GitHub - med0x2e/SigFlip: SigFlip is a tool for patching authenticode signed PE files (exe, dll, sys ..etc) without invalidating or breaking the existing signature.

By med0x2e

Archived: 2026-04-05 20:21:07 UTC

What is it ?

SigFlip is a tool for patching authenticode signed PE files (exe, dll, sys ..etc) in a way that doesn't affect or break the existing authenticode signature, in other words you can change PE file checksum/hash by embedding data (i.e shellcode) without breaking the file signature, integrity checks or PE file functionality.

SigInject encrypts and injects shellcode into a PE file's [WIN_CERTIFICATE] certificate table, the encryption key is printed out for usage with a basic BOF/C/C# loader (SigLoader), SigInject saves changes to a modified PE file and keeps its signature and certificate validity intact.

SigLoader is a basic loader which takes a modified PE file path created by SigInject and the decryption key as parameters, then extract and decrypt embedded shellcode for usage with a shellcode injection of choice.

SigFlip will check if PE hash was successfully changed and also check and exit gracefully in case endpoints are hardened against such common misconfiguration. (check "Details" section).

Quick Note: SigFlip, SigInject and SigLoader are available as BOF scripts and .NET assemblies, the only difference is that SigInject functionality is implemented as part of SigFlip (-i) in case if you choose to use .NET artifacts instead of BOFs.

Why ?

It can be used mainly for persistence, lateral movement or code/command execution and can help with:

- Application whitelisting bypasses, changing the PE file hash (msbuild.exe for ex) without breaking the signature.
- Bypassing EDRs relying on specific LOLBINs' hashes for malicious code/command execution detection.
- Load signed drivers using a different hash, might help circumvent any EDRs watching for common vulnerable signed drivers using a pre-defined list of hashes.
- Embed encrypted shellcode in a signed PE file and use a stager (sigloader) of your preference to parse, decrypt, load and execute it.
- Endpoint security vendors tend to classify signed PE files as benign most of the time, embedding your unsigned code (shellcode ..etc.) in a signed PE file makes it a little bit hard to detect/flag.
- Bypassing endpoint security vendors relying mainly on the default WinVerifyTrust for signature validation.

- Improving OPSEC and challenging defenders relying solely on typical signature verification utilities such as signtool, sigcheck, Get-AuthenticodeSignature ..etc to validate the authenticoode signature of PE files.

Usage & Examples:

Compile/Build:

Precompiled BOF's are not provided in this project, can be compiled using Mingw-w64, for **.NET** use VS or csc.exe to compile .NET projects (SigFlip, SigLoader), for **BOF** check steps below;

- `➜ i686-w64-mingw32-gcc -c sigflip.c -o sigflip.x86.o`
- `➜ x86_64-w64-mingw32-gcc -c sigflip.c -o sigflip.x64.o`
- `➜ x86_64-w64-mingw32-gcc -c SigLoader/sigloader.c -o sigloader.x64.o`
- `➜ i686-w64-mingw32-gcc -c SigLoader/sigloader.c -o sigloader.x86.o`

Make sure all object files are located in the same directory as sigflip.cna, then load sigflip.cna script to cobalt strike.

Quick Note: pre-compiled BOFs were tested and compatible with mingw-64 v8.0.0_3, **using mingw-64 >= v9 might work but might crash active beacons**, check #2 for more details.

Cobalt Strike:

1. **Execute-Assembly**
    - `execute-assembly SigFlip.exe -h`
    - `execute-assembly SigLoader -h`
2. **BOF**
    - For usage with cobalt strike, once you load the SigFlip.cna script, two new commands will be registered; SigFlip and SigInject, then use as below;
        - SigFlip: Change a PE file (DLL, EXE, SYS, OCX ..etc) hash without breaking the signature or the validity of the certificate:
            - `SigFlip "<PE\_FILE\_PATH>" "<OUTPUT\_PE\_FILE\_PATH (with extension)>"`
        - SigInject: Encrypts and Injects shellcode into a PE file's [WIN_CERTIFICATE] certificate table, encryption key is printed out for usage with a basic C/C# loader plus keeps the signature and certificate validity intact:
            - `SigInject "<PE\_FILE\_PATH> <OUTPUT\_PE\_FILE\_PATH (with extension)>" "<SHELLCODE\_FILE>"`
        - SigLoader: Loads encrypted shellcode from PE files created by SigInject, then use Early Bird queueuserapc to spawn/inject sc into a sacrificial process, shellcode injection logic can

be customized or replaced with any other code injection technique of choice:

- `SigLoader <PE_FILE_PATH_WITH_SH> <DECRYPTION_KEY> <SPAWNTO_PROCESS_PATH> <PARENT_PROCESS_ID>`

3. **Examples**

   - BOF:

     - Inject random data to msbuild.exe (aka bit flip msbuild.exe):
       - `SigFlip "C:\Windows\Microsoft.NET\Framework\v4.0.30319\msbuild.exe" "C:\lolbins\modified-msbuild.exe"`
     - Inject shellcode to kernel32.dll (Arguments order is different & make sure to take note of the decryption key):
       - `SigInject "C:\Windows\System32\kernel32.dll" "C:\random\modified-kernel32.dll" "C:\shellcode\cobaltstrike_or_msf_shellcode.bin"`
       - `Sigloader "C:\random\modified-kernel32.dll" "DECRYPTION_KEY" "C:\Windows\System32\werfault.exe" 6300`

   - Execute-Assembly:

     - Inject random data to msbuild.exe:
       - `execute-assembly SigFlip.exe -b C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe -o C:\Temp\MSBuild.exe`
     - Inject shellcode to kernel32.dll (Arguments order is different & make sure to take note of the decryption key):
       - `execute-assembly SigFlip.exe -i C:\Windows\System32\kernel32.dll -s C:\Temp\x86shellcode.bin -o C:\Temp\kernel32.dll -e TestSecretKey`
       - `execute-assembly SigLoader.exe -f C:\Temp\modified-kernel32.dll -e TestSecretKey -pid 2354`

Details:

This is a known technique used by APT#10 in multiple campaigns or intrusion sets.

Authenticode Digital Signatures ?

Authenticode is a Microsoft code-signing technology that identifies the publisher of Authenticode-signed software. Authenticode also verifies that the software has not been tampered with since it was signed and published.

How does it work ?

Microsoft relies mainly on Authenticode signing format for verifying the integrity and the origin of PE binaries, according to the Authenticode Portable Executable format specification the Authenticode signatures can be “embedded” in a Windows PE file, in a location specified by the Certificate Table entry in Optional Header Data Directories. When Authenticode is used to sign a Windows PE file, the algorithm that calculates the file’s Authenticode hash value excludes certain PE fields. When embedding the signature in the file, **the signing process can modify these fields without affecting the file’s hash value**. These fields are as follows: **the checksum, certificate table RVA, certificate table size and the **attribute certificate table**. The attribute certificate table contains a PKCS #7 SignedData structure containing the PE file’s **hash value**, a **signature** created by the software publisher’s private key, and the **X.509 v3 certificates** that bind the software publisher’s signing key to a legal entity.

In layman's terms, we can modify or embed data into fields execulded from the authenticode hash calculation without worrying about breaking the authenticode signature and file integrity checks.

More details about such excluded fields:

- Certificate table RVA and Size: A signed PE file optional header structure contains an array of data directories including the security directory **IMAGE_DIRECTORY_ENTRY_SECURITY** entry which has two fields, **RVA** and **Size**.
    - RVA: a file offset (not a memory offset) to the attribute certificate table.
    - Size: attribute certificate table size.
- Attribute Certificate Table: a data structure **WIN_CERTIFICATE** which encapsulate the signature and certificates and has the following fields:
    - `dwLength` : certificate table size.
    - `wRevision` : the “revision” of the `WIN_CERTIFICATE` .
    - `wCertificateType` : the kind of encapsulated certificate data.
    - `bCertificate` : the actual certificate data. For `WIN_CERT_TYPE_PKCS_SIGNED_DATA` , this is the PKCS#7 `SignedData` structure mentionned above (which contains the PE hash value, signature and x.509 certificate), this is exactly where SigFlip embed randm random data or shellcode.

With all of that in mind, now SifFlip does the following:

1. Check system configuration
2. Loading PE File & Verifying PE file signature & Compute Sha1 hash
3. Get "e_lfanew" offset (pointing to the PE FILE HEADER -> IMAGE_NT_HEADERS)
4. Get IMAGE_OPTIONAL_HEADER from IMAGE_NT_HEADERS
5. Get IMAGE_DATA_DIRECTORY from IMAGE_OPTIONAL_HEADER
6. Get IMAGE_DIRECTORY_ENTRY_SECURITY field and retrieve the RVA and SIZE of the Attribute Certificate Table (WIN_CERTIFICATE).
7. Patch the PE File blob by padding the Certificate Table with extra bytes (random/shellcode) of choice.
8. Update the optional header -> IMAGE_DIRECTORY_ENTRY_SECURITY data directory Size
9. Update WIN_CERTIFICATE (Certificate Table) dwLength

10. Generate the new PE cheksum and update it. (OPT Header Checksum)
11. Save the final PE with new size.
12. Verify modified PE file signature

The first step is essential to confirm if the system is misconfigured in a way to allow padding and injecting shellcode into authenticode signed PE files, therefore the following sanity checks are performed:

1. Check if MS13-098 fix is not installed (KB2893294), Keep in mind **IT COULD BE INSTALLED BUT REGISTRY KEYS ARE NOT PROPERLY SET, WHICH RENDERS THE PATCH USELESS**
2. Check Registry keys
    1. X86:
        - Check if registry key "HKLM:\Software\Microsoft\Cryptography\Wintrust\Config" is not available
            - -> if available then check if "EnableCertPaddingCheck" registry value is not available
    2. X64:
        - Check if registry key "HKLM:\Software\Wow6432Node\Microsoft\Cryptography\Wintrust\Config" is not available
            - -> if available then check if "EnableCertPaddingCheck" registry value is not available.

Why cannot read the injected data when the modified PE is loaded as a module into its own address space or the address space of other processes ?

Windows loader doesn't load certificate data into the process address space, reason why you need a custom loader to extract data such as shellcode and use it (ex: SigLoader). this should also explain why **IMAGE_DIRECTORY_ENTRY_SECURITY** data directory entry RVA is a file offset instead of a typical memory offset.

Detect/Prevent:

- https://docs.microsoft.com/en-us/security-updates/SecurityAdvisories/2014/2915720?redirectedfrom=MSDN
- Once the patch is installed and proper registry keys are set, No system restarts are required, you only need to restart the Cryptographic Services. The Applocker service will be also restarted as it depends on the cryptographic services.(@p0w3rsh3ll)
- Yara rule by Adrien; https://twitter.com/Int2e_/status/1330975808941330432

References

- https://docs.microsoft.com/en-us/security-updates/SecurityBulletins/2013/ms13-098?redirectedfrom=MSDN

- https://docs.microsoft.com/en-us/security-updates/SecurityAdvisories/2014/2915720?redirectedfrom=MSDN
- http://download.microsoft.com/download/9/c/5/9c5b2167-8017-4bae-9fde-d599bac8184a/authenticode_pe.docx
- https://msrc-blog.microsoft.com/2013/12/10/ms13-098-update-to-enhance-the-security-of-authenticode/
- https://www.specterops.io/assets/resources/SpecterOps_Subverting_Trust_in_Windows.pdf
- https://p0w3rsh3ll.wordpress.com/2014/05/24/testing-ms13-098-certificate-padding-check/
- http://jsac.jpcert.or.jp/archive/2021/pdf/JSAC2021_202_niwa-yanagishita_en.pdf

---