Practical exploitations of cryptographic flaws in Windows

Presentation

Yolan Romailler

- Applied CryptoGopher
- CTF dabbler
- Board game amateur
- @anomalroil

Sylvain PELISSIER

- Security researcher
- Applied Cryptography
- CTF player
- @Pelissier_S
- @ipolit@mastodon.social

CVE-2020-0601 | Windows CryptoAPI Spoofing Vulnerability

Security Vulnerability

Published: 01/14/2020 | Last Updated : 01/16/2020
MITRE CVE-2020-0601

A spoofing vulnerability exists in the way Windows CryptoAPI (Crypt32.dll) validates Elliptic Curve Cryptography (ECC) certificates.

An attacker could exploit the vulnerability by using a spoofed code-signing certificate to sign a malicious executable, making it appear the file was from a trusted, legitimate source. The user would have no way of knowing the file was malicious, because the digital signature would appear to be from a trusted provider.

A successful exploit could also allow the attacker to conduct man-in-the-middle attacks and decrypt confidential information on user connections to the affected software.

The security update addresses the vulnerability by ensuring that Windows CryptoAPI completely validates ECC certificates.

Acknowledgements

National Security Agency

Microsoft recognizes the efforts of those in the security community who help us

Crypt32.dll

- Cryptography library coming with Microsoft Windows.
- Provide symmetric, asymmetric crypto and PRNGs.
- Used by Microsoft Edge and Google Chrome for TLS certificates.
- Used by Windows for binary signatures.
- Supports ECC only since 2017.

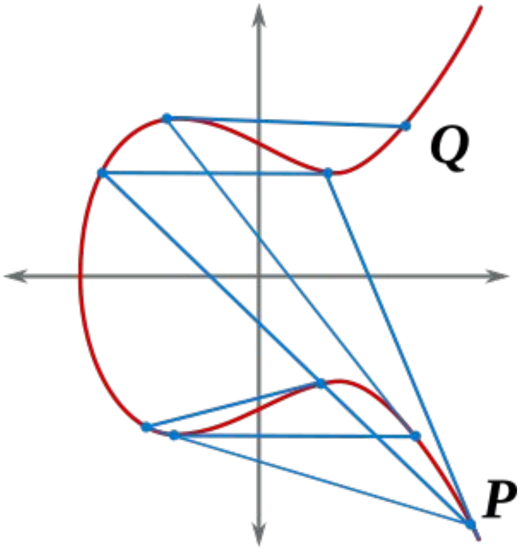
Elliptic Curve

A curve is defined by an equation $y^2=x^3+ax+b$

- over a finite field: GF(**p**)
- by its coefficients **a** and **b**
- by a generator **G** (or base point)

The “order” of a curve is its number of points.

Discrete logarithm



$Q = P + \ldots + P = k \cdot P$

Easy to compute **Q = k·P**
Hard to compute **k**
from **Q** and **P**

Elliptic Curves

```
$ openssl ecparam -list_curves
  secp128r1 : SECG curve over a 128 bit prime field
  secp128r2 : SECG curve over a 128 bit prime field
  secp160k1 : SECG curve over a 160 bit prime field
  secp160r1 : SECG curve over a 160 bit prime field
  secp160r2 : SECG/WTLS curve over a 160 bit prime field
  secp192k1 : SECG curve over a 192 bit prime field
  secp224k1 : SECG curve over a 224 bit prime field
  secp224r1 : NIST/SECG curve over a 224 bit prime field
  secp256k1 : SECG curve over a 256 bit prime field
  secp384r1 : NIST/SECG curve over a 384 bit prime field
  secp521r1 : NIST/SECG curve over a 521 bit prime field
  prime192v1: NIST/X9.62/SECG curve over a 192 bit prime field
```

Elliptic Curves

```
$ openssl ecparam -name secp384r1 -text -param_enc explicit
Field Type: prime-field
Prime:
    00:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:
    ...
A:
    00:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:
    ...
B:
    00:b3:31:2f:a7:e2:3e:e7:e4:98:8e:05:6b:e3:f8:
    ...
Generator (uncompressed):
    04:aa:87:ca:22:be:8b:05:37:8e:b1:c7:1e:f3:20:
```

Named curve

```
$ openssl ec -in p384-private-key.pem -text
read EC key
Private-Key: (384 bit)
priv:
    bd:1a:36:8f:72:ef:57:c9:74:a3:19:bf:e4:0a:7a:
    ...
pub:
    04:ef:1b:79:31:5b:e2:2c:fe:b6:da:48:44:0f:08:
    ...
ASN1 OID: secp384r1
NIST CURVE: P-384
```

Explicit parameters

```
$ openssl ec -in p384-private-key-explicit.pem -text
read EC key
Private-Key: (384 bit)
priv:
    54:f5:e3:8b:ef:a0:6b:7d:51:a2:15:d2:ee:c5:69:
   ...
Generator (uncompressed):
    04:aa:87:ca:22:be:8b:05:37:8e:b1:c7:1e:f3:20:
    ad:74:6e:1d:3b:62:8b:a7:9b:98:59:f7:41:e0:82:
    54:2a:38:55:02:f2:5d:bf:55:29:6c:3a:54:5e:38:
    72:76:0a:b7:36:17:de:4a:96:26:2c:6f:5d:9e:98:
    bf:92:92:dc:29:f8:f4:1d:bd:28:9a:14:7c:e9:da:
    31:13:b5:f0:b8:c0:0a:60:b1:ce:1d:7e:81:9d:7a:
    43:1d:7c:90:ea:0e:5f
```

Explicit parameters

- o namedCurve identifies all the required values for a particular set of elliptic curve domain parameters to be represented by an object identifier. This choice MUST be supported. See Section 2.1.1.1.

- o implicitCurve allows the elliptic curve domain parameters to be inherited. This choice MUST NOT be used.

- o specifiedCurve, which is of type SpecifiedECDomain type (defined in [X9.62]), allows all of the elliptic curve domain parameters to be explicitly specified. This choice MUST NOT be used. See Section 5, "ASN.1 Considerations".

Private and public keys

Private key: $\mathbf{k}$

Public key: $\mathbf{Q = k \cdot G}$

Private and public keys

Private key: **k**

Public key: $\mathbf{Q = k \cdot G}$

Generator defined in the specification of the **named** elliptic curve.

Private key crafting

Private key: **k**

Public key: $\mathbf{Q = k \cdot G}$

If **G** is not verified:

for a given public key **Q**

Choose your own $\mathbf{k' = 2}$

Compute your own $\mathbf{G' = 2^{-1} \cdot Q}$

Same public key: $\mathbf{Q = k' \cdot G'}$

Private key crafting

Private key: **k**

Public key: $\mathbf{Q = k \cdot G}$

If **G** is not verified:

for a given public key **Q**

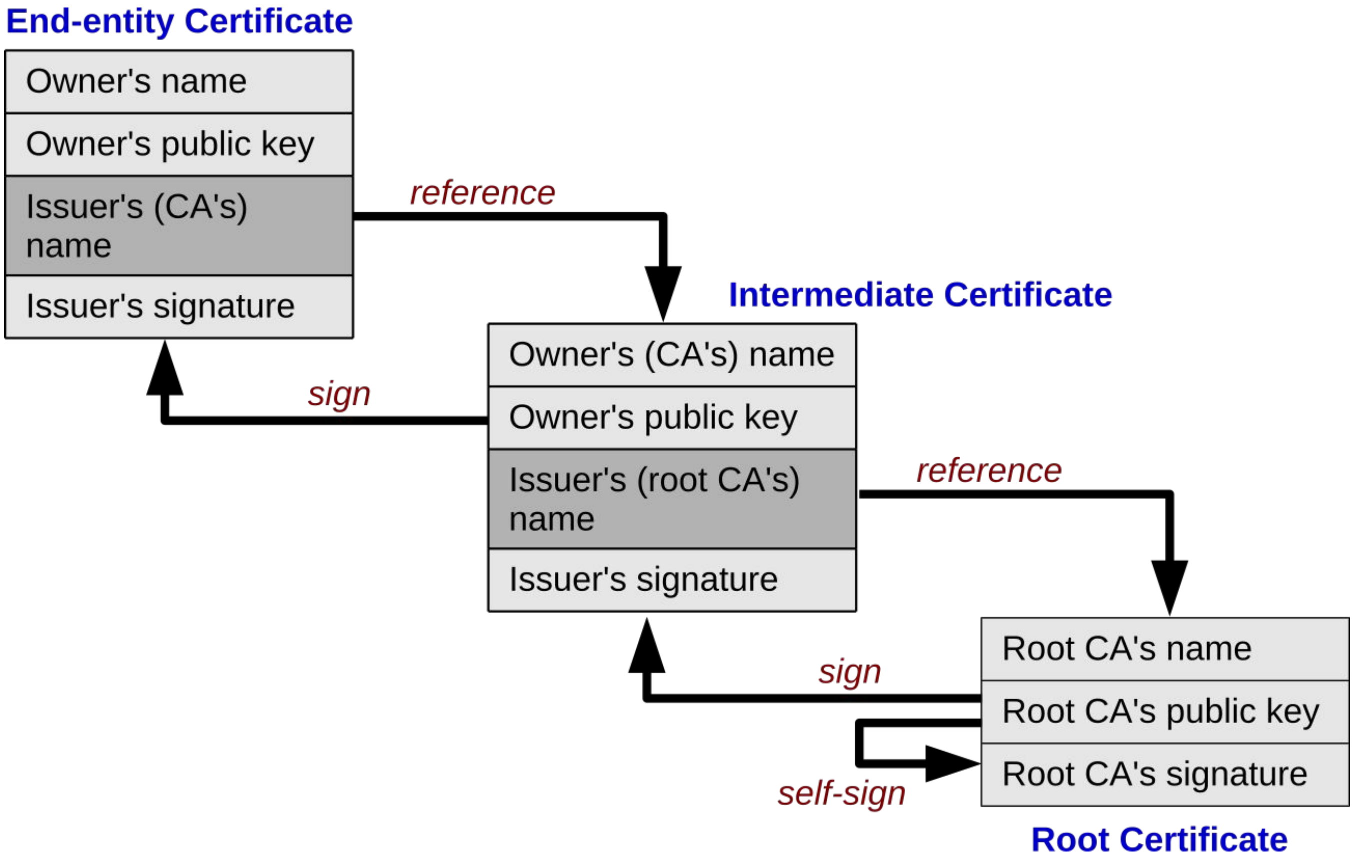
Choose your own $\mathbf{k' = 1}$

Compute your own $\mathbf{G' = Q}$

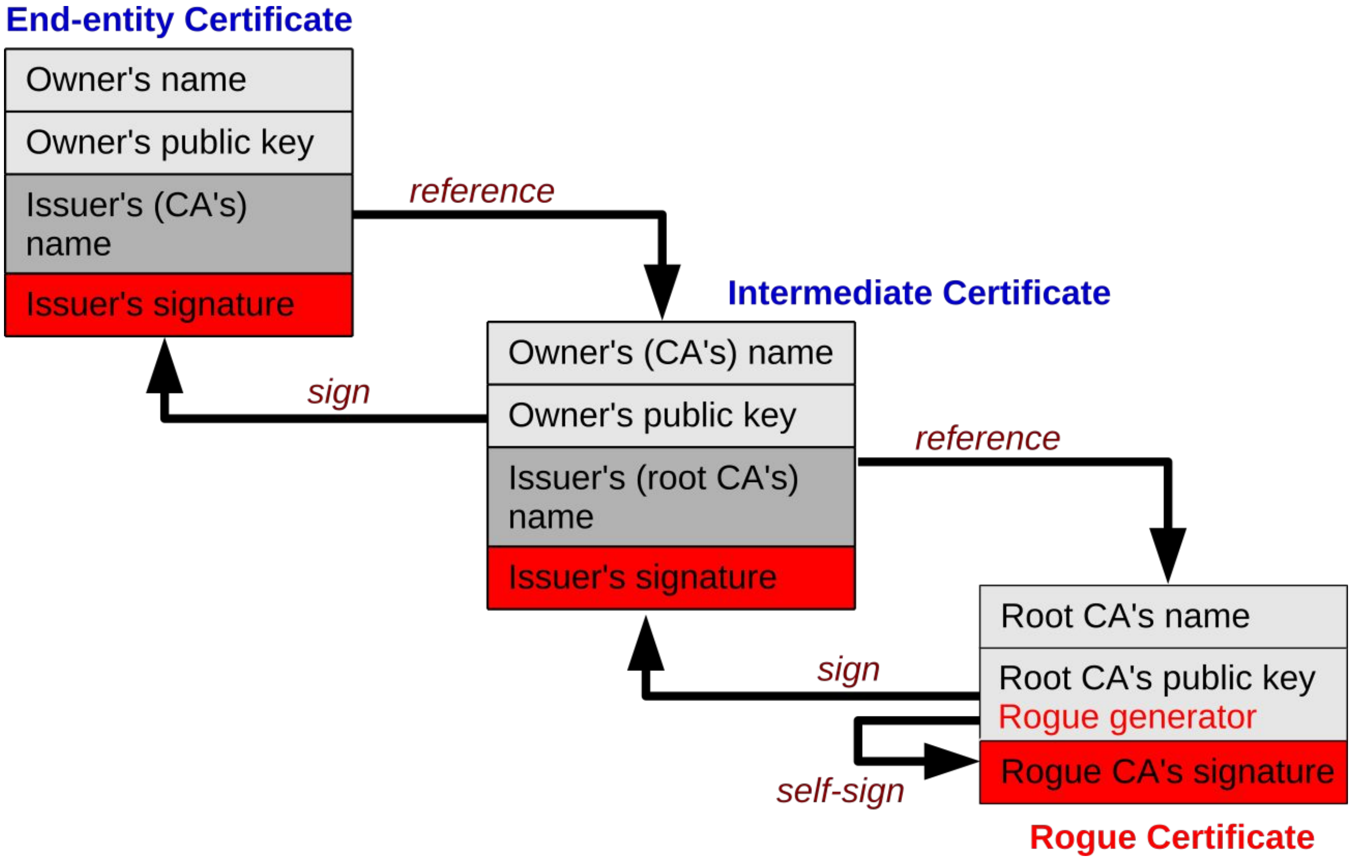
Same public key: $\mathbf{Q = G'}$

Works with 1 !

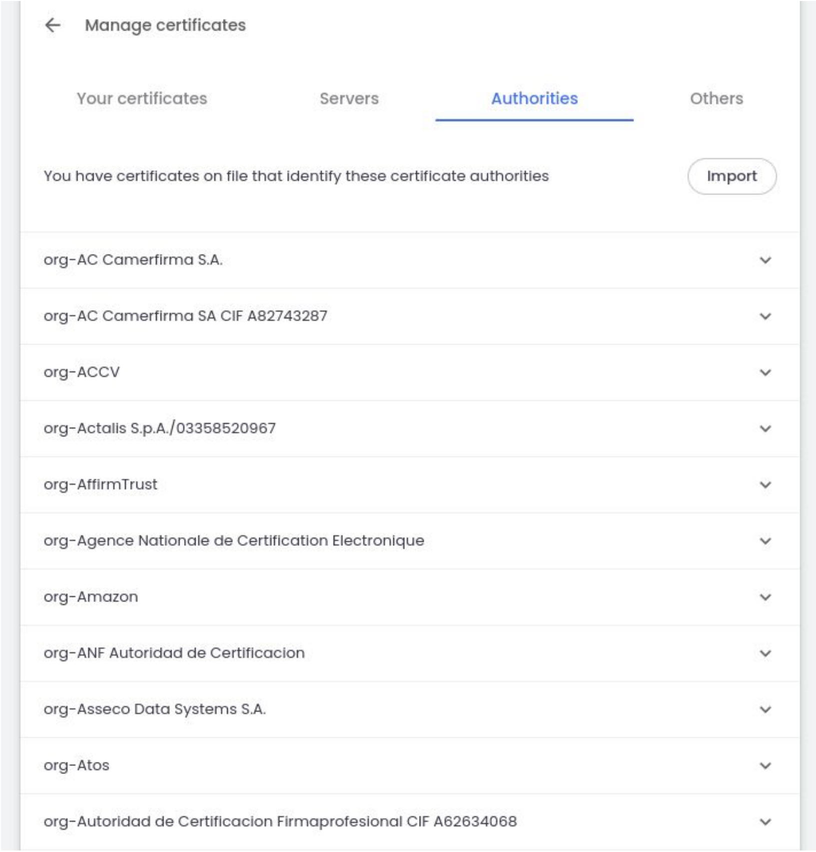
Chain of trust



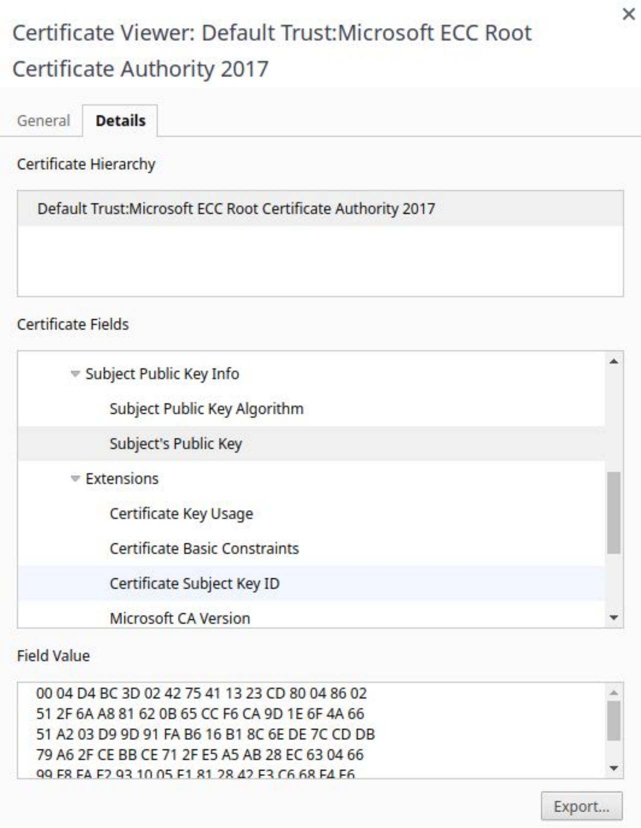
Chain of ~~trust~~ fools



PoC || GTFO


Manage certificates
Your certificates
Servers
Authorities
Others
You have certificates on file that identify these certificate authorities
Import
org-AC Camerfirma S.A.
org-AC Camerfirma SA CIF A82743287
org-ACCV
org-Actalis S.p.A./03358520967
org-AffirmTrust
org-Agence Nationale de Certification Electronique
org-Amazon
org-ANF Autoridad de Certificacion
org-Asseco Data Systems S.A.
org-Atos
org-Autoridad de Certificacion Firmaprofesional CIF A62634068

PoC || GTFO


Certificate Viewer: Default Trust:Microsoft ECC Root Certificate Authority 2017
General
Details
Certificate Hierarchy
Default Trust:Microsoft ECC Root Certificate Authority 2017
Certificate Fields
Subject Public Key Info
Subject Public Key Algorithm
Subject's Public Key
Extensions
Certificate Key Usage
Certificate Basic Constraints
Certificate Subject Key ID
Microsoft CA Version
Field Value
00 04 D4 BC 3D 02 42 75 41 13 23 CD 80 04 86 02
51 2F 6A A8 81 62 0B 65 CC F6 CA 9D 1E 6F 4A 66
51 A2 03 D9 9D 91 FA B6 16 B1 8C 6E DE 7C CD DB
79 A6 2F CE BB CE 71 2F E5 A5 AB 28 EC 63 04 66
Export...

Private key

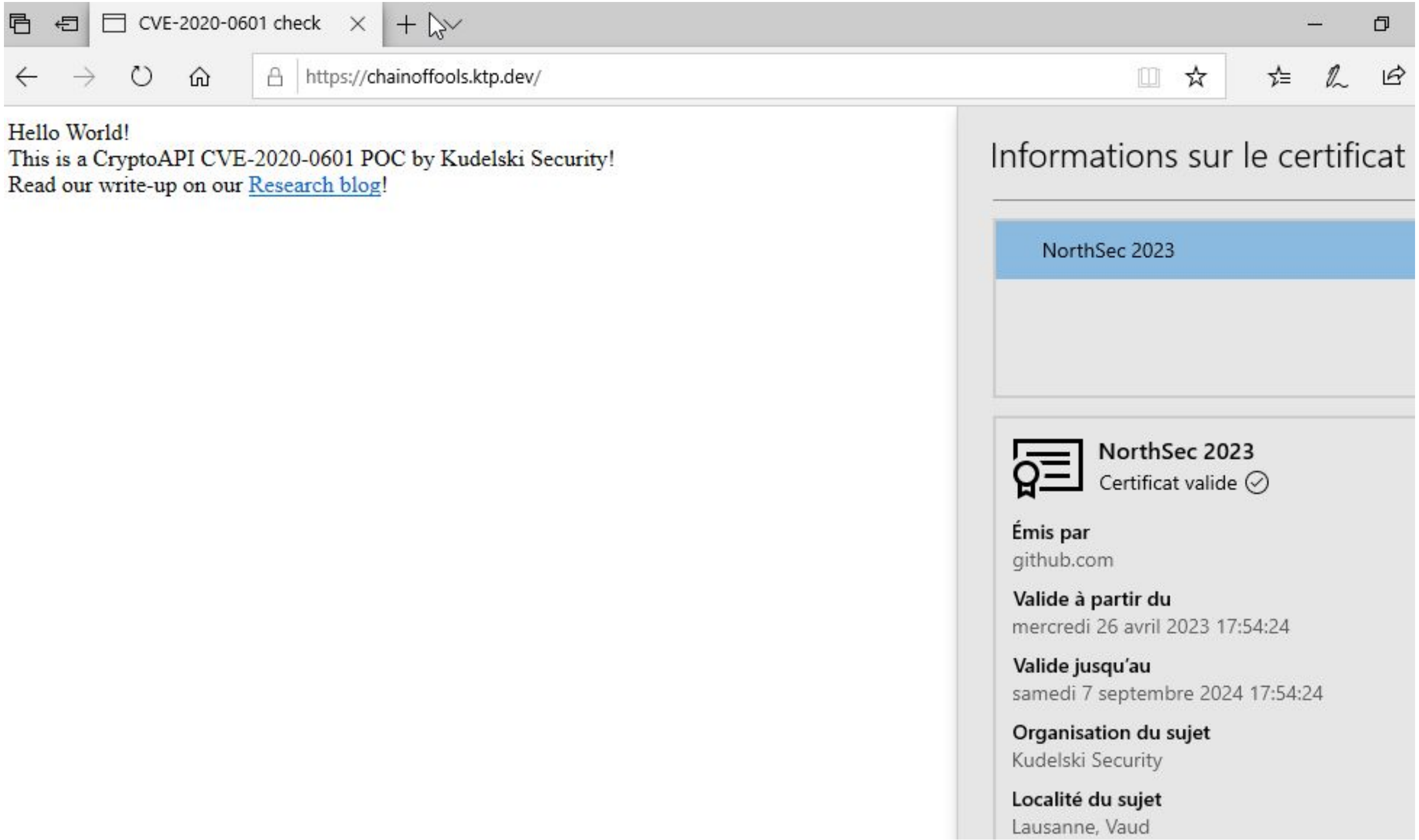
```
$ gen-key.py RootCert.pem
$ openssl ec -in p384-key-rogue.pem -text
Private-Key: (384 bit)
priv:
    00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:
    00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:
    00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:
    00:00:02
pub:
    04:d4:bc:3d:02:42:75:41:13:23:cd:80:04:86:02:
    51:2f:6a:a8:81:62:0b:65:cc:f6:ca:9d:1e:6f:4a:
    66:51:a2:03:d9:9d:91:fa:b6:16:b1:8c:6e:de:7c:
    cd:db:79:a6:2f:ce:bb:ce:71:2f:e5:a5:ab:28:ec:
    63:04:66:99:f8:fa:f2:93:10:05:e1:81:28:42:e3:
```

Generator

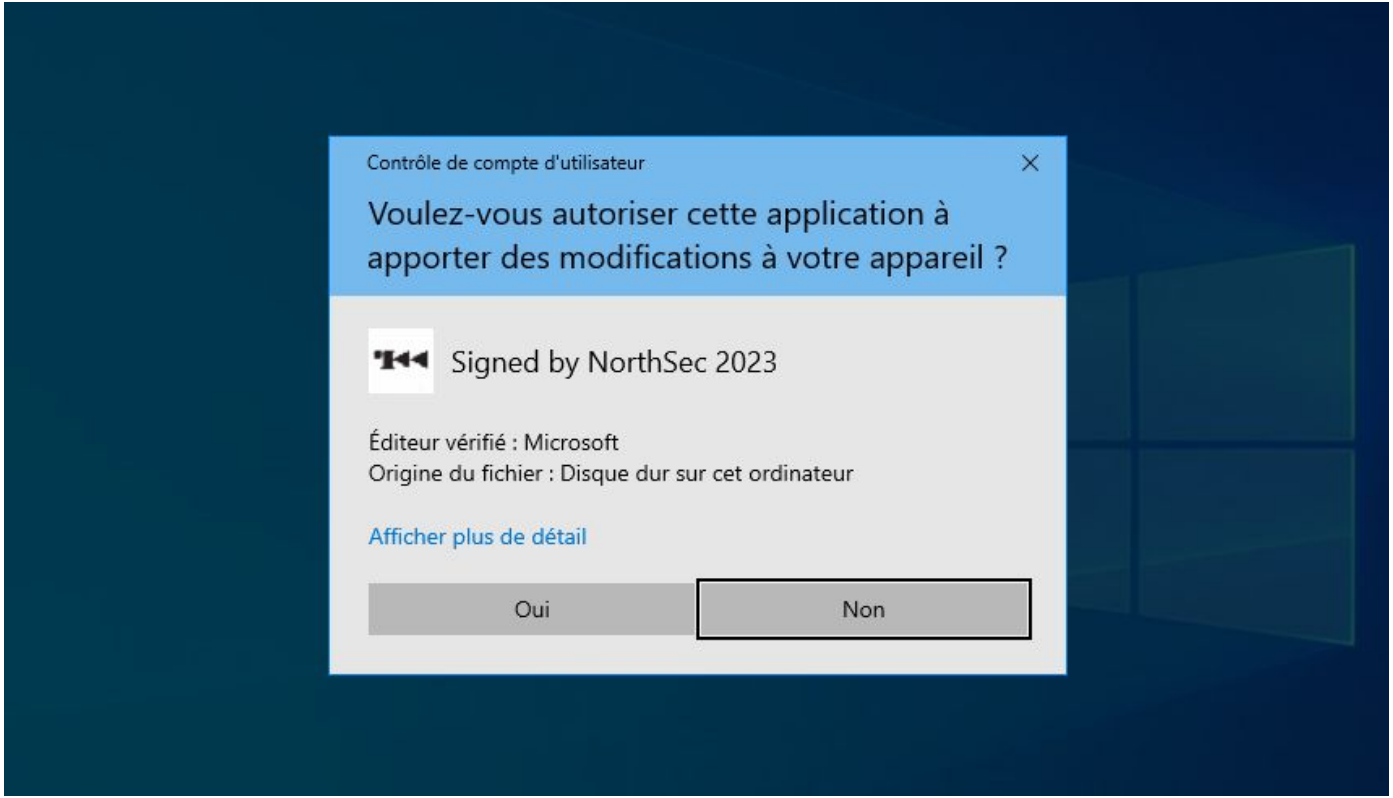
```
$ openssl ec -in p384-key-rogue.pem -text
...
Generator (uncompressed):
    04:43:1f:be:a6:2d:85:8b:84:3e:38:7b:d2:90:49:
    ea:70:55:a0:e6:2e:65:b9:17:b2:83:df:d2:d2:0b:
    8c:3b:65:b2:5d:f1:23:2f:df:40:46:81:7b:21:02:
    73:b0:65:05:e9:e9:0e:84:3e:d9:78:7a:a4:8d:64:
    a0:58:b6:4d:6c:f6:2f:0e:9e:0a:9b:8f:12:cb:64:
    e9:aa:ff:97:aa:60:5b:52:55:9a:dc:4b:b3:25:30:
    69:79:ad:99:70:5d:31
Order:
    00:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:
    ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:c7:63:4d:81:f4:
    37:2d:df:58:1a:0d:b2:48:b0:a7:7a:ec:ec:19:6a:
    cc:c5:29:73
```

Demo time

Website impersonation



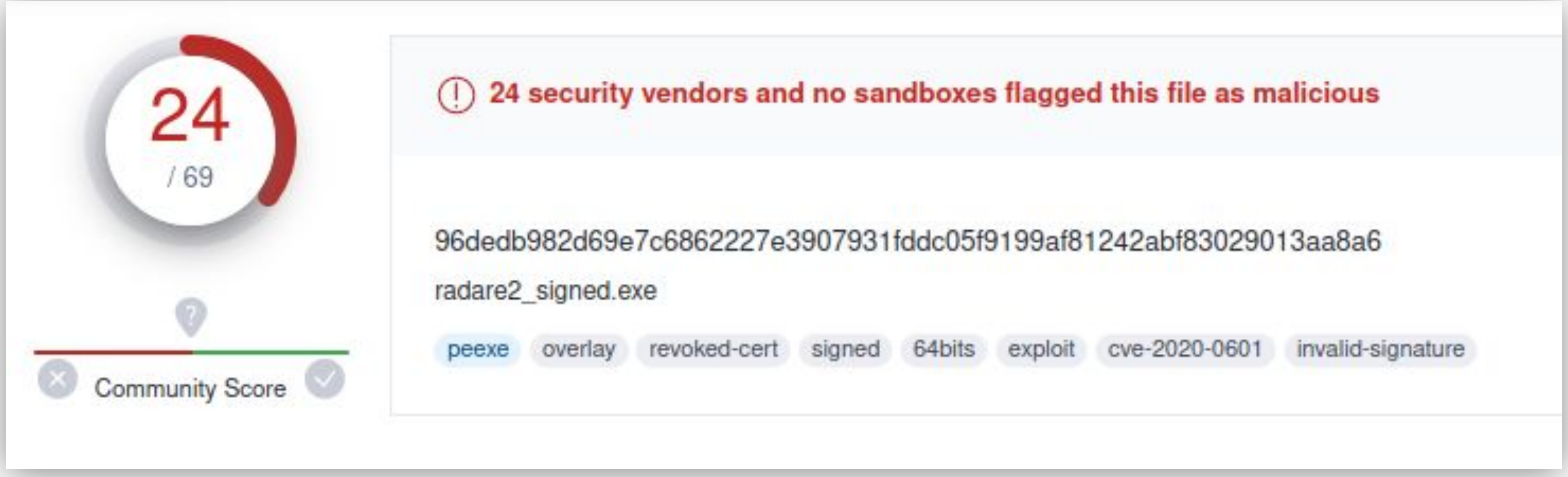
Binary signing



Possibilities

- Meddler in the Middle
- Impersonation
- Signed malwares
- *May* escape anti-virus

Possibilities



Correction and detection

Correction: Install patch KB4534306

Detection: Explicit parameters should trigger a warning

```
[0x00407354]> yara add crypto_signatures.yar
[0x00407354]> yara scanS
CRC32_poly_Constant
0x00003f41: $c0 : 20 83 b8 ed
CRC32_poly_Constant
0x00003f41: $c0 : 20 83 b8 ed
ecc_order
0x001619f7: $secp384r1 : ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff c7 63 4d 81 f4 37 2d df 58 1a
0d b2 48 b0 a7 7a ec ec 19 6a cc c5 29 73
```

In the wild


HELPNETSECURITY
TOP 10 MOST EXPLOITED VULNERABILITIES FROM 2020
1. CVE-2020-0796: Windows SMBv3 Client/Server Remote Code Execution Vulnerability (codename: SMBGhost)
2. CVE-2020-5902: F5 Networks BIG-IP TMUI RCE vulnerability
3. CVE-2020-1472: Microsoft Netlogon Elevation of Privilege (codename: Zerologon)
4. CVE-2020-0601: Windows CryptoAPI Spoofing Vulnerability (codename: CurveBall)
5. CVE-2020-14882: Oracle WebLogic Server RCE
6. CVE-2020-1938: Apache Tomcat AJP File Read/Inclusion Vulnerability (codename: GhostCat)
7. CVE-2020-3452: Cisco ASA and Firepower Path Traversal Vulnerability
8. CVE-2020-0688: Microsoft Exchange Server Static Key Flaw Could Lead to Remote Code Execution
9. CVE-2020-16898: Windows TCP/IP Vulnerability (codename: Bad Neighbor)
10. CVE-2020-1350: Critical Windows DNS Server RCE (codename: SIGRed)
SOURCE: vFeed

Windows CryptoAPI Spoofing Vulnerability

CVE-2022-34689
Security Vulnerability

Released: Oct 11, 2022

Assigning CNA: Microsoft

CVE-2022-34689

Exploitability

The following table provides an exploitability assessment for this vulnerability at the time of original publication.

Publicly Disclosed	Exploited	Latest Software Release
No	No	Exploitation More Likely

Acknowledgements

UK National Cyber Security Centre (NCSC) and the National Security Agency (NSA)

Microsoft recognizes the efforts of those in the security community who help us prote

PoC

- Akamai were the first to [publish a PoC] for Meddler in the Middle attacks along with [a blog post].
- Published colliding certificates (no secret keys) and MitM scripts.
- Not customizable for your needs.

Exploiting a Critical Spoofing Vulnerability in Windows CryptoAPI

Akamai Security Research

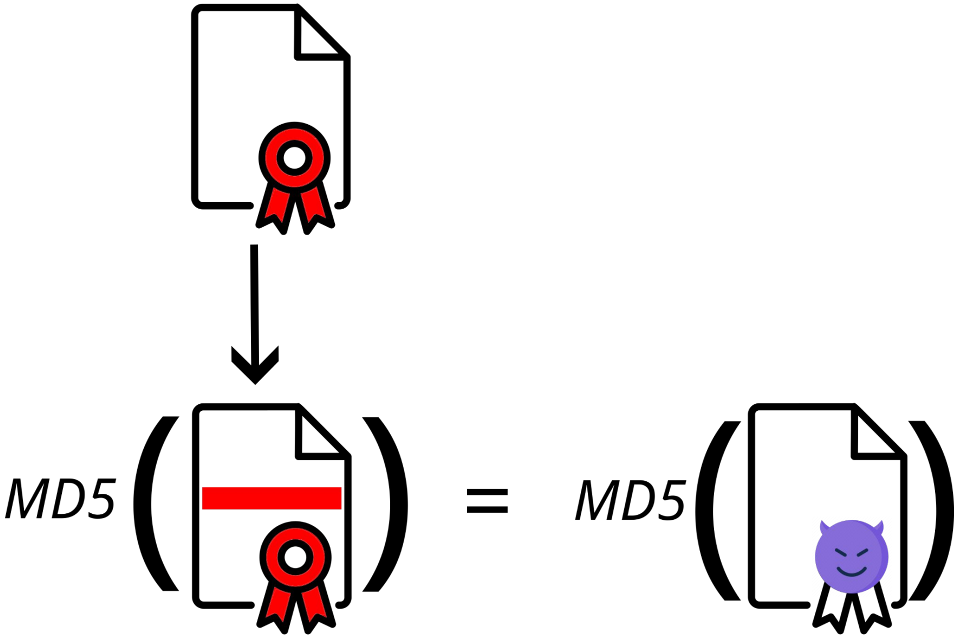
January 25, 2023

Culprit: certificate cache

- A verified certificate may be cached by Windows
- The cache is a hashtable using the MD5 hash of the cert
- If a certificate is in cache it is not verified again
- Bypass signature verification.

CVE-2022-34689

MD5 is known to be vulnerable to chosen-prefix collision attacks since **2005**!



Certificate tweaking

The MD5 is taken over the full TBS certificate but …

```
CertificateList  ::=  SEQUENCE  {
     tbsCertList          TBSCertList,
     signatureAlgorithm   AlgorithmIdentifier,
     signatureValue       BIT STRING  }
```

```
AlgorithmIdentifier  ::=  SEQUENCE  {
     algorithm               OBJECT IDENTIFIER,
     parameters              ANY DEFINED BY algorithm OPTIONAL  }
```

To cache or not to cache

- It applies only if the certificate is cached

Value	Meaning
CERT_CHAIN_CACHE_END_CERT 0x00000001	Information in the end certificate is cached. By default, information in all certificates except the end certificate is cached as a chain is built. Setting this flag extends the caching to the end certificate.

Code signing

- In the advisory the vulnerability should apply to code signing
- It applies only if the certificate is cached

Value	Meaning
CERT_CHAIN_CACHE_END_CERT 0x00000001	Information in the end certificate is cached. By default, information in all certificates except the end certificate is cached as a chain is built. Setting this flag extends the caching to the end certificate.

- We expected intermediate to be cached …
- It seems for code signing verification they are not cached…

Code signing

- github.com/kudelskisecurity/northsec_crypto_api_attacks
- Contributions welcomed !

Conclusion

- With Cryptography implementations, details matter
- Do not implement and use deprecated features or algorithms like MD5
- More crypto attacks this afternoon with Matt Cheung!
- Next time you see an announcement from NSA, bindiff FTW

Questions