AppUNBlocker: Bypassing AppLocker | Tripwire

By Editorial Staff

Published: 2016-10-27 · Archived: 2026-04-05 23:46:12 UTC

Windows AppLocker is a powerful whitelisting technology built into modern Windows operating systems. It provides the ability to lock down installers, scripts and executables on the local machine via either a white list or a black list of file data. For many organizations, this is a great technology to reduce the attack surface of the endpoint by limiting what can run to a specific set of known files. To make this decision on what is able to execute, AppLocker can inspect the file's path, hash, or publisher information. Each of these has their pros and cons, which I will discuss here. Stating which paths can execute on a file system is the easiest step for an organization to lock down an endpoint. Using the golden image as a reference system, the expected Windows and installed applications can be profiled and imported into AppLocker relatively easily. This prevents mainly unwanted software from running, as it would be installed in an "unknown" location on the operating system. Additionally, this is the easiest to maintain, as expected locations for applications to execute from will maintain fairly static. However, for a malicious attacker, simply placing the malware in a "known" location can bypass the AppLocker's intended function. The next option is to import a list of known hashes into AppLocker and block anything else from running. This also has the benefit of preventing unwanted software from running on the endpoint, be it in a known or unknown location. While daunting at first, it's easy to use PowerShell scripts to scan a reference system, format an AppLocker XML policy and import directly into the endpoint's AppLocker configuration. For static environments, this is a great option, as causing a hash collision is incredibly difficult. Most IT endpoints are not static, causing this AppLocker configuration to be difficult to manage at scale. Each Patch Tuesday will require the new hashes to be imported into the AppLocker configuration on each endpoint. Without doing this first, unexpected and potentially catastrophic results could ensue from AppLocker blocking critical system processes. This leaves the final option of validating the publisher information of system files. Windows system files are signed by a valid Microsoft certificate, while most software vendors are switching to signing their software, as well. Any remaining files which aren't signed can be inspected manually via either the path or hash functionality. There are two methods to bypass publisher verification in AppLocker. Both methods do require inserting a malicious trusted certificate into the root certificate trust store. While this may sound difficult, the trust store is simply in a predictable point within the registry. On any Windows operating system, the root trust store is located in HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\SystemCertificates\ROOT\Certificates. There's a Key for each trusted certificate on the machine. Injecting a malicious certificate here allows the system to trust a file an attacker signed with the appropriate certificate. In order to bypass AppLocker, the attacker can also create an additional AppLocker rule in the AppLocker configuration. This can potentially be very noisy from a logging perspective. The better method would be to inspect the AppLocker configuration by pulling the active configuration from a PowerShell command, such as "Get-AppLockerPolicy –Effective –XML". If the system is trusting the signed Windows files, for example, this is easily bypassed. The first step is to create a malicious certificate authority and associated certificate to sign the piece of malware. In my example, I am using a Meterpreter Reverse TCP Shell intended to look like Windows Calculator. When creating the CA and certificate, use the following information:

- **Country**: US
- **State**: Washington
- **Locality Name**: Redmond
- **Organization Name**: Microsoft Corporation
- **Common Name**: Microsoft Windows Operating System

Next, sign the file with the created certificate. While there are many tools out there, using Microsoft's SignTool is a simple and easy way to accomplish this. Execute "Signtool.exe sign /f <certificate_file> /p <certificate_password> <malicious_file_name>" to sign the file with the new certificate. The next step, getting the new CA to be trusted by the victim, is more difficult. If you have physical control over the device, you can double-click the CA file to install or use Microsoft's CertMgr tool to inject the new CA. These are obviously not ideal in a real-world scenario, but they are available should you find yourself in that predicament. More likely, you'll want to inject the CA data blob into the registry directly. If you can gain Meterpreter beforehand, the inject_ca module can do this for you with a few modifications. Without Meterpreter, injecting directly into the registry works great, as the certificate looks the same on any machine, meaning you can inject the certificate into a machine you control and export the registry key from there to be used later on the victim. Once the malicious certificate authority has been injected into the victim, the signed piece of malware will bypass the publisher verification checks for Windows system files. AppLocker first checks that the executable is signed by a trusted certificate, which is why the malicious CA had to be injected. After this, AppLocker will do a string comparison on the publisher data. Since the certificate was created using Microsoft's information, the string characters match and the file is allowed to execute. Even though this bypass seems dangerous on the surface, it does require administrative privileges to inject the malicious certificate authority into the registry. Grant local administrative privileges to end users only if absolutely necessary. If there is a business justification for local administrative privileges, monitoring the registry entries at HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\SystemCertificates\ROOT\Certificates is absolutely necessary. The certificates stored here will rarely change, if ever, over the life of a typical endpoint. Included below is a list of expected Windows certificates, which ship with various Windows platforms. Many enterprises will have additional trusted certificates on their endpoints which can only be identified by the enterprise which minted the certificate; however following this list can reduce the overall workload of validating the various trusted certificates across the enterprise. The following keys are located in HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\SystemCertificates\ROOT\Certificates\
18F7C1FCC3090203FD5BAA2F861A754976C8DD25 245C97DF7514E7CF2DF8BE72AE957B9E04741E85
3B1EFD3A66EA28B16697394703A72CA340A05BD5 7F88CD7223F3C813818C994614A89C99FA3B5247
8F43288AD272F3103B6FB1428485EA3014C0BCFE A43489159A520F0D93D032CCAF37E7FE20A8B419
BE36A4562FB2EE05DBB3D32323ADF445084ED656 CDD4EEAE6000AC7F40C3802C171E30148030C072
Users of Tripwire Enterprise and CCM can download policies from the Tripwire Customer Center to both monitor the installed certificates, as well as verify they are part of the trusted certificates which Microsoft provides. Download the TE_AppUnBlocker.zip file located on the Threat Rules tab for Tripwire Enterprise content.

Source: https://www.tripwire.com/state-of-security/off-topic/appunblocker-bypassing-applocker/