the ability to modify the `.text` segment. Each of these SMB packets was modified in turn to observe the behavior of the loading process.

Trigger Packet	Result of Modification
1	No change in program output – program executes
2	Change in program output – program executes < vulnerable
3	Hash validation failure – program fails to run

The output for each of these modifications can be seen below, the second showing the vulnerability:



This change in behavior demonstrates that NCC was able to successfully modify a program, have it execute without triggering a hash validation failure and thus Windows AppLocker's susceptibility to the TOCTOU vulnerability class was proven. These results also showed that it is the third read of the binary in the case of hash based rules which is used validate the binary. NCC also confirmed AppLocker's susceptibility to the same vulnerability when using digitally signed binaries and rules based on publisher. However in this case the first read is used to validate the binary.

3.5 USB attack scenario applicability

During the course of this research NCC attempted to replicate this attack locally over USB using a Facedancer [19] [20] board configured to emulate a mass storage device.

This attack scenario was unsuccessful with the following the behavior observed:

- An initial single read is performed on the binary on first execution.
- Each subsequent execution occurs from a local cache resulting in no further USB access.

4 Mitigation advice when using AppLocker

For organizations that deploy Windows AppLocker as part of their defense in depth strategy NCC recommends the following mitigations to minimize the likelihood of exploitation of this or any other similar issues in the future:

- Deploy SMBv2 packet signing to mitigate Man in the Middle style attacks.
- Restrict paths that binaries can be executed from to local non-removable media sources where possible.

5 Mitigation advice to software architects and developers

For software architects and developers considering this type of threat as part of their Secure Development Lifecycle the following mitigation advice is provided:

- Where an attacker controls the source of data or can intercept and modify traffic then only data that is validated should be used i.e. multiple reads should be avoided. These sources can include network based sources as well as physical medium sources.
- Where a single read cannot be used then source data could be copied or cached to a local source where multiple reads can occur without risk of modification.

6 Further areas of research

The base technique outlined this paper may potentially have applicability to similar solutions both on Microsoft Windows and other operating systems.

NCC Group did for example attempt to obtain to an evaluation copy of Bit9 [23] but were unable too prior to publication.

NCC sees that the following as possible areas for future research in order to understand the applicability of this base technique:

- Any operating system that loads large files over remote file a system that performs input validation to mitigate memory corruption and then subsequently reads data from these remote sources after successful validation.
- Bit9 and competing software restriction enforcement products on Microsoft Windows when loading binaries over SMB.
- Linux binary signing when loading binaries over SMB, NFS or similar remote file systems.

7 Conclusions

This paper has summarized NCC's research into and successfully demonstrated Windows AppLocker's susceptibility to a TOCTOU vulnerability. Of particular note is the risk of data modification where file handles are not invalidated because an attacker controls the source of data or has the ability to perform Man in the Middle style attacks. NCC believes that this style of logic vulnerability will become increasingly popular with researchers and attackers alike as existing classes of vulnerability become harder to exploit or less common.

8 References & further reading

The following material was used during the course of this research project.

1. Windows AppLocker – Technet
   http://technet.microsoft.com/en-us/library/dd759117.aspx
2. Windows AppLocker – Technet (Video)
   http://technet.microsoft.com/en-us/windows/applocker.aspx
3. Windows Software Restriction Policies
   http://technet.microsoft.com/en-us/library/dd349795(v=ws.10).aspx
4. Using Software Restriction Policies to Protect Against Unauthorized Software
   http://technet.microsoft.com/en-us/library/cc507878.aspx
5. How Software Restriction Policies Work
   http://technet.microsoft.com/en-us/library/cc786941(v=ws.10).aspx
6. National Security Agency – Application Whitelisting Using SRP
   http://www.nsa.gov/ia/_files/os/win2k/Application_Whitelisting_Using_SRP.pdf
7. National Security Agency - Mitigation Monday #3: Defense against Malware on Removable Media
   http://www.nsa.gov/ia/_files/factsheets/Mitigation_Monday_3.pdf
8. New Zealand Nation Cyber Security Centre – Application Whitelisting with Windows AppLocker
   http://www.ncsc.govt.nz/sites/default/files/articles/NCSC%20Applocker-public%20v1.0.5.pdf
9. CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition
   http://cwe.mitre.org/data/definitions/367.html
10. Time of Check Time of Use
    http://en.wikipedia.org/wiki/Time_of_check_to_time_of_use
11. Read It Twice! A mass-storage-based TOCTOU attack – Collin Mulliner and Benjamin Michele
    https://www.usenix.org/system/files/conference/woot12/woot12-final28.pdf

12. CreateProcess – MSDN
http://msdn.microsoft.com/en-gb/library/windows/desktop/ms682425(v=vs.85).aspx
13. Microsoft System Internals – Process Monitor
http://technet.microsoft.com/en-gb/sysinternals/bb896645.aspx
14. CreateFile – MSDN
http://msdn.microsoft.com/en-gb/library/windows/desktop/aa363858(v=vs.85).aspx
15. CreateFileMapping – MSDN
http://msdn.microsoft.com/en-gb/library/windows/desktop/aa366537(v=vs.85).aspx
16. MapViewOfFile – MSDN
http://msdn.microsoft.com/en-gb/library/windows/desktop/aa366761(v=vs.85).aspx
17. Case Study - Software Restriction Policies and Large EXE Files
http://blogs.msdn.com/b/ntdebugging/archive/2010/01/18/safer-and-large-exes.aspx
18. Canape by ContextIS
http://www.contextis.co.uk/research/tools/canape/
19. Facedancer 11
http://goodfet.sourceforge.net/hardware/facedancer11/

20. Emulating USB Devices with Python
http://travisgoodspeed.blogspot.co.uk/2012/07/emulating-usb-devices-with-python.html
21. You can circumvent AppLocker rules by using an Office macro on a computer that is running Windows 7 or Windows Server 2008 R2
http://support.microsoft.com/kb/2532445
22. Circumventing SRP and AppLocker, By Design – Didier Stevens
http://blog.didierstevens.com/2011/01/24/circumventing-srp-and-applocker-by-design/
23. Bit9 Endpoint and Server Security
https://www.bit9.com/
24. CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition
http://cwe.mitre.org/data/definitions/367.html

9 Acknowledgements

The author wishes to thank his colleagues Matt Lewis, David Wood, Richard Turnball and Sharique Shaikh of NCC Group for their peer review. The author also wishes to thank Andy Davis of NCC Group for this input on the USB attack scenario.

The author would like to thank James Forshaw for releasing Canape which significantly expedited the validation of the original hypothesis.

Finally we'd like to thank Jeremy Tinder and Katie Moussouris of Microsoft for facilitating the release of this white paper.

