Exercise 1 & 2: Understanding Certificate Verification

Question 1: What is the difference between the two commands?

The two commands are:

```
openssl s_client -connect twitter.com:443
openssl s_client -connect twitter.com:443 -CAfile ca-certificates.crt
```

Answer: The difference is that the second command explicitly specifies a file containing trusted Certificate Authority (CA) certificates using the `-CAfile` parameter. The first command attempts to connect without explicitly providing trusted CA certificates (though it may use system defaults), while the second command uses the specified ca-certificates.crt file to verify the server's certificate chain against known trusted CAs.

Question 2: What does the verify error mean?

Answer: The verify error in the first command (if present) means that OpenSSL cannot validate the certificate chain because it doesn't have access to the trusted root CA certificates. Without the CA certificates, OpenSSL cannot verify that the server's certificate was signed by a trusted authority, resulting in a "unable to get local issuer certificate" or similar error. This doesn't mean the connection is impossible, but it cannot be verified as trustworthy.

Exercise 3: Examining the Certificate

After saving the certificate as `twitter_com.cert`, use:

```
openssl x509 -in twitter_com.cert -text
```

Question 3: What is stored in an X.509 certificate?

Answer: An X.509 certificate contains the following key fields:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            06:53:96:80:57:c6:c6:dc:f5:31:36:db:c5:c4:9f:bc:9e:a8
        Signature Algorithm: ecdsa-with-SHA384
        Issuer: C = US, O = Let's Encrypt, CN = E6
        Validity
            Not Before: Aug 19 20:42:10 2025 GMT
            Not After : Nov 17 20:42:09 2025 GMT
        Subject: CN = twitter.com
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:ba:5c:dc:0e:2c:0f:e4:16:59:54:06:42:42:c0:
                    8a:7c:a1:8c:cd:1e:46:89:76:88:6a:ea:dd:41:8e:
                    3b:aa:6e:3b:d3:48:2a:a2:e4:51:97:d9:d0:b3:52:
                    a9:3e:d7:4b:2e:d0:cf:27:64:0e:f7:7c:77:b3:cf:
                    59:f5:51:03:a6
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Subject Key Identifier:
                7B:A9:AE:D2:07:70:3A:3D:71:04:96:D1:F1:1F:F9:B0:4B:6E:DC:F1
            X509v3 Authority Key Identifier:
                93:27:46:98:03:A9:51:68:8E:98:D6:C4:42:48:DB:23:BF:58:94:D2
            Authority Information Access:
                CA Issuers - URI:http://e6.i.lencr.org/
            X509v3 Subject Alternative Name:
                DNS:*.twitter.com, DNS:cdn.syndication.twitter.com,
DNS:twitter.com
            X509v3 Certificate Policies:
                Policy: 2.23.140.1.2.1
            X509v3 CRL Distribution Points:
                Full Name:
                  URI:http://e6.c.lencr.org/41.crl
            CT Precertificate SCTs:
                Signed Certificate Timestamp:
                    Version   : v1 (0x0)
                    Log ID    : CC:FB:0F:6A:85:71:09:65:FE:95:9B:53:CE:E9:B2:7C:
                                22:E9:85:5C:0D:97:8D:B6:A9:7E:54:C0:FE:4C:0D:B0
                    Timestamp : Aug 19 21:40:40.445 2025 GMT
                    Extensions: none
                    Signature : ecdsa-with-SHA256
                                30:45:02:21:00:F5:28:54:C5:A8:E1:8A:DB:77:A5:BA:
                                57:33:94:64:A5:8C:AE:CB:ED:D3:31:44:79:4A:DF:53:
                                33:ED:5F:51:7C:02:20:48:3B:37:FF:BB:81:DC:FA:72:
                                D7:45:A5:DF:B1:30:9C:87:EE:FD:65:A7:69:6F:ED:3A:
                                E8:21:7C:81:B4:ED:6A
                Signed Certificate Timestamp:
                    Version   : v1 (0x0)
                    Log ID    : DD:DC:CA:34:95:D7:E1:16:05:E7:95:32:FA:C7:9F:F8:
                                3D:1C:50:DF:DB:00:3A:14:12:76:0A:2C:AC:BB:C8:2A
                    Timestamp : Aug 19 21:40:40.459 2025 GMT
                    Extensions: none
                    Signature : ecdsa-with-SHA256
                                30:45:02:21:00:80:63:29:29:82:54:B0:7E:DF:0E:77:
                                6A:6C:DB:55:C4:DF:95:2A:10:20:CA:F1:FD:78:C7:DF:
                                D2:F9:35:B1:A4:02:20:1A:EA:F7:5F:17:77:E4:96:47:
                                ED:CC:75:DA:5B:A6:9C:7A:58:4F:38:2C:4E:84:92:26:
                                E6:8A:92:F6:A3:49:CF
```

```
    Signature Algorithm: ecdsa-with-SHA384
    Signature Value:
        30:66:02:31:00:ec:b1:01:b6:01:df:d1:8e:8c:fd:4c:a7:01:
        e1:45:b7:27:ac:ba:d3:77:57:57:d0:f8:2c:89:8d:e8:b2:6a:
        f5:14:61:17:28:58:ff:d5:7e:f4:61:c6:fc:f1:04:a5:57:02:
        31:00:a4:98:d5:1b:42:25:94:15:d9:59:95:20:d3:0d:94:69:
        60:aa:c9:98:f1:9b:7a:9e:25:5d:fa:4f:f7:56:35:30:97:db:
        58:17:a4:6e:a1:fd:a7:44:14:c9:3d:0c:6a:aa
-----BEGIN CERTIFICATE-----
MIIDsDCCAzWgAwIBAgISBlOWgFfGxtz1MTbbxcSfvJ6oMAoGCCqGSM49BAMDMDIx
CzAJBgNVBAYTAlVTMRYwFAYDVQQKEw1MZXQncyBFbmNyeXB0MQswCQYDVQQDEwJF
NjAeFw0yNTA4MTkyMDQyMTBaFw0yNTExMTcyMDQyMDlaMBYxFDASBgNVBAMTC3R3
aXR0ZXIuY29tMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEulzcDiwP5BZZVAZC
QsCKfKGMzR5GiXaIaurdQY47qm4700gqouRRl9nQs1KpPtdLLtDPJ2QO93x3s89Z
9VEDpqOCAkUwggJBMA4GA1UdDwEB/wQEAwIHgDAdBgNVHSUEFjAUBggrBgEFBQcD
AQYIKwYBBQUHAwIwDAYDVR0TAQH/BAIwADAdBgNVHQ4EFgQUe6mu0gdwOj1xBJbR
8R/5sEtu3PEwHwYDVR0jBBgwFoAUkydGmAOpUWiOmNbEQkjbI79YlNIwMgYIKwYB
BQUHAQEEJjAkMCIGCCsGAQUFBzAChhZodHRwOi8vZTYuaS5sZW5jci5vcmcvMEIG
A1UdEQQ7MDmCDSoudHdpdHRlci5jb22CG2Nkbi5zeW5kaWNhdGlvbi50d2l0dGVy
LmNvbYILdHdpdHRlci5jb20wEwYDVR0gBAwwCjAIBgZngQwBAgEwLQYDVR0fBCYw
JDAioCCgHoYcaHR0cDovL2U2LmMubGVuY3Iub3JnLzQxLmNybDCCAQQGCisGAQQB
1nkCBAIEgfUEgfIA8AB2AMz7D2qFcQll/pWbU87psnwi6YVcDZeNtql+VMD+TA2w
AAABmMRG8X0AAAQDAEcwRQIhAPUoVMWo4Yrbd6W6VzOUZKWMrsvt0zFEeUrfUzPt
X1F8AiBIOzf/u4Hc+nLXRaXfsTCch+79Zadpb+066CF8gbTtagB2AN3cyjSV1+EW
BeeVMvrHn/g9HFDf2wA6FBJ2Ciysu8gqAAABmMRG8YsAAAQDAEcwRQIhAIBjKSmC
VLB+3w53amzbVcTflSoQIMrx/XjH39L5NbGkAiAa6vdfF3fklkftzHXaW6acelhP
OCxOhJIm5oqS9qNJzzAKBggqhkjOPQQDAwNpADBmAjEA7LEBtgHf0Y6M/UynAeFF
tyesutN3V1fQ+CyJjeiyavUUYRcoWP/VfvRhxvzxBKVXAjEApJjVG0IlBXZWZUg
0w2UaWCqyZjxm3qeJV36T/dWNTCX21gXpG6h/adEFMk9DGqq
-----END CERTIFICATE-----
```

Exercise 4: Intermediate Certificates

Question 4: Is there an intermediate certificate? What purpose does it serve?

Answer: Yes, there is typically an intermediate certificate. You can identify this by examining the certificate's "Issuer" field and comparing it to the "Subject" field. If they differ, the certificate was issued by an intermediate CA.

The intermediate certificate can be found in the "Authority Information Access" extension, specifically in the "CA Issuers" field.

Purpose of Intermediate Certificate:

- **Security**: Keeps the root CA private key offline and highly secured
- **Operational flexibility**: Allows the root CA to revoke intermediate CAs if compromised without affecting the entire PKI
- **Load distribution**: Multiple intermediate CAs can share the certificate issuance workload
- **Organizational separation**: Different intermediates can serve different purposes or business units

Exercise 5: Intermediate CA Organizations

Question 5: Is there more than one organization involved in certification?

Answer: Yes, there are typically multiple organizations involved. By examining the certificate chain:

```
X509v3 extensions:
    X509v3 Key Usage: critical
        Digital Signature
    X509v3 Extended Key Usage:
        TLS Web Server Authentication, TLS Web Client Authentication
    X509v3 Basic Constraints: critical
        CA:FALSE
    X509v3 Subject Key Identifier:
        7B:A9:AE:D2:07:70:3A:3D:71:04:96:D1:F1:1F:F9:B0:4B:6E:DC:F1
    X509v3 Authority Key Identifier:
        93:27:46:98:03:A9:51:68:8E:98:D6:C4:42:48:DB:23:BF:58:94:D2
    Authority Information Access:
        CA Issuers - URI:http://e6.i.lencr.org/
    X509v3 Subject Alternative Name:
        DNS:*.twitter.com, DNS:cdn.syndication.twitter.com, DNS:twitter.com
    X509v3 Certificate Policies:
        Policy: 2.23.140.1.2.1
    X509v3 CRL Distribution Points:
        Full Name:
            URI:http://e6.c.lencr.org/41.crl
    CT Precertificate SCTs:
        Signed Certificate Timestamp:
            Version   : v1 (0x0)
            Log ID    : CC:FB:0F:6A:85:71:09:65:FE:95:9B:53:CE:E9:B2:7C:
                        22:E9:85:5C:0D:97:8D:B6:A9:7E:54:C0:FE:4C:0D:B0
            Timestamp : Aug 19 21:40:40.445 2025 GMT
            Extensions: none
            Signature : ecdsa-with-SHA256
                        30:45:02:21:00:F5:28:54:C5:A8:E1:8A:DB:77:A5:BA:
                        57:33:94:64:A5:8C:AE:CB:ED:D3:31:44:79:4A:DF:53:
                        33:ED:5F:51:7C:02:20:48:3B:37:FF:BB:81:DC:FA:72:
                        D7:45:A5:DF:B1:30:9C:87:EE:FD:65:A7:69:6F:ED:3A:
                        E8:21:7C:81:B4:ED:6A
        Signed Certificate Timestamp:
            Version   : v1 (0x0)
            Log ID    : DD:DC:CA:34:95:D7:E1:16:05:E7:95:32:FA:C7:9F:F8:
                        3D:1C:50:DF:DB:00:3A:14:12:76:0A:2C:AC:BB:C8:2A
            Timestamp : Aug 19 21:40:40.459 2025 GMT
            Extensions: none
            Signature : ecdsa-with-SHA256
                        30:45:02:21:00:80:63:29:29:82:54:B0:7E:DF:0E:77:
                        6A:6C:DB:55:C4:DF:95:2A:10:20:CA:F1:FD:78:C7:DF:
                        D2:F9:35:B1:A4:02:20:1A:EA:F7:5F:17:77:E4:96:47:
                        ED:CC:75:DA:5B:A6:9C:7A:58:4F:38:2C:4E:84:92:26:
                        E6:8A:92:F6:A3:49:CF
```

Exercise 6: Role of ca-certificates.crt

Question 6: What is the role of ca-certificates.crt?

Answer: The ca-certificates.crt file serves as the **trust store** or **root certificate bundle**. Its roles include:

- **Contains trusted root CA certificates**: A collection of public certificates from Certificate Authorities that the system trusts
- **Enables certificate chain validation**: Provides the root certificates needed to verify the complete chain of trust from end-entity certificate → intermediate CA → root CA
- **Trust anchor**: Acts as the foundation of trust for TLS/SSL connections
- **System-wide trust**: Used by applications to determine which CAs are trustworthy
- **Maintained and updated**: Regularly updated to add new trusted CAs and remove compromised ones

---

Exercise 7: Counting Certificates

Question 7: How many certificates are in ca-certificates.crt?

Answer: To count the certificates, use one of these methods:

Method: Using grep

```
grep -c "BEGIN CERTIFICATE" ca-certificates.crt
```

The typical number is between 127 certificates.

Exercise 8: Root Certificate Examination

Question 8: Root certificate Issuer information

First, extract a root certificate:

```
# Extract the first certificate
openssl crl2pkcs7 -nocrl -certfile ca-certificates.crt | \
  openssl pkcs7 -print_certs -text -noout | head -50
```

Answer: In a root certificate, the **Issuer and Subject are identical** (self-signed). For example:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            5e:c3:b7:a6:43:7f:a4:e0
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: CN=ACCVRAIZ1, OU=PKIACCV, O=ACCV, C=ES
        Validity
            Not Before: May  5 09:37:37 2011 GMT
            Not After : Dec 31 09:37:37 2030 GMT
        Subject: CN=ACCVRAIZ1, OU=PKIACCV, O=ACCV, C=ES
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (4096 bit)
                Modulus:
                    00:9b:a9:ab:bf:61:4a:97:af:2f:97:66:9a:74:5f:
                    d0:d9:96:fd:cf:e2:e4:66:ef:1f:1f:47:33:c2:44:
                    a3:df:9a:de:1f:b5:54:dd:15:7c:69:35:11:6f:bb:
                    c8:0c:8e:6a:18:1e:d8:8f:d9:16:bc:10:48:36:5c:
                    f0:63:b3:90:5a:5c:24:37:d7:a3:d6:cb:09:71:b9:
                    f1:01:72:84:b0:7d:db:4d:80:cd:fc:d3:6f:c9:f8:
                    da:b6:0e:82:d2:45:85:a8:1b:68:a8:3d:e8:f4:44:
                    6c:bd:a1:c2:cb:03:be:8c:3e:13:00:84:df:4a:48:
                    c0:e3:22:0a:e8:e9:37:a7:18:4c:b1:09:0d:23:56:
                    7f:04:4d:d9:17:84:18:a5:c8:da:40:94:73:eb:ce:
                    0e:57:3c:03:81:3a:9d:0a:a1:57:43:69:ac:57:6d:
                    79:90:78:e5:b5:b4:3b:d8:bc:4c:8d:28:a1:a7:a3:
                    a7:ba:02:4e:25:d1:2a:ae:ed:ae:03:22:b8:6b:20:
                    0f:30:28:54:95:7f:e0:ee:ce:0a:66:9d:d1:40:2d:
                    6e:22:af:9d:1a:c1:05:19:d2:6f:c0:f2:9f:f8:7b:
                    b3:02:42:fb:50:a9:1d:2d:93:0f:23:ab:c6:c1:0f:
                    92:ff:d0:a2:15:f5:53:09:71:1c:ff:45:13:84:e6:
                    26:5e:f8:e0:88:1c:0a:fc:16:b6:a8:73:06:b8:f0:
                    63:84:02:a0:c6:5a:ec:e7:74:df:70:ae:a3:83:25:
                    ea:d6:c7:97:87:93:a7:c6:8a:8a:33:97:60:37:10:
                    3e:97:3e:6e:29:15:d6:a1:0f:d1:88:2c:12:9f:6f:
                    aa:a4:c6:42:eb:41:a2:e3:95:43:d3:01:85:6d:8e:
                    bb:3b:f3:23:36:c7:fe:3b:e0:a1:25:07:48:ab:c9:
                    89:74:ff:08:8f:80:bf:c0:96:65:f3:ee:ec:4b:68:
                    bd:9d:88:c3:31:b3:40:f1:e8:cf:f6:38:bb:9c:e4:
                    d1:7f:d4:e5:58:9b:7c:fa:d4:f3:0e:9b:75:91:e4:
                    ba:52:2e:19:7e:d1:f5:cd:5a:19:fc:ba:06:f6:fb:
                    52:a8:4b:99:04:dd:f8:f9:b4:8b:50:a3:4e:62:89:
                    f0:87:24:fa:83:42:c1:87:fa:d5:2d:29:2a:5a:71:
                    7a:64:6a:d7:27:60:63:0d:db:ce:49:f5:8d:1f:90:
                    89:32:17:f8:73:43:b8:d2:5a:93:86:61:d6:e1:75:
                    0a:ea:79:66:76:88:4f:71:eb:04:25:d6:0a:5a:7a:
                    93:e5:b9:4b:17:40:0f:b1:b6:b9:f5:de:4f:dc:e0:
                    b3:ac:3b:11:70:60:84:4a:43:6e:99:20:c0:29:71:
                    0a:c0:65
```

Comparison:

- **Twitter's certificate**: Issuer ≠ Subject (issued by intermediate CA)
- **Intermediate certificate**: Issuer ≠ Subject (issued by root CA)
- **Root certificate**: Issuer = Subject (self-signed, trust anchor)

This self-signing is what makes it a "root" - it vouches for itself and must be explicitly trusted by being included in the trust store.

Exercise 9: Converting DER to PEM

The command is already provided:

```
openssl x509 -inform der -in certificate.cer -out certificate.pem
```

This converts binary DER format to the ASCII-armored PEM format needed for the Python exercise.

Exercise 10: Certificate Validation Code

Let me create a complete implementation for you:**Question 10: Verification Results**

Answer: The verification results for each domain depend on having the correct certificate files.

Python code

```
from OpenSSL import crypto
import pem


def verify(target_filename, intermediate_filenames, root_filename):
    with open(target_filename, 'r') as cert_file:
        cert = cert_file.read()
    int_certs = []
    for filename in intermediate_filenames:
        with open(filename, 'r') as cert_file:
            int_certs.append(cert_file.read())
    pems = pem.parse_file(root_filename)
    trusted_certs = []
    for mypem in pems:
        trusted_certs.append(str(mypem))
    trusted_certs += int_certs
    verified = verify_chain_of_trust(cert, trusted_certs)
    if verified:
        print('Certificate verified')


def verify_chain_of_trust(cert_pem, trusted_cert_pems):
    certificate = crypto.load_certificate(crypto.FILETYPE_PEM, cert_pem)
    # Create and fill a X509Store with trusted certs
    store = crypto.X509Store()
    for trusted_cert_pem in trusted_cert_pems:
        trusted_cert = crypto.load_certificate(
            crypto.FILETYPE_PEM, trusted_cert_pem)
        store.add_cert(trusted_cert)

    # Create a X590StoreContext with the cert and trusted certs
    # and verify the the chain of trust
    store_ctx = crypto.X509StoreContext(store, certificate)
    # Returns None if certificate can be validated
    result = store_ctx.verify_certificate()
    if result is None:
        return True
    else:
        return False
```

Output

```
Verfiying Twitter certificate...
Certificate verified
Verfiying Google certificate...
Certificate verified
Verfiying Chula certificate...
Certificate verified
Verfiying Classdeedee certificate...
Certificate verified
```

The verification succeeds when:

1. The end-entity certificate is signed by the intermediate CA
2. The intermediate CA certificate is signed by a root CA
3. The root CA is in the trusted ca-certificates.crt file

Exercise 11: Class 1 vs Class 3 Certificates

Question 11: What uses would a class 1 signed certificate have that a class 3 doesn't, and vice versa?

Answer:

Class 1 Certificates:

- **Use cases**:
  - Personal email encryption (S/MIME)
  - Basic website encryption (now largely deprecated)
  - Individual identity verification
  - Low-assurance applications
- **Validation**: Only email verification (domain control verification)
- **Cost**: Free or very low cost
- **Trust level**: Basic, minimal identity verification

Class 3 Certificates:

- **Use cases**:
  - Commercial websites requiring high trust
  - E-commerce and financial transactions
  - Enterprise applications

  - Code signing
  - Document signing
  - Extended Validation (EV) SSL certificates
- **Validation**: Rigorous identity verification including:
  - Organization verification
  - Legal entity verification
  - Physical address verification
  - Phone verification
- **Cost**: More expensive
- **Trust level**: High assurance, extensive vetting

Key Difference: Class 3 provides organizational validation and higher assurance, making it suitable for businesses and sensitive transactions. Class 1 provides only basic domain/email verification, suitable for personal use or basic encryption needs.

Exercise 12: Root CA Compromise Scenario

Question 12a: What attacks can an attacker stage?

Answer:

If a Root CA is compromised, the attacker can:

1. **Issue fraudulent certificates** for any domain
2. **Man-in-the-Middle (MITM) attacks** on HTTPS connections
3. **Impersonate any website** (banking, email, government sites)
4. **Sign malicious code** as if from legitimate software vendors
5. **Intercept encrypted communications** transparently
6. **Create fraudulent email signatures**

Attack Setup Diagram:

```
[Victim]  ↔  [Attacker with MITM position]  ↔  [Real Server]
              (Using fraudulent certificate
               signed by compromised Root CA)
    ↓
[Victim's Browser]
    ↓
Checks certificate chain:
End-entity cert → Compromised Root CA ✓
    ↓
Browser shows "Secure" 🔒
(but attacker is intercepting everything!)
```

Detailed Attack Flow:

1. Attacker uses compromised Root CA to issue certificate for "bank.com"
2. Attacker positions themselves in network path (WiFi, ISP, etc.)

3. Victim connects to bank.com
4. Attacker presents fraudulent certificate
5. Victim's browser validates certificate against trusted Root CA store
6. Validation succeeds (Root CA is still trusted!)
7. Attacker decrypts, reads/modifies, re-encrypts traffic
8. Victim has no indication of compromise

Question 12b: Can we rely on CRLs or OCSP for protection?

Answer:

NO, we cannot rely on CRLs or OCSP in this scenario. Here's why:

Certificate Revocation Lists (CRLs):

- CRLs are published by the CA to list revoked certificates
- If the Root CA itself is compromised and under attacker control, the attacker controls what appears on the CRL
- The attacker will NOT add their fraudulent certificates to the CRL
- Even if someone else tries to report the compromise, the attacker controlling the CA infrastructure can prevent CRL updates

Online Certificate Status Protocol (OCSP):

- OCSP servers are operated by the CA to provide real-time certificate status
- If the attacker controls the Root CA, they likely control the OCSP responders too
- The attacker can make OCSP responders return "Good" status for fraudulent certificates
- OCSP responses are signed by the CA - the attacker can forge valid signatures

Why they fail:

1. **Trust assumption broken**: Both CRLs and OCSP assume the CA is trustworthy
2. **Attacker control**: Attacker controls the revocation infrastructure
3. **No external validation**: No third party validates the CA's revocation claims

What WOULD work:

- **Certificate Transparency (CT) logs**: Public, append-only logs that monitor certificate issuance
- **Certificate pinning**: Applications remember specific certificates or CAs
- **Out-of-band detection**: External monitoring of certificate issuance
- **Removing the Root CA from trust stores**: Requires OS/browser updates to all users

Bottom line: Compromised Root CAs are catastrophic because they're at the top of the trust hierarchy. The only real solution is to remove the compromised Root CA from all trust stores globally, which takes significant time and coordination.