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.

- 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

- An overview of selected activities that the Windows system conducts when a user installs a malicious *APPX* package.

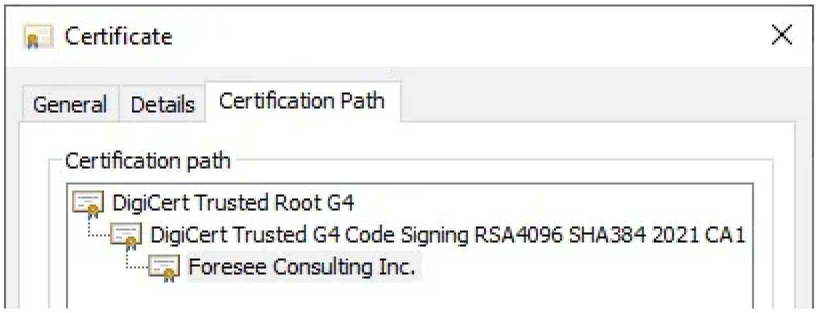
This enables a better understanding of what occurs on a system when a user falls prey to an attack that involves a malicious Windows App – from the operating system activities at first user interaction with the file to the malware deployment that the file triggers.

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.

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.

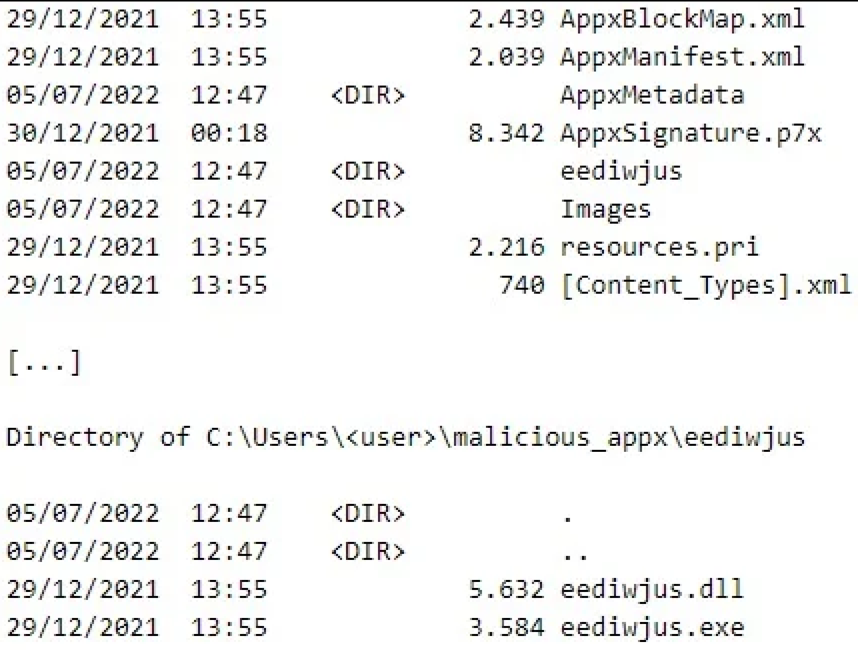
Figure 1 depicts the certificate chain of the digital signature of `edge_update.appx` before a Certificate Authority revoked the certificate.



DigiCert Trusted Root G4
DigiCert Trusted G4 Code Signing RSA4096 SHA384 2021 CA1
Foresee Consulting Inc.

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.



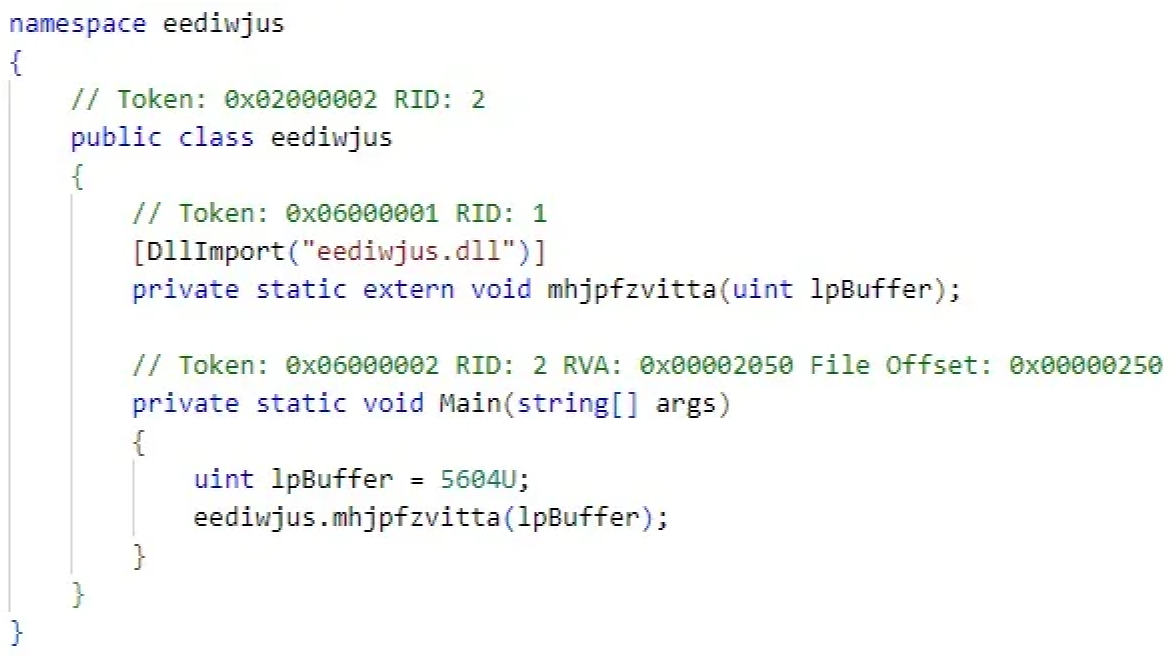
```
29/12/2021  13:55             2.439 AppxBlockMap.xml
29/12/2021  13:55             2.039 AppxManifest.xml
05/07/2022  12:47    <DIR>          AppxMetadata
30/12/2021  00:18             8.342 AppxSignature.p7x
05/07/2022  12:47    <DIR>          eediwjus
05/07/2022  12:47    <DIR>          Images
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*

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

`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.

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

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).

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`.

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

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`.

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

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

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 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`.

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).

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

certificate – the `dwFlags` parameter of `CertGetCertificateChain` has the value of **0×20000000** (*CERT_CHAIN_REVOCATION_CHECK_CHAIN*, see Figure 12).

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* (**0×00000004**) 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).

Figure 14: *CertGetCertificateChain* stores the *CERT_TRUST_IS_REVOKED* (*0×00000004*) error code in a *CERT_TRUST_STATUS* structure

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.

- 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

SentinelLABS

- 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

SHARE

PDF

ALEKSANDAR MILENKOSKI

Aleksandar Milenkoski is a Senior Threat Researcher at SentinelLabs. With expertise in malware research and focus on targeted attacks, he brings a blend of practical and deep insights to the forefront of cyber threat intelligence. Aleksandar has a PhD in system security and is the author of numerous reports on cyberespionage and high-impact cybercriminal operations, conference talks, and peer-reviewed research papers. From 2011 to 2014, he was a European Commission Marie Skłodowska-Curie Research Fellow. His research has won awards from SPEC, the Bavarian Foundation for Science, and the University of Würzburg.

Targets of Interest | Russian Organizations Increasingly Under Attack By Chinese APTs

LockBit 3.0 Update | Unpicking the Ransomware's Latest Anti-Analysis and Evasion Techniques

RELATED POSTS

Cloud Malware | A Threat Hunter's Guide to Analysis, Techniques and Delivery

OCTOBER 24 2024

Exploring the VirusTotal Dataset | An Analyst's Guide to Effective Threat Research

AUGUST 29 2024

SentinelLABS

Search ...

SIGN UP

Get notified when we post new content.

RECENT POSTS

Cloud Malware | A Threat Hunter's Guide to Analysis, Techniques and Delivery

OCTOBER 24, 2024

China's Influence Ops | Twisting Tales of Volt Typhoon at Home and Abroad

OCTOBER 16, 2024

LABS CATEGORIES

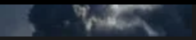
- Crimeware
- Security Research
- Advanced Persistent Threat
- Adversary
- LABScon
- Security & Intelligence

SENTINELLABS

In the era of interconnectivity, when markets, geographies, and jurisdictions merge in the melting pot of the digital domain, the perils of the threat ecosystem become unparalleled. Crimeware families achieve an unparalleled level of technical sophistication, APT groups are competing in fully-fledged cyber warfare, while once decentralized and scattered threat actors are forming adamant alliances of operating as elite corporate espionage teams.

RECENT POSTS

SentinelLABS



China's Influence Ops | Twisting Tales of Volt Typhoon at Home and Abroad
OCTOBER 16, 2024

Kryptina RaaS | From Unsellable Cast-Off to Enterprise Ransomware
SEPTEMBER 23, 2024

SIGN UP

Get notified when we post new content.

Business Email >

Twitter LinkedIn