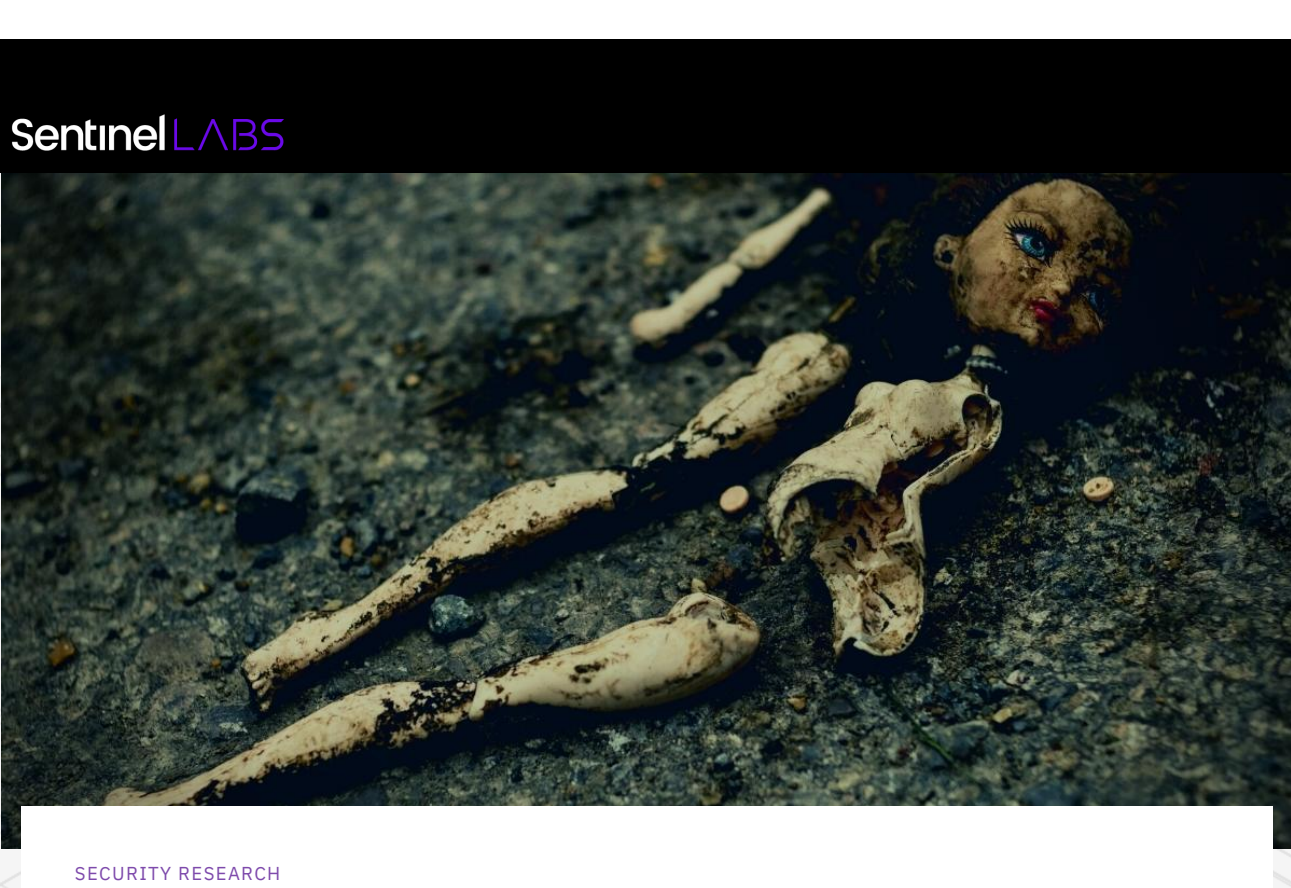
SentinelLABS



SECURITY RESEARCH

Inside Malicious Windows Apps for Malware Deployment

ALEKSANDAR MILENKOSKI / JULY 14, 2022

This article discusses Windows Apps – Windows applications packaged into *APPX* or *MSIX* packages – as a medium to deploy malware. Though not as widely abused as other infection vectors, there have been a number of recent high profile attacks that use Windows Apps.

- November, 2021: BazarBackdoor was distributed in the form of an *APPX* package.
- December, 2021: Emotet malware was distributed by abusing a spoofing vulnerability in the Windows App Installer, software that installs Windows Apps.
- January, 2022: Malicious Windows Apps in *APPX* format masquerading as critical browser updates were used to drop Magniber ransomware.
- February, 2022: Windows Apps laced with the Electron Bot malware were uploaded to the Microsoft Store, a trusted library of Microsoft-certified packaged Windows Apps.

Since Microsoft's announcement that Office applications will by default disable the execution of Office macros in the context of documents that originate from untrusted sources, there has been an uptick in malicious actors using alternative mediums for deploying malware such as Windows Apps and Windows shortcut *LNK* files. Despite Microsoft recently rolling back the decision to disable by default Office macro execution, these media complement malicious macros and remain a threat to watch for.

Previous research on malicious Windows Apps focused on concrete system infections involving particular instances of such Apps. This article complements previous research by taking a generic perspective. I take an *APPX* package that malicious actors have used to deploy malware as a running example and provide:

- A summarizing overview of the layout and content of *APPX* packages (*APPX* packages are structurally very similar to the alternative *MSIX* packages and the two Windows App packaging formats share the same deployment infrastructure).
- An overview of selected activities that the Windows system conducts when a user installs a malicious *APPX* package.

This enables a better understanding of what occurs on a system when a user falls prey to an attack that involves a malicious Windows App – from the operating system activities at first user interaction with the file to the malware deployment that the file triggers.

For Windows to install an *APPX* package, whether malicious or not, the system first has to establish trust in the package. Microsoft provides the Microsoft Store to Windows users as a library of certified packaged Windows Apps. These packages are digitally counter-signed by Microsoft. Windows trusts by default *APPX* packages that originate from the Windows Store. However, when the Windows App sideloading feature is enabled, users can install *APPX* packages that do not originate from the Microsoft Store as well, that is, packages that are not counter-signed by Microsoft. Such are the majority of the malicious *APPX* packages that the security community has observed in attacks.

In this article, we will dig into how malicious *APPX* packages can be installed on a Windows 10 system, with a focus on how Windows establishes trust in an *APPX* package. I focus on this aspect since trust is a crucial part of the Windows App installation process. This process is what ultimately deploys malware on a system. I will use the malicious *APPX* package `edge_update.appx` (SHA1: *e491af786b5ee3c57920b79460da351ccf8f6f6b*) as a running example.

What Does a Malicious Windows App Look Like?

The `edge_update.appx` *APPX* package is signed with a code signing certificate issued to Foresee Consulting Inc. and a root certificate issued by DigiCert Trusted Root G4. Windows systems trust DigiCert root certificates and they are placed in the *Trusted Root Certification Authorities* certificate store.

Figure 1 depicts the certificate chain of the digital signature of `edge_update.appx` before a Certificate Authority revoked the certificate.

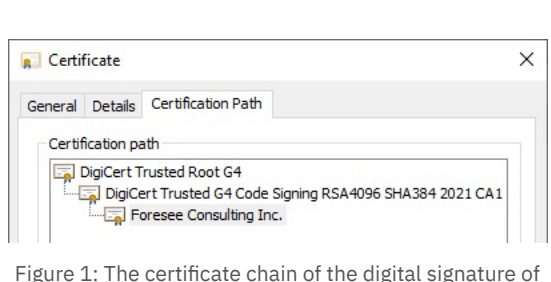


Figure 1: The certificate chain of the digital signature of *edge_update.appx* prior certificate revocation

APPX packages are ZIP archive files that store the executable files that implement the packaged Windows App and additional files. Figure 2 depicts the content of `edge_update.appx`. The `eediwjus` directory stores the malicious Windows App that the executable files `eediwjus.exe` and `eediwjus.dll` implement. *eediwjus.exe* is a .NET Windows App (see Figure 3) that invokes the `mhjpfzvitta` function from `eediwjus.dll`. This function executes malicious code that is heavily control-flow obfuscated with unconditional jumps (see Figure 4). The analysis of the malicious code is out of the scope of this article.

```
Directory of C:\Users\<user>\malicious_appx\

05/07/2022  12:47    <DIR>          .
05/07/2022  12:47    <DIR>          ..
29/12/2021  13:55             2.439 AppxBlockMap.xml
29/12/2021  13:55             2.039 AppxManifest.xml
05/07/2022  12:47    <DIR>          AppxMetadata
30/12/2021  00:18             8.342 AppxSignature.p7x
05/07/2022  12:47    <DIR>          eediwjus
05/07/2022  12:47    <DIR>          Images
29/12/2021  13:55             2.216 resources.pri
29/12/2021  13:55               740 [Content_Types].xml

[...]

Directory of C:\Users\<user>\malicious_appx\eediwjus

05/07/2022  12:47    <DIR>          .
05/07/2022  12:47    <DIR>          ..
29/12/2021  13:55             5.632 eediwjus.dll
29/12/2021  13:55             3.584 eediwjus.exe
```

Figure 2: The content of *edge_update.appx*

```
namespace eediwjus
{
    // Token: 0x02000002 RID: 2
    public class eediwjus
    {
        // Token: 0x06000001 RID: 1
        [DllImport("eediwjus.dll")]
        private static extern void mhjpfzvitta(uint lpBuffer);

        // Token: 0x06000002 RID: 2 RVA: 0x00002050 File Offset: 0x00000250
        private static void Main(string[] args)
        {
            uint lpBuffer = 5604U;
            eediwjus.mhjpfzvitta(lpBuffer);
        }
    }
}
```

Figure 3: The implementation of *eediwjus.exe*

```
.text:000000018000113F loc_18000113F:                          ; CODE XREF: mhjpfzvitta↓p
.text:000000018000113F                 push    rbp
.text:0000000180001140                 jmp     short loc_180001196
.text:0000000180001142 ; ---------------------------------------------------------------------------
.text:0000000180001142
.text:0000000180001142 loc_180001142:                          ; CODE XREF: .text:00000001800011B8↓j
.text:0000000180001142                 push    r14
.text:0000000180001144
.text:0000000180001144 loc_180001144:                          ; CODE XREF: .text:0000000180001179↓j
.text:0000000180001144                 jmp     loc_1800011F2
.text:0000000180001144 ; ---------------------------------------------------------------------------
.text:0000000180001149                 db 83h, 99h, 21h
.text:000000018000114C ; ---------------------------------------------------------------------------
.text:000000018000114C
.text:000000018000114C loc_18000114C:                          ; CODE XREF: .text:0000000180001212↓j
.text:000000018000114C                 lea     rdx, [rbp-8]
.text:0000000180001150                 jmp     loc_1800011E4
.text:0000000180001150 ; ---------------------------------------------------------------------------
.text:0000000180001155                 db 0F6h, 92h, 33h
.text:0000000180001158                 db 7Ch, 7Dh
.text:000000018000115A ; ---------------------------------------------------------------------------
.text:000000018000115A
.text:000000018000115A loc_18000115A:                          ; CODE XREF: .text:0000000180001228↓j
.text:000000018000115A                 mov     rbp, rsp
.text:000000018000115D                 jmp     short loc_1800011C9
.text:000000018000115D ; ---------------------------------------------------------------------------
```

Figure 4: Control-flow obfuscated malicious code in *eediwjus.dll*

In addition to the files that implement a Windows App, an *APPX* package stores additional files, such as `AppxManifest.xml`, `AppxBlockMap.xml`, and `AppxSignature.p7x`.

`AppxManifest.xml` is the package manifest, a file in XML (Extensible Markup Language) format that contains the information that Windows needs to deploy, display, and update a Windows App. This includes information about:

- The publisher of the App, where the publisher is the entity that digitally signs the *APPX* package and that is responsible for the development and release of the Windows App.
- Windows App properties, such as display name and logo.
- Software dependencies and capabilities: The Windows system controls what system resources a Windows App can access with respect to the capabilities that the publisher has assigned to the App. System resources include the Internet, filesystem locations, and networking. In summary, Windows Apps execute in a sandboxed, access-restricted, environment for security reasons.

Figure 5 depicts the content of `AppxManifest.xml` in the malicious `edge_update.appx`. The publisher of the Windows App is *Foresee Consulting Inc.*, the display name of the App is *Edge Update*, and the App has the capabilities `internetClient` and `runFullTrust`. The `internetClient` capability enables the malicious Windows App to download data from the Internet, probably a payload from an attacker-controlled endpoint.

```
<?xml version="1.0" encoding="utf-8"?>
<Package xmlns="http://schemas.microsoft.com/appx/manifest/foundation/windows10"
[...]>
  <Identity Name="3669e262-ec02-4e9d-bcb4-3d008b4afac9"
  Publisher="CN=Foresee Consulting Inc., O=Foresee Consulting Inc., L=North York, S=Ontario, C=CA,
  SERIALNUMBER=1004913-1, OID.1.3.6.1.4.1.311.60.2.1.3=CA, OID.2.5.4.15=Private Organization"
  Version="96.0.1072.0" ProcessorArchitecture="neutral" />
  <Properties>
    <DisplayName>Edge Update</DisplayName>
    <PublisherDisplayName>Microsoft Inc</PublisherDisplayName>
    <Logo>Images\StoreLogo.png</Logo>
  </Properties>
  <Dependencies>
    <TargetDeviceFamily Name="Windows.Universal" MinVersion="10.0.14393.0" MaxVersionTested="10.0.19041.0" />
    <TargetDeviceFamily Name="Windows.Desktop" MinVersion="10.0.14393.0" MaxVersionTested="10.0.19041.0" />
  </Dependencies>
  <Resources>
    <Resource Language="EN-US" />
  </Resources>
  <Applications>
    <Application Id="App" Executable="eediwjus\eediwjus.exe" EntryPoint="Windows.FullTrustApplication">
    [...]
        <uap:SplashScreen Image="Images\SplashScreen.png" />
        <uap:LockScreen BadgeLogo="Images\BadgeLogo.png" Notification="badgeAndTileText" />
      </uap:VisualElements>
    </Application>
  </Applications>
  <Capabilities>
    <Capability Name="internetClient" />
    <rescap:Capability Name="runFullTrust" />
  </Capabilities>
</Package>
```

Figure 5: The content of *AppxManifest.xml* in *edge_update.appx* (trimmed for brevity)

`AppxBlockMap.xml` is the package block map, a file in XML format that stores Base-64 encoded hash values of data blocks in the files that an *APPX* package archives. Windows uses these hashes to verify the data integrity of these files when installing an *APPX* package, a topic that I discuss further in this article.

Figure 6 depicts the content of `AppxBlockMap.xml` in the malicious `edge_update.appx`. The `HashMethod` XML attribute specifies the hash algorithm for calculating the data block hash values in `AppxBlockMap.xml`. The *File* XML element specifies a file in the *APPX* package and the size of the file. The *Block* XML element specifies the hash value and the size of a single data block in the file. For example, a data block in `eediwjus.exe` that is 1304 bytes big has the SHA-256 hash value of *ad4f74c0c3ac37e6f1cf600a96ae203c38341d263dbac0741e602686794c4f5a* (in hexadecimal format).

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<BlockMap
    xmlns="http://schemas.microsoft.com/appx/2010/blockmap"
    HashMethod="http://www.w3.org/2001/04/xmlenc#sha256">
    <File Name="Images\Square150x150Logo.scale-150.png" Size="29258" LfhSize="68">
        <Block Hash="aXbz/aM0VnEE3GcuC1Vt/Ku4UCD4p/9/BDEDOhksIUg="/>
    </File>
    [...]
    <File Name="eediwjus\eediwjus.exe" Size="3584" LfhSize="51">
        <Block Hash="rU90wMOsN+bxz2AKlq4gPDg0HSY9usB0HmAmhnlMT1o=" Size="1304"/>
    </File>
    <File Name="eediwjus\eediwjus.dll" Size="5632" LfhSize="51">
        <Block Hash="9CO9barmyAAqz1wgMmfgFfe+tMUu1Up4eJ3YarNeRsY=" Size="4280"/>
    </File>
    <File Name="resources.pri" Size="2216" LfhSize="43">
        <Block Hash="GkBWEftTjm5ET4h7RkUls8EJnS8kLkg5AJ7GqeVioGo=" Size="823"/>
    </File>
    <File Name="AppxManifest.xml" Size="2039" LfhSize="46">
        <Block Hash="WlMcXpSwuuwjmFw0XpesPEIJi7NxLrsAzFwTFulBHJQ=" Size="902"/>
    </File>
</BlockMap>
```

Figure 6: The content of *AppxBlockMap.xml* in *edge_update.appx* (trimmed for brevity)

`AppxSignature.p7x` is the *APPX* package signature, a PKCS (Public-Key Cryptography Standards) #7 digital signature data in ASN.1 (Abstract Syntax Notation One) format. `AppxSignature.p7x` stores signature data, such as the certificate chain of the digital signature and the actual signed data. The signed data includes hashes of files in the APPX package, such as `AppxManifest.xml` and `AppxBlockMap.xml`. Figure 7 depicts the formatted content of `AppxSignature.p7x` in the malicious `edge_update.appx`.

```
SEQUENCE (2 elem)
OBJECT IDENTIFIER 1.2.840.113549.1.7.2 signedData (PKCS #7)
[0] (1 elem)
  SEQUENCE (5 elem)
    INTEGER 1
    SET (1 elem)
      SEQUENCE (2 elem)
        OBJECT IDENTIFIER 2.16.840.1.101.3.4.2.1 sha-256 (NIST Algorithm)
        NULL
    SEQUENCE (2 elem)
      OBJECT IDENTIFIER 1.3.6.1.4.1.311.2.1.4 spcIndirectDataContext (Microsoft code signing)
      [0] (1 elem)
      [...]
           OBJECT IDENTIFIER 2.5.4.3 commonName (X.520 DN component)
           PrintableString DigiCert Trusted Root G4
      [...]
           OBJECT IDENTIFIER 2.5.4.3 commonName (X.520 DN component)
           PrintableString DigiCert Trusted G4 Code Signing RSA4096 SHA384 2021 CA1
      [...]
           OBJECT IDENTIFIER 2.5.4.3 commonName (X.520 DN component)
           PrintableString Foresee Consulting Inc.
      [...]
```

Figure 7: The content of *AppxSignature.p7x* in *edge_update.appx* (trimmed for brevity)

Is a Malicious Windows App Trustworthy?

The `AppxSvc` service (`Appx Deployment Service`), which the DLL (dynamic-link library) `%SystemRoot%\System32\appxdeploymentserver.dll` implements, orchestrates the installation of *APPX* packages. When a user installs an *APPX* package, the `AppxSvc` service verifies the data integrity of the package and verifies that the package satisfies certain trust criteria. Figure 8 depicts a simplified overview of the data integrity verification process that the `AppxSvc` service conducts.

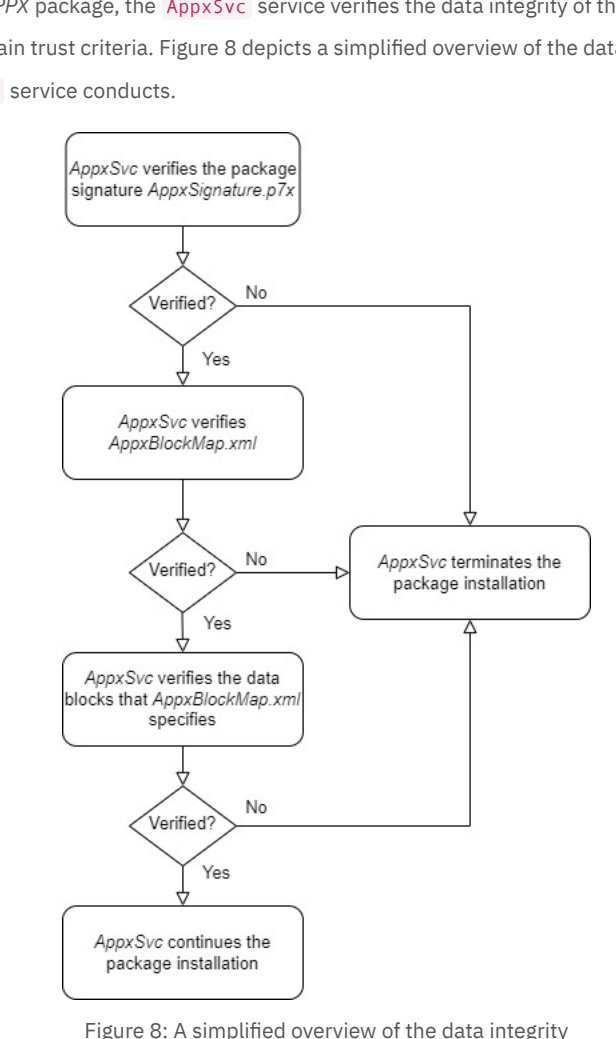


Figure 8: A simplified overview of the data integrity verification process that the *AppxSvc* service conducts

In summary, the data integrity verification process consists of the following steps:

1. The `AppxSvc` service invokes the *WinVerifyTrust* function to verify the *APPX* package signature `AppxSignature.p7x`, that is, to verify the signed data in `AppxSignature.p7x` using the certificate chain in `AppxSignature.p7x` (see Figure 7). The signed data includes hashes of files that the APPX package stores, including `AppxBlockMap.xml` (see Figure 6). For example, Figure 9 depicts the hash value of `AppxBlockMap.xml` in the malicious `edge_update.appx` and the same value in `AppxSignature.p7x`, the package signature of `edge_update.appx`.

```
PS C:\Users\<user>\Desktop\malicious_appx\malicious_appx> Get-FileHash .\AppxBlockMap.xml

Algorithm       Hash
---------       ----
SHA256          025C22C3EAD2720DA646FEA322EEBEC95B5B1CD1D0991850FFD28C7890F157C9

AppxSignature.p7x

Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000090  04 81 B8 41 50 50 58 41 58 50 43 52 64 C7 4C 18  ..¸APPXAXPCRdÇL.
000000A0  6E 14 91 78 57 38 9E 33 16 FF 99 6F 1F 61 79 12  n.'xW8ž3.ÿ™o.ay.
000000B0  B2 F9 AD 0B D6 86 C2 4D 74 82 2D 41 58 43 44 17  ²ù..Ö†ÂMt‚-AXCD.
000000C0  DC A1 20 4C 87 AC 12 D0 F4 C8 DD CA A1 B8 AC FD  Ü¡ L‡¬.ÐôÈÝÊ¡¸¬ý
000000D0  A5 D2 2D 29 24 DD 9D 43 EF 4E 52 14 C5 17 D7 41  ¥Ò-)$Ý.CïNR.Å.×A
000000E0  58 43 54 7F 1F FF 1E DF 04 EC A3 F9 A4 D2 BA 85  XCT..ÿ.ß.ì£ù¤Òº…
000000F0  E5 D9 3E 55 84 C4 10 E0 BF 69 38 E8 BB 08 E2 A5  åÙ>U„Ä.à¿i8è».â¥
00000100  58 1B A3 41 58 42 4D 02 5C 22 C3 EA D2 72 0D A6  X.£AXBM.\"ÃêÒr.¦
00000110  46 FE A3 22 EE BE C9 5B 5B 1C D1 D0 99 18 50 FF  Fþ£"î¾É[[.ÑÐ™.Pÿ
00000120  D2 8C 78 90 F1 57 C9 41 58 43 49 2C 9D 79 B5 12  ÒŒx.ñWÉAXCI,.yµ.
00000130  93 60 7F D6 11 14 93 FB 0F 54 3D F8 A1 8A 9E DA  "`.Ö.."û.T=ø¡ŠžÚ
```

Figure 9: The hash value of *AppxBlockMap.xml* in the package signature of *edge_update.appx*

2. The successful verification of the signed data in `AppxSignature.p7x` enables the `AppxSvc` service to verify the data integrity of the files whose hashes are part of the signed data, including `AppxBlockMap.xml`. The `AppxSvc` service does this by first computing the SHA-256 hash value of `AppxBlockMap.xml` and then comparing the computed value with the SHA-256 hash value of `AppxBlockMap.xml` in the previously verified signed data.
3. Once the `AppxSvc` service verifies the data integrity of `AppxBlockMap.xml`, the service verifies the integrity of the data blocks that `AppxBlockMap.xml` specifies (see Figure 6). The service does this by first computing the hash values of the data blocks and then comparing the computed values with the Base-64 encoded hash values in `AppxBlockMap.xml`.

The steps above ensure that the data in an *APPX* package is credible, with the overall process relying on a successful verification of the signed data in `AppxSignature.p7x`. However, for Windows to install an *APPX* package, also a malicious package, the system also has to verify that the package satisfies a set of trust criteria.

Previous research provides more background information on the steps above. This article focuses on the trust criteria that relate to the certificates in `AppxSignature.p7x` that the `AppxSvc` service uses to verify the signed data in `AppxSignature.p7x`.

These certificates represent the root of trust for the data integrity verification of an APPX package and for establishing trust in the package. In addition, in contrast to other *APPX* package-internal data structures for data integrity and trust verification, the certificates are more relevant entities from an operational perspective to end-users.

When the `AppxSvc` service installs the malicious `edge_update.appx`, the service executes:

- The *CertVerifyCertificateChainPolicy* function to validate the certificates in `AppxSignature.p7x` against the following certificate validation policies: *CERT_CHAIN_POLICY_AUTHENTICODE* (2), *CERT_CHAIN_POLICY_AUTHENTICODE_TS* (3), *CERT_CHAIN_POLICY_BASE* (1), and *CERT_CHAIN_POLICY_BASIC_CONSTRAINTS* (5). Among other certificate properties, `CertVerifyCertificateChainPolicy` validates whether the certificates are valid for code signing and whether the root certificate is trusted – present in a certificate store for trusted root certificates, such as *Trusted Root Certification Authorities*.
- The `CertGetCertificateChain` function to check the revocation status of the certificates in `AppxSignature.p7x`.

If any validation by `CertVerifyCertificateChainPolicy` or `CertGetCertificateChain` fails, the `AppxSvc` service does not establish trust in the *APPX* package and terminates the installation of `edge_update.appx`.

To demonstrate a failed validation by `CertVerifyCertificateChainPolicy`, Figure 10 depicts a scenario that I crafted: The `AppxSvc` service executes `CertVerifyCertificateChainPolicy` to validate the certificates in the package signature of `edge_update.appx` against the *CERT_CHAIN_POLICY_AUTHENTICODE* (2) and *CERT_CHAIN_POLICY_BASE* (1) certificate validation policies. The *CERT_CHAIN_POLICY_BASE* policy fails the validation of the certificates due to an untrusted root certificate – the root certificate issued by DigiCert Trusted Root G4 is not present in a certificate store for trusted root certificates. This causes the `AppxSvc` service to terminate the installation of `edge_update.appx`.

```
Breakpoint 0 hit
CRYPT32!CertVerifyCertificateChainPolicy:
00007ffb`34485560 4c8bdc          mov     r11,rsp
0:003> r @rcx
rcx=0000000000000002
0:003> g
Breakpoint 0 hit
CRYPT32!CertVerifyCertificateChainPolicy:
00007ffb`34485560 4c8bdc          mov     r11,rsp
0:003> r @rcx
rcx=0000000000000001
0:003> g

[...]
```

```
Add-AppxPackage : Deployment failed with HRESULT: 0x800B010A, A certificate chain could not be built to a trusted root
authority.
error 0x800B010A: The root certificate and all intermediate certificates of the signature in the app package or bundle
must be trusted.
NOTE: For additional information, look for [ActivityId] c5a4e929-920f-0000-60f0-a4c50f92d801 in the Event Log or use
the command line Get-AppPackageLog -ActivityID c5a4e929-920f-0000-60f0-a4c50f92d801
At line:1 char:1
+ Add-AppxPackage .\malicious_appx.appx
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : NotSpecified: (C:\Users\aleks\...cious_appx.appx:String) [Add-AppxPackage], Exception
    + FullyQualifiedErrorId : DeploymentError,Microsoft.Windows.Appx.PackageManager.Commands.AddAppxPackageCommand
```

Figure 10: *CertVerifyCertificateChainPolicy* fails the validation of the certificates in the package signature of *edge_update.appx* due to an untrusted root certificate

In practice, `CertVerifyCertificateChainPolicy` successfully validates the certificates in the package signature of the malicious `edge_update.appx`. This is because the root certificate, which is issued by DigiCert Trusted Root G4, is present in the *Trusted Root Certification Authorities* certificate store.

Revoked, or Not, That is the Question

The `AppxSvc` service executes the `CertGetCertificateChain` function to check the revocation status of the certificates in the package signature of `edge_update.appx`, starting from the end certificate in the certificate chain – the one issued to Foresee Consulting Inc. (see Figure 11).

```
0:008> db 00000246`cdbe9ad0 L0x7e8
00000246`cdbe9ad0  30 82 07 e4 30 82 05 cc-a0 03 02 01 02 02 10 0b  0...0...........
00000246`cdbe9ae0  c0 f1 8d a3 67 02 e3 02-db 17 0d 91 dc 92 02 30  ....g..........0
00000246`cdbe9af0  0d 06 09 2a 86 48 86 f7-0d 01 01 0b 05 00 30 69  ...*.H........0i
00000246`cdbe9b00  31 0b 30 09 06 03 55 04-06 13 02 55 53 31 17 30  1.0...U....US1.0
00000246`cdbe9b10  15 06 03 55 04 0a 13 0e-44 69 67 69 43 65 72 74  ...U....DigiCert
00000246`cdbe9b20  2c 20 49 6e 63 2e 31 41-30 3f 06 03 55 04 03 13  , Inc.1A0?..U...]
00000246`cdbe9b30  38 44 69 67 69 43 65 72-74 20 54 72 75 73 74 65  8DigiCert Truste
00000246`cdbe9b40  64 20 47 34 20 43 6f 64-65 20 53 69 67 6e 69 6e  d G4 Code Signin
00000246`cdbe9b50  67 20 52 53 41 34 30 39-36 20 53 48 41 33 38 34  g RSA4096 SHA384
00000246`cdbe9b60  20 32 30 32 31 20 43 41-31 30 1e 17 0d 32 31 31   2021 CA10...211
00000246`cdbe9b70  31 32 34 30 30 30 30 30-30 5a 17 0d 32 32 31 31  124000000Z..2211
00000246`cdbe9b80  32 33 32 33 35 39 35 39-5a 30 81 c0 31 1d 30 1b  23235959Z0..1.0.
00000246`cdbe9b90  06 03 55 04 0f 0c 14 50-72 69 76 61 74 65 20 4f  ..U....Private O
00000246`cdbe9ba0  72 67 61 6e 69 7a 61 74-69 6f 6e 31 13 30 11 06  rganization1.0..
[...]
00000246`cdbe9c10  0a 13 17 46 6f 72 65 73-65 65 20 43 6f 6e 73 75  ...Foresee Consu
00000246`cdbe9c20  6c 74 69 6e 67 20 49 6e-63 2e 31 20 30 1e 06 03  lting Inc.1 0...
00000246`cdbe9c30  55 04 03 13 17 46 6f 72-65 73 65 65 20 43 6f 6e  U....Foresee Con
00000246`cdbe9c40  73 75 6c 74 69 6e 67 20-49 6e 63 2e 30 82 02 22  sulting Inc.0.."
[...]
00000246`cdbe9f10  04 81 ad 30 81 aa 30 53-a0 51 a0 4f 86 4d 68 74  ...0..0S.Q.O.Mht
00000246`cdbe9f20  74 70 3a 2f 2f 63 72 6c-33 2e 64 69 67 69 63 65  tp://crl3.digice
00000246`cdbe9f30  72 74 2e 63 6f 6d 2f 44-69 67 69 43 65 72 74 54  rt.com/DigiCertT
00000246`cdbe9f40  72 75 73 74 65 64 47 34-43 6f 64 65 53 69 67 6e  rustedG4CodeSign
00000246`cdbe9f50  69 6e 67 52 53 41 34 30-39 36 53 48 41 33 38 34  ingRSA4096SHA384
00000246`cdbe9f60  32 30 32 31 43 41 31 2e-63 72 6c 30 53 a0 51 a0  2021CA1.crl0S.Q.
00000246`cdbe9f70  4f 86 4d 68 74 74 70 3a-2f 2f 63 72 6c 34 2e 64  O.Mhttp://crl4.d
00000246`cdbe9f80  69 67 69 63 65 72 74 2e-63 6f 6d 2f 44 69 67 69  igicert.com/Digi
00000246`cdbe9f90  43 65 72 74 54 72 75 73-74 65 64 47 34 43 6f 64  CertTrustedG4Cod
00000246`cdbe9fa0  65 53 69 67 6e 69 6e 67-52 53 41 34 30 39 36 53  eSigningRSA4096S
00000246`cdbe9fb0  48 41 33 38 34 32 30 32-31 43 41 31 2e 63 72 6c  HA3842021CA1.crl
[...]
```

Figure 11: The certificate issued to Foresee Consulting Inc. in the context of the *CertGetCertificateChain* function (in ASN.1 format)

`AppxSvc` executes `CertGetCertificateChain` such that the function verifies the revocation status of all certificates in the certificate chain, including the root certificate – the `dwFlags` parameter of `CertGetCertificateChain` has the value of **0x20000000** (*CERT_CHAIN_REVOCATION_CHECK_CHAIN*, see Figure 12).

```
Breakpoint 3 hit
CRYPT32!CertGetCertificateChain:
00007ffb`6e9b4ae0 488bc4          mov     rax,rsp
[...]
0:008> dps @rsp
000000ab`68ffef58  00007ffb`46fce19b appxdeploymentserver![...]
000000ab`68ffef60  00000000`00000000
000000ab`68ffef68  00007ffb`46fcd00c appxdeploymentserver![...]
000000ab`68ffef70  00000246`cb9746a0
000000ab`68ffef78  000000ab`68ffefb0
000000ab`68ffef80  000000ab`68ffefa0
000000ab`68ffef88  00000246`20000000
[...]
```

Figure 12: *CertGetCertificateChain* verifies the revocation status of all certificates in the certificate chain

Prior to the revocation of the end certificate issued to Foresee Consulting Inc., `CertGetCertificateChain` did not indicate an issue with the certificate chain in the package signature of `edge_update.appx`. This resulted in the `AppxSvc` service completing the installation of the malicious *APPX* package and therefore compromising the system. Windows places installed Windows Apps in the `%ProgramFiles%\WindowsApps` directory (see Figure 13).

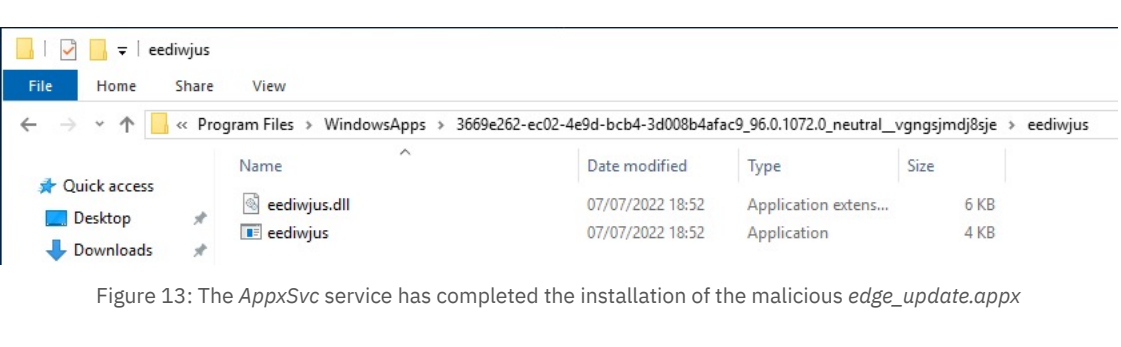


Figure 13: The *AppxSvc* service has completed the installation of the malicious *edge_update.appx*

After the revocation of the end certificate, `CertGetCertificateChain` indicates that a certificate in the package signature of `edge_update.appx` has been revoked by storing the *CERT_TRUST_IS_REVOKED* (**0x00000004**) error code in a *CERT_TRUST_STATUS* structure (see Figure 14). This results in the `AppxSvc` service terminating the installation of the malicious *APPX* package with an error (see Figure 15).

```
0:008> db 00000246`cb949c60
00000246`cb949c60  48 00 00 00 04 00 00 00-00 01 00 00 01 00 00 00  H...............
00000246`cb949c70  e0 9c 94 cb 46 02 00 00-00 00 00 00 00 00 00 00  ....F...........
[...]
```

Figure 14: *CertGetCertificateChain* stores the *CERT_TRUST_IS_REVOKED* (0x00000004) error code in a *CERT_TRUST_STATUS* structure

```
add-appxpackage : Deployment failed with HRESULT: 0x800B010C, A certificate was explicitly revoked by its issuer.
Deployment Add operation with target volume C: on Package 3669e262-ec02-4e9d-bcb4-3d008b4afac9_96.0.1072.0_neutral__vgngsjmdj8sje
from:  (malicious_appx.appx)  failed with error 0x800B010C. See http://go.microsoft.com/fwlink/?LinkId=235160 for help diagnosing
app deployment issues.
NOTE: For additional information, look for [ActivityId] b3f723a8-91de-0000-4277-f7b3de91d801 in the Event Log or use the command
line Get-AppPackageLog -ActivityID b3f723a8-91de-0000-4277-f7b3de91d801
At line:1 char:1
+ add-appxpackage C:\Users\<user>\Desktop\malicious_appx\malicious_appx. ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : NotSpecified: (C:\Users\<user>\...cious_appx.appx:String) [Add-AppxPackage], Exception
    + FullyQualifiedErrorId : DeploymentError,Microsoft.Windows.Appx.PackageManager.Commands.AddAppxPackageCommand
```

Figure 15: The *AppxSvc* service terminates the installation of the malicious *edge_update.appx* with an error

Recommendations for Users and Administrators

For Windows to install a Windows App that is packaged, for example, into an APPX package, the system first has to establish trust in the package. To this end, Windows verifies the data integrity of the package based on the package signature and evaluates whether the package satisfies certain trust criteria. Some of the trust criteria that relate to the certificates in the package signature are the following:

- If Windows App sideloading is not enabled on the system, the *APPX* package must originate from the Microsoft Store and be therefore counter-signed by Microsoft. App sideloading is not enabled by default on recent Windows versions. However, organizations may enable App sideloading on their managed devices or ask customers to turn App sideloading on in order to enable the deployment of Windows Apps built specifically for internal or customer use. These Windows Apps are known as LOB (line-of-business) Windows Apps.
- If Windows App sideloading is enabled on the system, the *APPX* package must be signed such that:
  - The system trusts the root certificate of the certificate chain in the signature – the certificate must be present in a certificate store for trusted root certificates.
  - No certificate in the chain is revoked at package installation time.

The majority of the malicious *APPX* packages that the security community has observed as part of attacks have satisfied the criteria above.

Recommendations for users and administrators for protecting against attacks that involve malware delivery via malicious Windows Apps include the following:

- For users:
  - Avoid downloading Windows Apps from the Microsoft Store without thoroughly examining relevant information, such as App vendor details, and number and quality of published user reviews. Be careful about typosquatting attempts – malicious Windows Apps with names that are very similar to legitimate, popular Windows Apps. Typosquatting is popular among malicious actors. Attackers have recently managed to plant malicious Windows Apps in the Microsoft Store. In addition, SentinelLabs has recently investigated typosquatting attacks against the crates.io Rust and the PyPI Python software repositories, referred to as CrateDepression and Pymafka.
  - Stay vigilant against phishing attacks and avoid installing software and software updates from unknown sources. Malicious Windows Apps often come under the disguise of critical software updates.
- For administrators: Malicious actors often use compromised legitimate code signing material, with end certificates that chain to trusted root certificates, to sign malicious Windows Apps. As this article shows, this enables the installation of the malicious App packages on victim systems. Therefore:
  - Make sure that the systems under your management can timely and correctly verify whether a certificate has been revoked. This includes unrestricted access to CRL (Certificate Revocation List) and OCSP (Online Certificate Status Protocol) URLs, and/or up-to-date local CRL and OCSP caches. Certificate revocation is a crucial measure against the malicious Windows App threat.
  - Make sure that the code signing material of your organization is kept secure. This is to prevent malicious actors from distributing malware masquerading as Windows Apps that originate from your organization.

CRIMEWARE | WINDOWS

ALEKSANDAR MILENKOSKI

Aleksandar Milenkoski is a Senior Threat Researcher at SentinelLabs, with expertise in reverse engineering, malware research, and threat actor analysis. Aleksandar has a PhD in system security and is the author of numerous research papers, book chapters, blog posts, and conference talks. His research has won awards from SPEC, the Bavarian Foundation for Science, and the University of Würzburg.