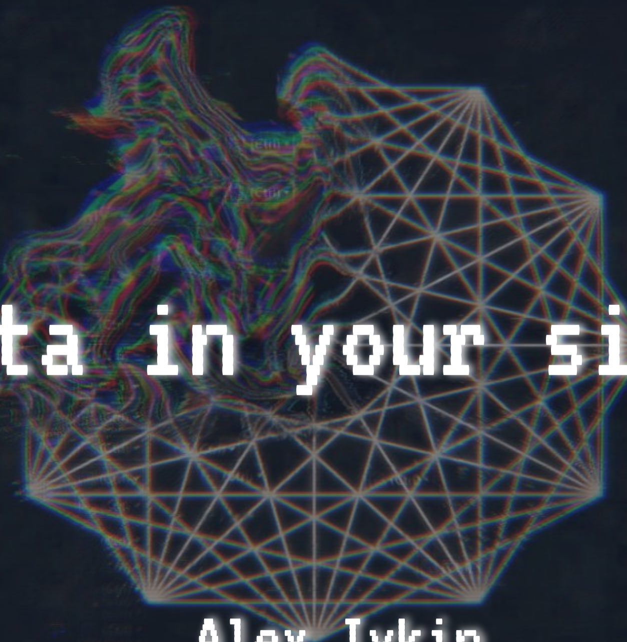


My data in your signed code

Alex Ivkin

zloizloizloi.tumblr.com

The startup challenge

The app is shipped from the server

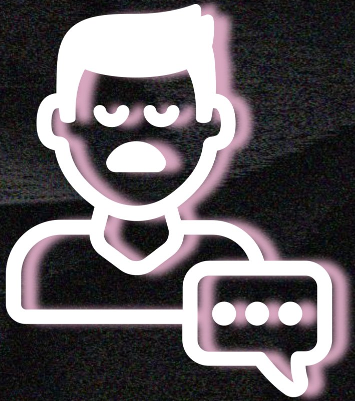
Each client configures the app to themselves

Can we make it better?

It is secure

Signed kernel driver

Signed application



What is Authenticode

Somebody created a hash of the code.

That somebody had access to a private key

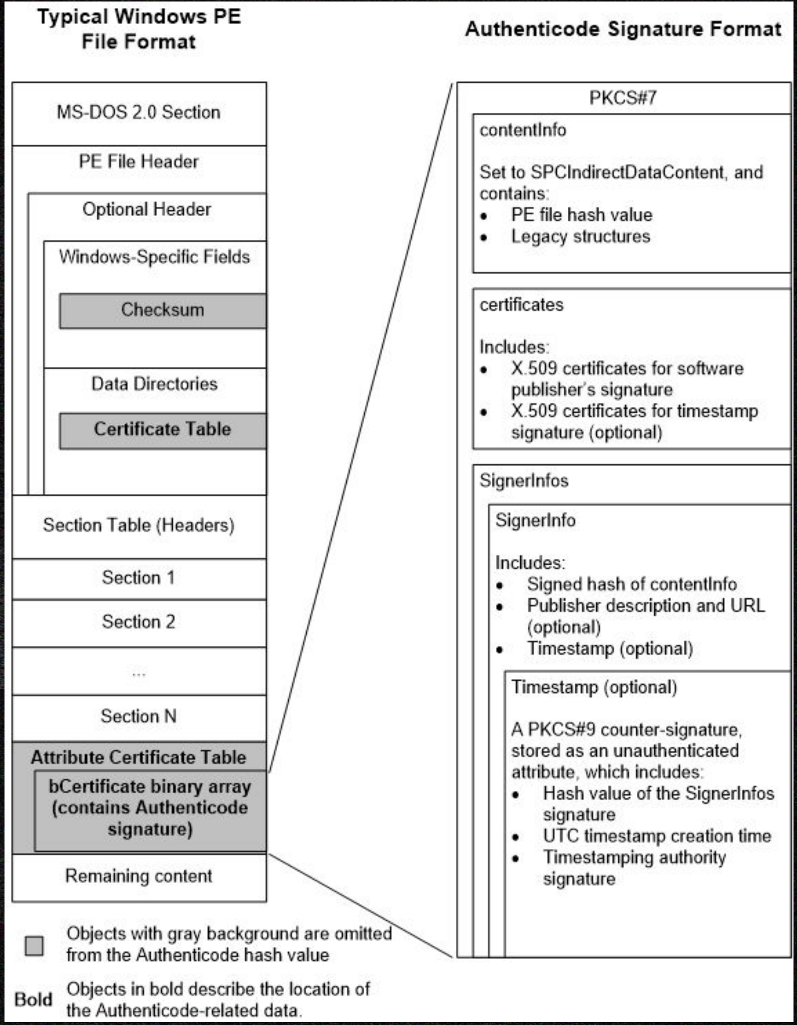
The public certificate was trusted by MS

Authenticode PE Hash is not for everything

How it works

Not the whole PE is signed

- Windows Checksum
- Certificate Table
- Certs
- Info past the end of the last section, e.g. debug info



Typical Windows PE File Format

MS-DOS 2.0 Section

PE File Header

Optional Header

Windows-Specific Fields

Checksum

Data Directories

Certificate Table

Section Table (Headers)

Section 1

Section 2

...

Section N

Attribute Certificate Table

bCertificate binary array (contains Authenticode signature)

Remaining content

Authenticode Signature Format

PKCS#7

contentInfo

Set to SPCIndirectDataContent, and contains:
- PE file hash value
- Legacy structures

certificates

Includes:
- X.509 certificates for software publisher's signature
- X.509 certificates for timestamp signature (optional)

SignerInfos

SignerInfo

Includes:
- Signed hash of contentInfo
- Publisher description and URL (optional)
- Timestamp (optional)

Timestamp (optional)

A PKCS#9 counter-signature, stored as an unauthenticated attribute, which includes:
- Hash value of the SignerInfos signature
- UTC timestamp creation time
- Timestamping authority signature

▨ Objects with gray background are omitted from the Authenticode hash value

Bold Objects in bold describe the location of the Authenticode-related data.

The Good

BH 2016 - Hiding data in the IMAGE_DATA_DIRECTORY by modifying dwLength of the WIN_CERTIFICATE structure

Custom PE loader to extract it

https://github.com/med0x2e/SigFlip

https://www.blackhat.com/docs/us-16/materials/us-16-Nipravsky-Certificate-Bypass-Hiding-And-Executing-Malware-From-A-Digitally-Signed-Executable-wp.pdf

The bad

No support for 64 bit

No support for DLL forwarding

Stopping PE loader breaks the process

The ugly

Known technique used by APT10/POTASSIUM in multiple campaigns

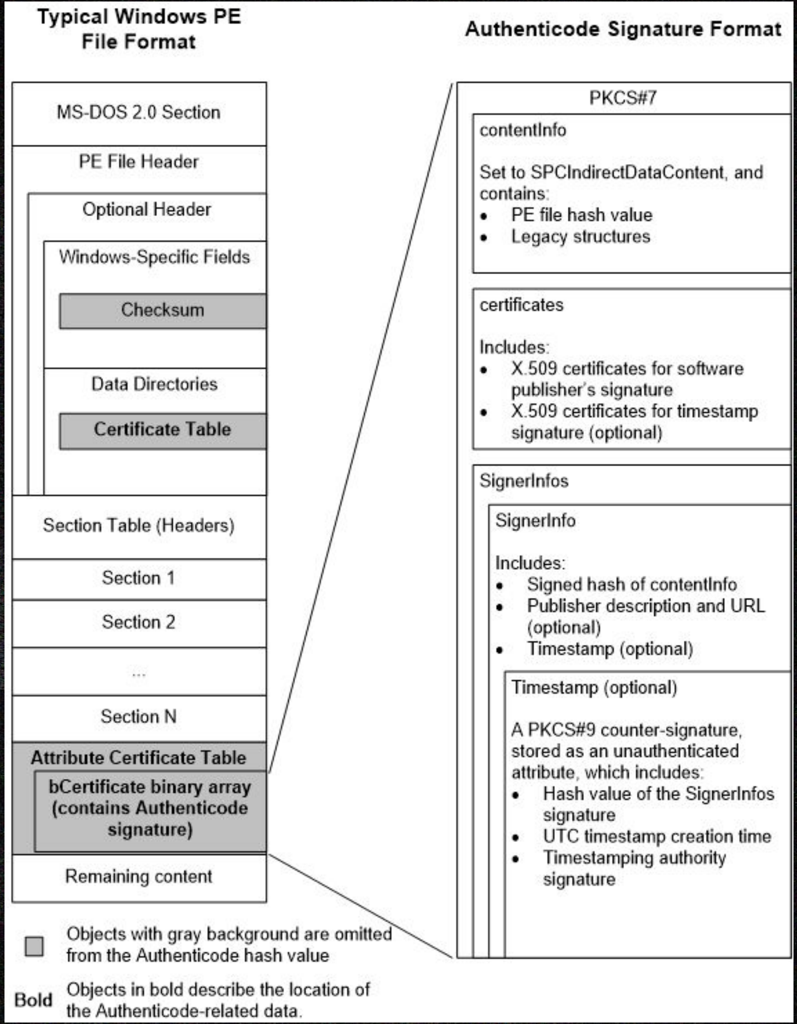
MS13-098 fix (KB2893294)

PKCS#7

RFC 5652

Authenticated signature

“Custom unauthenticated attributes”



Typical Windows PE File Format

- MS-DOS 2.0 Section
- PE File Header
- Optional Header
  - Windows-Specific Fields
    - Checksum
  - Data Directories
    - Certificate Table
- Section Table (Headers)
- Section 1
- Section 2
- ...
- Section N
- **Attribute Certificate Table**
  - **bCertificate binary array (contains Authenticode signature)**
- Remaining content

Authenticode Signature Format

PKCS#7

contentInfo

Set to SPCIndirectDataContent, and contains:

- PE file hash value
- Legacy structures

certificates

Includes:

- X.509 certificates for software publisher's signature
- X.509 certificates for timestamp signature (optional)

SignerInfos

SignerInfo

Includes:

- Signed hash of contentInfo
- Publisher description and URL (optional)
- Timestamp (optional)

Timestamp (optional)

A PKCS#9 counter-signature, stored as an unauthenticated attribute, which includes:

- Hash value of the SignerInfos signature
- UTC timestamp creation time
- Timestamping authority signature

▨ Objects with gray background are omitted from the Authenticode hash value

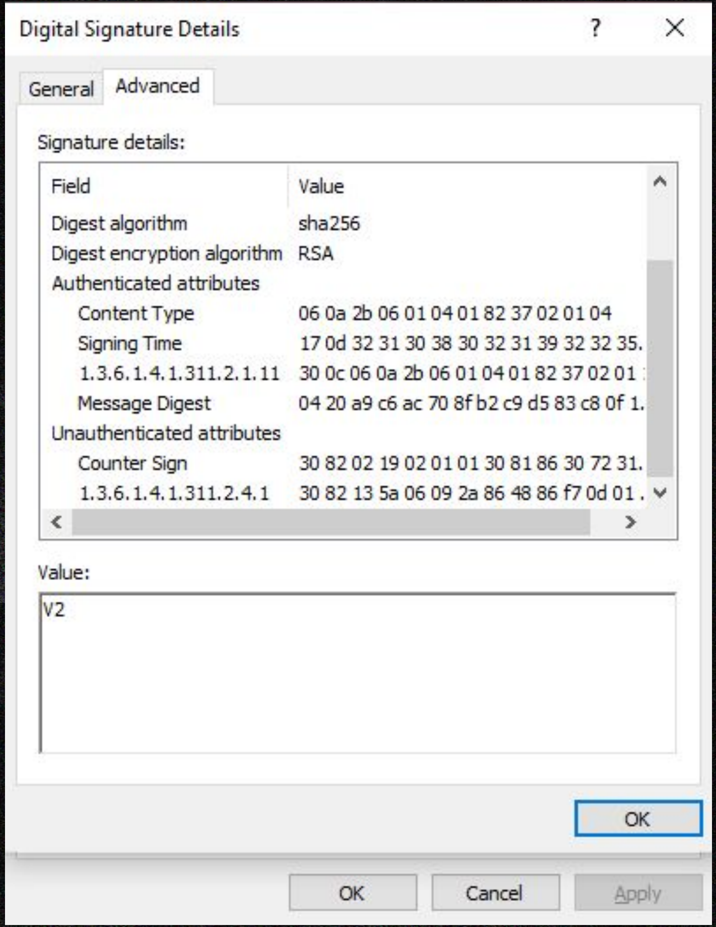
Bold Objects in bold describe the location of the Authenticode-related data.

Unauthed attributes

Mocking up the table

Non-standard

Easy to detect



Signature details:

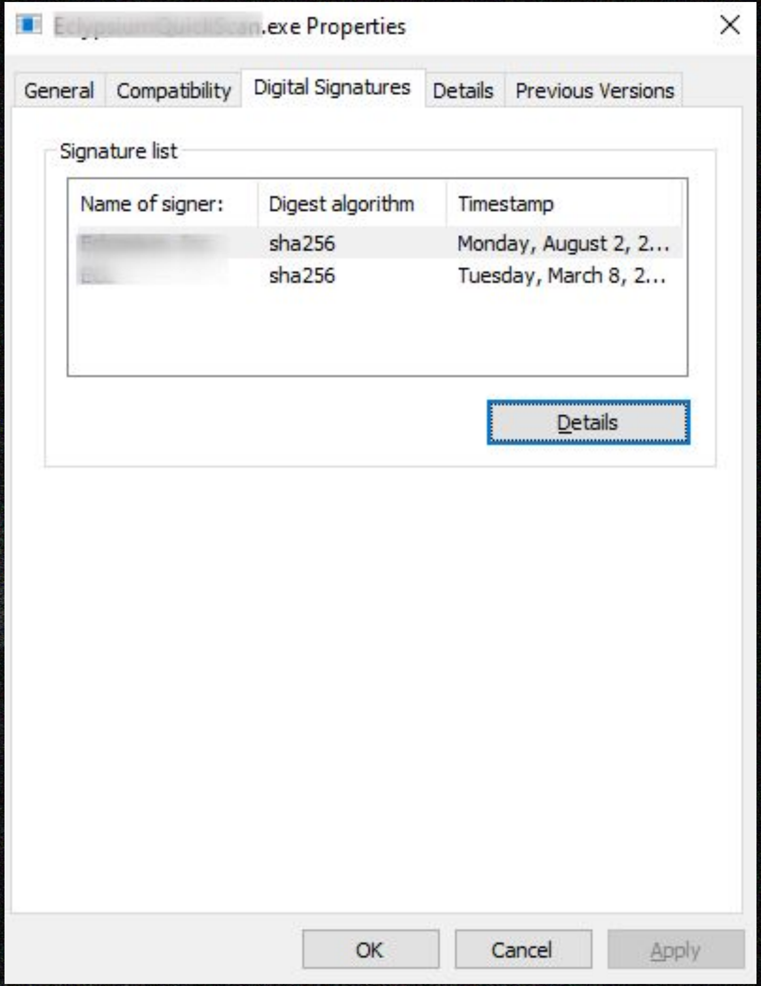
Field	Value
Digest algorithm	sha256
Digest encryption algorithm	RSA
Authenticated attributes	
Content Type	06 0a 2b 06 01 04 01 82 37 02 01 04
Signing Time	17 0d 32 31 30 38 30 32 31 39 32 32 35.
1.3.6.1.4.1.311.2.1.11	30 0c 06 0a 2b 06 01 04 01 82 37 02 01
Message Digest	04 20 a9 c6 ac 70 8f b2 c9 d5 83 c8 0f 1.
Unauthenticated attributes	
Counter Sign	30 82 02 19 02 01 01 30 81 86 30 72 31.
1.3.6.1.4.1.311.2.4.1	30 82 13 5a 06 09 2a 86 48 86 f7 0d 01 .

Value:

V2

Sign it again

What happens if we throw in another valid signature?



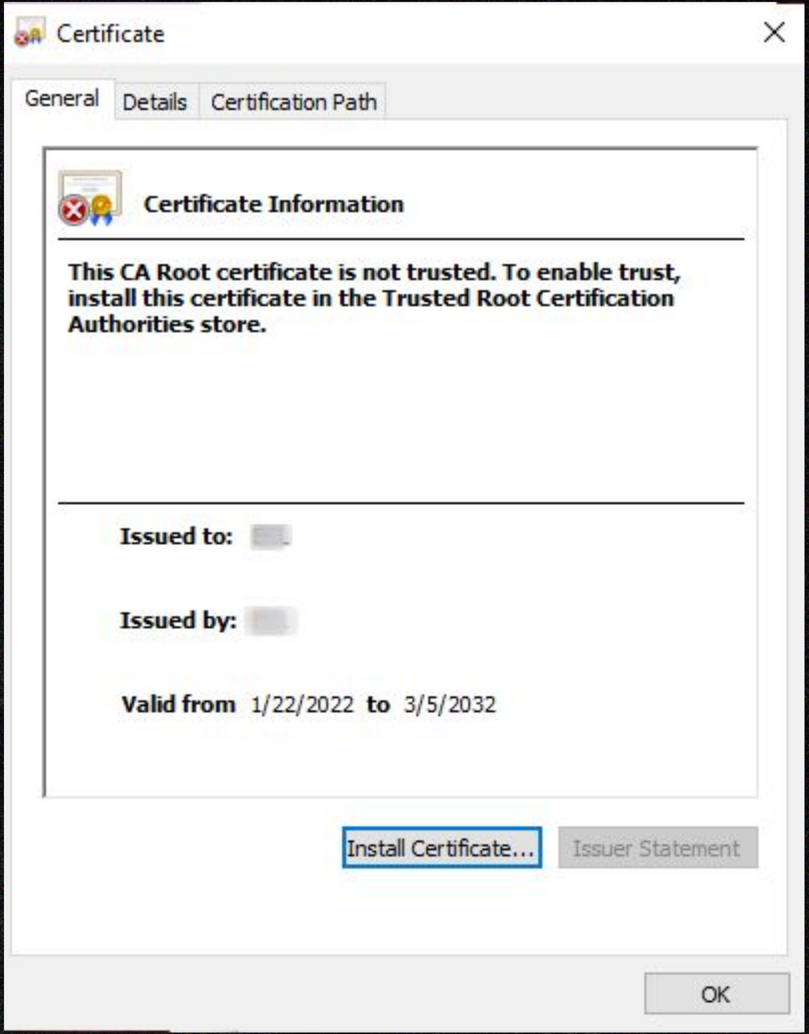
[redacted].exe Properties

Signature list

Name of signer:	Digest algorithm	Timestamp
[redacted]	sha256	Monday, August 2, 2...
[redacted]	sha256	Tuesday, March 8, 2...

Obviously

Untrusted



Certificate Information

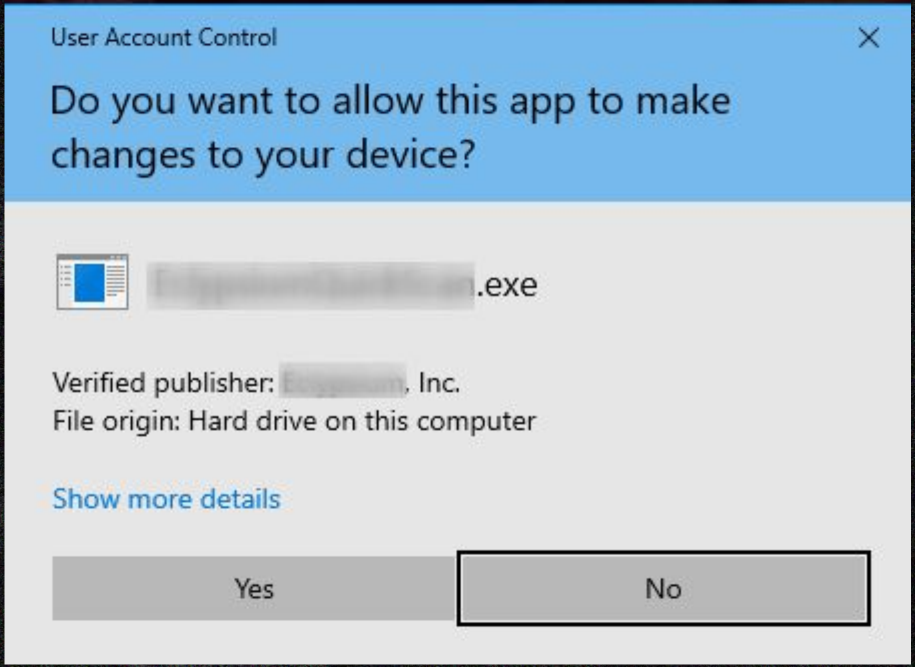
This CA Root certificate is not trusted. To enable trust, install this certificate in the Trusted Root Certification Authorities store.

Issued to: [redacted]

Issued by: [redacted]

Valid from 1/22/2022 to 3/5/2032

What if I run it?



User Account Control

Do you want to allow this app to make changes to your device?

[redacted].exe

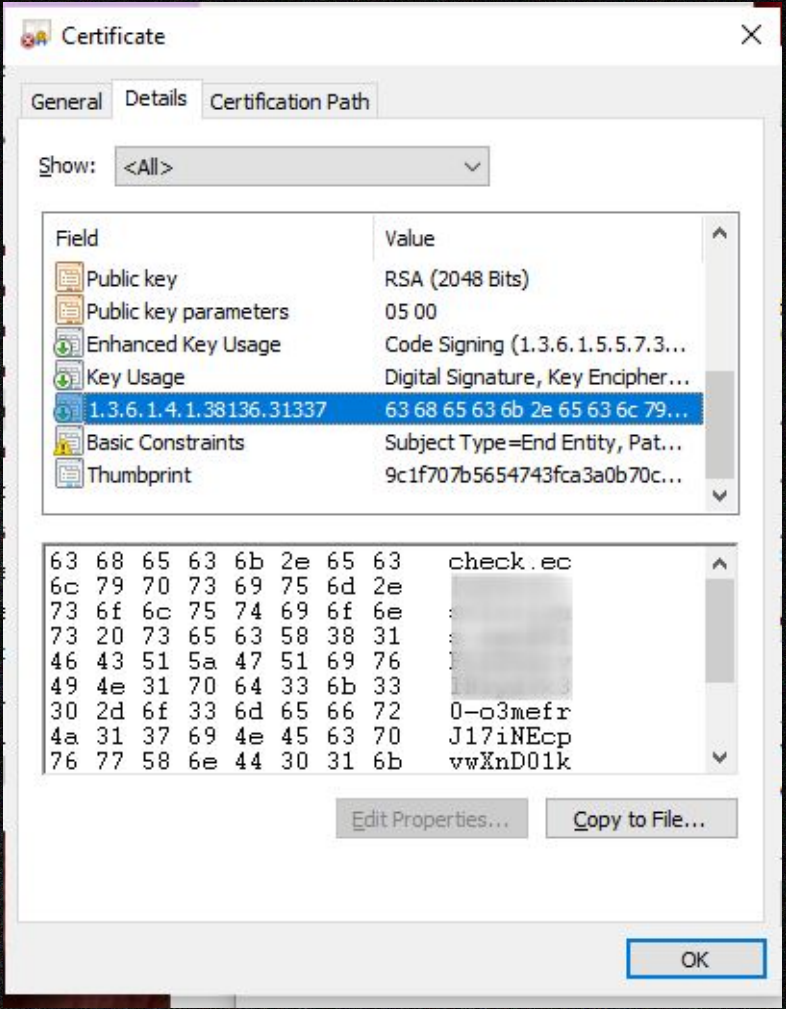
Verified publisher: [redacted], Inc.

File origin: Hard drive on this computer

Show more details

Game on

Packing custom attributes chock-full of data



Show: <All>

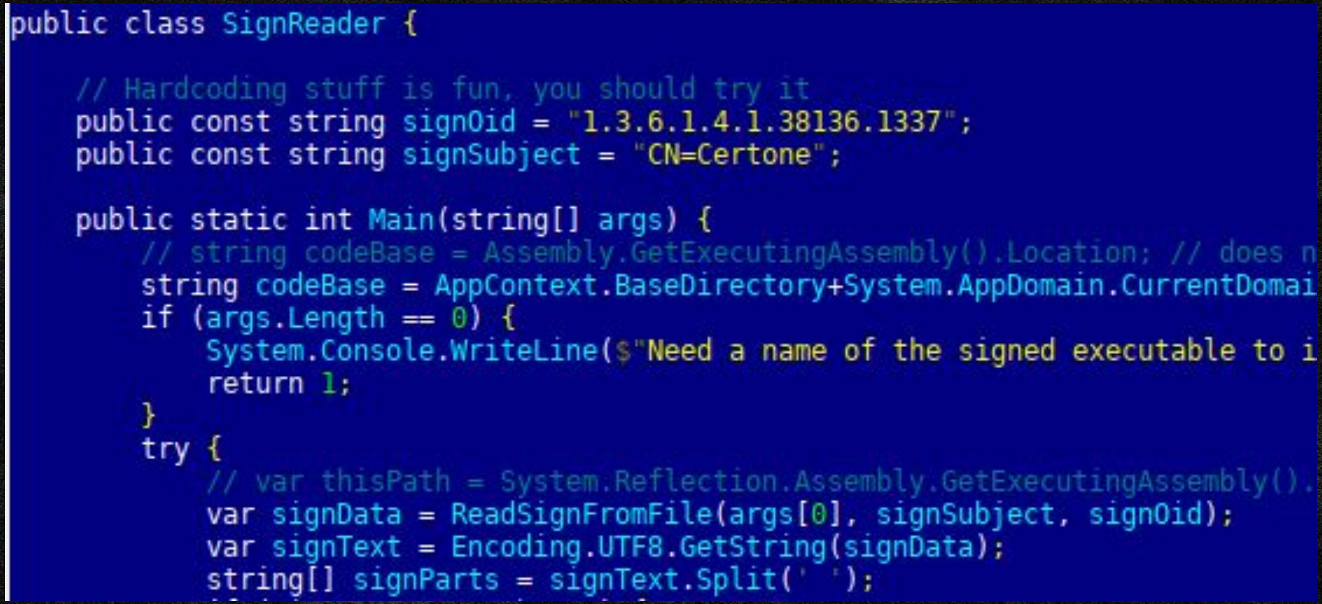
Field	Value
Public key	RSA (2048 Bits)
Public key parameters	05 00
Enhanced Key Usage	Code Signing (1.3.6.1.5.5.7.3...
Key Usage	Digital Signature, Key Encipher...
1.3.6.1.4.1.38136.31337	63 68 65 63 6b 2e 65 63 6c 79...
Basic Constraints	Subject Type=End Entity, Pat...
Thumbprint	9c1f707b5654743fca3a0b70c...

```
63 68 65 63 6b 2e 65 63    check.ec
6c 79 70 73 69 75 6d 2e    [redacted]
73 6f 6c 75 74 69 6f 6e
73 20 73 65 63 58 38 31
46 43 51 5a 47 51 69 76
49 4e 31 70 64 33 6b 33
30 2d 6f 33 6d 65 66 72    0-o3mefr
4a 31 37 69 4e 45 63 70    J17iNEcp
76 77 58 6e 44 30 31 6b    vwXnD01k
```

Injecting data

1. Start with a signed code
2. Create a self-signed Code Signing certificate
3. Add custom attributes
4. signtool/osslsigncode
5. Profit

Extracting data at runtime



```
public class SignReader {

    // Hardcoding stuff is fun, you should try it
    public const string signOid = "1.3.6.1.4.1.38136.1337";
    public const string signSubject = "CN=Certone";

    public static int Main(string[] args) {
        // string codeBase = Assembly.GetExecutingAssembly().Location; // does n
        string codeBase = AppContext.BaseDirectory+System.AppDomain.CurrentDomai
        if (args.Length == 0) {
            System.Console.WriteLine($"Need a name of the signed executable to i
            return 1;
        }
        try {
            // var thisPath = System.Reflection.Assembly.GetExecutingAssembly().
            var signData = ReadSignFromFile(args[0], signSubject, signOid);
            var signText = Encoding.UTF8.GetString(signData);
            string[] signParts = signText.Split(' ');
```

Personalized installers

Release as usual
Have the webserver run injection on each download
The users just run it

Mmm-mmm-goodness

Hiding data in “benign” executable files

Spreading shellcode in multiple signed execs

Bypassing entropy detectors

Dynamically encrypting shellcode with keys in certs

Modding vulnerable kernel drivers to bypass “known bad” hash detections by EDR https://github.com/alexivkin/signreader-cs

https://github.com/alexivkin/signwriter-cs

Everything is obvious

(once you know the answer)

https://github.com/alexivkin

@alexivkinx

EOF