TLS Assignment

Due: April 4, 2022 @9PM Eastern via Gradescope

Team Size: 2 Students Per Team (You may submit either a single submission or independent submissions)

Assignment Overview

In this assignment you are going to be flexing your TLS muscles.

Resources

Cloudflare Blog

- Detailed Look at TLS 1.3

TLS Documentation

- TLS v1.2 RFC
- TLS v1.3 RFC

Tasks

Review Questions (20 pts.)

1. Before you can run a MITM attack on a protocol running between two hosts, you must be in the position to intercept communication between the hosts at the level where the protocol messages are being sent and received. The first step for doing this is to intercept network packets between two hosts. Name three ways for an attacker to intercept network traffic that we have learned about in class (each way should be exploiting a different protocol.)

2. How does TLS reduce the harms an adversary can induce by intercepting network traffic?

3. Consider a flawed TLS implementation where the client "forgets" to check the signature on the server's certificate. Write down exactly how a man-in-the-middle attacker that intercepts the communications between client and server can establish one key k between itself and the client, and another key k' between itself and the server when the client tries to connect to the server. Explain why, by doing this, the attacker can silently intercept, read, and pass on any data sent from client to server, and vice versa, without the client or server ever realizing that their communications have been read.

Tinkering With TLS (40 pts.)

In this set of questions, we are going to see what happened to the security of TLS1.2 when various parts of the protocol are changed. In order to do that, we need to fix some notation. Here is the full version of TLS:

- Client → Server: ClientHello = (Version, CipherSuiteList, r_c)
- Client ← Server: ServerHello = (Version, CipherSuite, r_s)
- Client ← Server: Certificate = ($cert$)
- Client ← Server:
  - case KeyEncapsulation: ServerKeyExchange = ()
  - case Static DH: ServerKeyExchange = ()
  - case Ephemeral DH: ServerKeyExchange = ($g^b, \sigma = Sign(pk_{sign,server},\ r_c \mid\mid r_s \mid\mid g^b)$)
- Client ← Server: ServerHelloDone = ()
- Client → Server: ClientKeyExchange:
  - case KeyEncapsulation: ServerKeyExchange = ($c = Enc(pk_{enc,server}, pms)$)
  - case Static DH: ServerKeyExchange = (g^a)
  - case Ephemeral DH: ServerKeyExchange = (g^a)
- Client and Server set $pms = g^{ab}$ if in a DH mode
- Client and Server set $msk = KDF(pms$, master-secret-label, $r_c + r_s)$
- Client → Server: Finished = (Enc(msk, client-finished-label || H(handshake) || MAC(msk, client-finished-label || H(handshake))))
- Client ← Server: Finished = (Enc(msk, server-finished-label || H(handshake) || MAC(msk, server-finished-label || H(handshake))))

1. Imagine a version of TLS without r_c. Does there exist a replay attack that can be launched against this protocol? If so, what is the attack and what are its implications? If not, why not? The protocol in question is written out explicitly below:

- Client → Server: ClientHello = (Version, CipherSuiteList)
- Client ← Server: ServerHello = (Version, CipherSuite, r_s)
- Client ← Server: Certificate = ($cert$)
- Client ← Server:
  - case KeyEncapsulation: ServerKeyExchange = ()
  - case Static DH: ServerKeyExchange = ()
  - case Ephemeral DH: ServerKeyExchange = ($g^b, \sigma = Sign(pk_{sign,server},\ r_s \mid\mid g^b)$)
- Client ← Server: ServerHelloDone = ()
- Client → Server: ClientKeyExchange:
  - case KeyEncapsulation: ServerKeyExchange = ($c = Enc(pk_{enc,server}, pms)$)
  - case Static DH: ServerKeyExchange = (g^a)
  - case Ephemeral DH: ServerKeyExchange = (g^a)
- Client and Server set $pms = g^{ab}$ if in a DH mode
- Client and Server set $msk = KDF(pms$, master-secret-label, $r_s)$
- Client → Server: Finished = (Enc(msk, client-finished-label || H(handshake) || MAC(msk, client-finished-label || H(handshake))))
- Client ← Server: Finished = (Enc(msk, server-finished-label ||

H(handshake) || MAC(msk, server-finished-label || H(handshake))))

2. Imagine a version of TLS without r_s. Does there exist a replay attack that can be launched against this protocol? If so, what is the attack and what are its implications? If not, why not? The protocol in question is written out explicitly below:

- Client → Server: ClientHello = (Version, CipherSuiteList, r_c)
- Client ← Server: ServerHello = (Version, CipherSuite)
- Client ← Server: Certificate = ($cert$)
- Client ← Server:
  - case KeyEncapsulation: ServerKeyExchange = ()
  - case Static DH: ServerKeyExchange = ()
  - case Ephemeral DH: ServerKeyExchange = ($g^b, \sigma = Sign(pk_{sign,server},\ r_c \mid\mid g^b)$)
- Client ← Server: ServerHelloDone = ()
- Client → Server: ClientKeyExchange:
  - case KeyEncapsulation: ServerKeyExchange = ($c = Enc(pk_{enc,server}, pms)$)
  - case Static DH: ServerKeyExchange = (g^a)
  - case Ephemeral DH: ServerKeyExchange = (g^a)
- Client and Server set $pms = g^{ab}$ if in a DH mode
- Client and Server set $msk = KDF(pms$, master-secret-label, $r_c)$
- Client → Server: Finished = (Enc(msk, client-finished-label || H(handshake) || MAC(msk, client-finished-label || H(handshake))))
- Client ← Server: Finished = (Enc(msk, server-finished-label || H(handshake) || MAC(msk, server-finished-label || H(handshake))))

3. Imagine a version of TLS without the H of the handshake message. Can the attacker reduce the security of the TLS protocol to the weakest ciphers that are supported by the client and the server? what is the attack and what are its implications? If not, why not? The protocol in question is written out explicitly below:

- Client → Server: ClientHello = (Version, CipherSuiteList, r_c)
- Client ← Server: ServerHello = (Version, CipherSuite, r_s)
- Client ← Server: Certificate = ($cert$)
- Client ← Server:
  - case KeyEncapsulation: ServerKeyExchange = ()
  - case Static DH: ServerKeyExchange = ()
  - case Ephemeral DH: ServerKeyExchange = ($g^b, \sigma = Sign(pk_{sign,server},\ r_c \mid\mid r_s \mid\mid g^b)$)
- Client ← Server: ServerHelloDone = ()
- Client → Server: ClientKeyExchange:
  - case KeyEncapsulation: ServerKeyExchange = ($c = Enc(pk_{enc,server}, pms)$)
  - case Static DH: ServerKeyExchange = (g^a)
  - case Ephemeral DH: ServerKeyExchange = (g^a)
- Client and Server set $pms = g^{ab}$ if in a DH mode
- Client and Server set $msk = KDF(pms$, master-secret-label, $r_c + r_s)$
- Client → Server: Finished = (Enc(msk, client-finished-label || MAC(msk, client-finished-label)))

- Client ← Server: Finished = (Enc(msk, server-finished-label || MAC(msk, server-finished-label)))

In showing your attacks, *be sure to give specific instructions.* In particular, your answers should be specific enough that it would be easy to write code that implements your attack. What messages would you send? What would the contents of those messages be? If you are using messages that were sent during a previous interaction, make sure you are clear what values you are reusing. After the attack, what power does the attacker have and what values that *should* be secret do they know?

Tricking your Browser (40 pts.)

In order to get a better feel for how certificates and certificate signing works, you are going to go through the process of generating a certificate. Very helpful instructions for this process can be found here, but you are free to look wherever — there are lots of good guides on the internet. Your goal is to trick your browser into connect over TLS (ie. verifies the certificate chain) to a domain that you don't own by adding the CS558 certificate to your set of trusted certificates. This simulates the attack that someone could launch if they managed to steal a CA's signing key.

You are going to turn in a diary of the process. Be sure to include all the commands you ran on the command line in your diary. Let us know what tools you used for the server and how you got the proxy working. Your submission should be a step-by-step guide for someone in the future (maybe your future self) to do this activity! Finally, be sure to include the screenshot of your browser connecting to your local server.

In order to do this, I am going to give you the information that a CA would hold in order to actually do the signing. First, the CA's certificate:

```
-----BEGIN CERTIFICATE-----
MIIGBzCCA++gAwIBAgIJAK0Ks8oTI57OMA0GCSqGSIb3DQEBCwUAMIGZMQswCQYD
VQQGEwJVUzELMAkGA1UECAwCTUExDzANBgNVBAcMBkJvc3RvbjEeMBwGA1UECgwV
Qm9zdG9uVW5pdmVyc2l0eUNTNTU4MQ4wDAYDVQQLDAVDUzU1ODEVMBMGA1UEAwwM
Y3M1NTguYnUuZWR1MSUwIwYJKoZIhvcNAQkBFhZ1bnNhdm9yeWNhZmZpbmVAYnUu
ZWR1MB4XDTIyMDMyODAyNDc0MVoXDTIyMDQyNzAyNDc0MVowgZkxCzAJBgNVBAYT
AlVTMQswCQYDVQQIDAJNQTEPMA0GA1UEBwwGQm9zdG9uMR4wHAYDVQQKDBVCb3N0
b25Vbml2ZXJzaXR5Q1M1NTgxDjAMBgNVBAsMBUNTNTU4MRUwEwYDVQQDDAxjczU1
OC5idS5lZHUxJTAjBgkqhkiG9w0BCQEWFnVuc2F2b3J5Y2FmZmluZUBidS5lZHUw
ggIiMA0GCSqGSIb3DQEBAQUAA4ICDwAwggIKAoICAQDdqTfYQuKpG//tJDWvUecM
pV+d6kgy5DifU4qOHHQgwn0yNm2d/ZQDg/8zwtL1x8CgoJuXcY1WN+TNqMIMhrKb
i77wUtGBq7lQINoQ9zC8xqxCw7O4EG+tKStrltDMxkN0IHGutK5RUnUbuWaCE/jg
S7hUPUPpT1W1lvKsMuYYZBPlgmmGUXmGmFgrcxCd0lnGZ0srroF2244CD2A2DqdK
b5tqVR1EUFnLhMhWnYZHYFz9fFW6R6Rvqsgw8ch9Vm/VD9TdlxNXubKTiFbd7An9
Kbv3dlzFstb64iiBJ3aaNRN80YrW3bhfDxtfIvybPcfktv0pMWFnWk0nRexyCdka
sR6D45ypfqfmLflUkc3O4No3uOBVIUMPMo7FYZWR3YZM90FdYkCudr3YwQcEiRHc
```

```
xC7mA44Okop0e/KU7razQUswHBah81VqELqzUXR0fEw/+8QgY//4VQh8eHxcCaWS
9neExkHAkXX70nk+L2f4De4ZDSxf01yHsBFNHO9n9dw9Hvy+SniRV/AFwgva3qpM
DVJs/Yiyhidv2Ec6gfCGQG3w0eySw0aChk5It4EAsdVuCnKGJyHytKVQ3mdfVc81
RhmXjYy6A4TyGt9nTXzv98PEzdVIV3DPLL6ZhFxlYpy2Wae9G2kWiljEezGPf9oN
BPjwZ0444i3Muwvj1mURQwIDAQABo1AwTjAdBgNVHQ4EFgQUF0itt9HF+n507EAO
Oqht2c64fu8wHwYDVR0jBBgwFoAUF0itt9HF+n507EAOOqht2c64fu8wDAYDVR0T
BAUwAwEB/zANBgkqhkiG9w0BAQsFAAOCAgEAUuIXTjZTc8PhcmAXHUq6sEpz9ZN3
F1YUfPpsWF8HKEiZ0d1Q5R6QZO3bR4loqWmtQNqbldD8AvDovvWqbfck0Ud+pM04
BIWzL7xhf2NfGlUjl5JwHETL9h+Lv0GxqcC3fQJsaF3JtEPgy8ZD+ObzdvFn0mQ2
cPKNbDwDUJih6yz4VSkrllTeMQ6GHEOY6YYgj1rNNjKr3F3r9E3U5vLZov0bhx/k
SuExYy19mxV7gR+Gs/A0+DZSzjRYRL9PhVBHPCXM0oeGAWmYWicuui7YsZgYR0xa
GKVvemTsAZUUOWnANd8rMVqpY/SI4vl82CXg7D/b0oig++J/3PD3STZl+rvQ4dw6
G2JV22mf5WSiMJNKifl5IfJRSSarhKSZAb3san7HNU5QVRvIYrQGU3YL9O96NVcn
85tTHOTpmgZwCTAL2hrLng6Jf6EqAk9Cvn02TI1vWfHwkRkAnzvXkQs6A6gH5qBx
8sK6gMSQTW19QBGICkcAFcTKGcz1oBYc2x4OJI6ZDL9ZU1Y8Y/De0SGpA1ZzYUmE
dbb9vo6912jqE+sNoDY5n4g/gYW+ANm11HaRjSzj0HTbgtWmdNnc6R1ftN9kB+cE
FpXuSTDojM5PIksNWIHuGQimnADSPg0QMSYCx5gMqsKnJHzDPgPrYId5NsqR9+HM
XRuPJCIAbUN0cnw=
-----END CERTIFICATE-----
```

Note that I'm ok with you making a new certificate if you would like. As long as the top level certificate in the chain you create is a self signed certificate under the following private key:

```
-----BEGIN RSA PRIVATE KEY-----
MIIJKQIBAAKCAgEA3ak32ELiqRv/7SQ1r1HnDKVfnepIMuQ4n1OKjhx0IMJ9MjZt
nf2UA4P/M8LS9cfAoKCbl3GNVjfkzajCDIaym4u+8FLRgau5UCDaEPcwvMasQsOz
uBBvrSkra5bQzMZDTiBxrrSuUVJ1G7lmghP44Eu4VD1D6U9VtZbyrDLmGGQT5YJp
hlF5hphYK3MQndJZxmdLK66Bdtu0Ag9gNg6nSm+balUdRFBZy4TIVp2GR2Bc/XxV
ukekb6rIMPHIfVZv1Q/U3ZcTV7myk4hW3ewJ/Sm793ZcxbLW+uIogSd2mjUTfNGK
1t24Xw8bXyL8mz3H5Lb9KTFhZ1pNJ0XscgnZGrEeg+OcqX6n5i35VJHNzuDaN7jg
VSFDDzKOxWGVkd2GTPdBXWJArna92MEHBIkR3MQu5gOODpKKTnvyl062s0FLMBwW
ofNVahC6s1F0dHxMP/vEIGP/+FUIfHh8XAmlkvZ3hMZBwJF1+zp5Pi9n+A3uGQ0s
Xztch7ARTRzvZ/XcPR78vkp4kVfwBcIL2t6qTA1SbP2IsoYnb9hHOoHwhkBt8Dns
ksDmgoZOSLeBALHVbgpyhich8rSlUN5nX1XPNUYZl42MugOE8hrfZ0187/fDxM3V
SFdwzyy+mYRcZWKctlmnvRtpFopYxHsxj3/aDQT48GdOOOItzLsL49ZlEUMCAwEA
AQKCAgBlhxafJbOwBbUpp4Y3cWpE7pJnQGIlfUc6Iwe5o+rE/pBdqXR4AygCnDkO
OlRqYz4l1KqvqUE1lpBkasHG/wNcH5wrc6Omo0NUIlf/oVlffhh01DLDQjQEunC6
7O9ifAVkCZRIk1WsxfoB4t/DAObjxYr+erlaag42CJfKq92cmmpKm3s+HJ9vOORZ
snCP+UNJjxJtRZbjHBlldCl7WSbi/0/OWoH3Ql5+y6j/k1Nn6gltyb9yfVIiG7Vq
RbSxRCAhFQlJHeOsMNBMpwwyxeSlYrJH3J0NqKazb1diIPNAGsN8TnYriI7ka4T8
BIhzis6+QdqfPZEBx+jC7lIowb4Ak0/T0v3HmIWxpFvrOUvO3IToTP5x2E0l9Spm
cGbgNYNVPjvTQozSFwSYrlPsLQwMv1GZpZpmdDPR65rVUtuDDdu2ubMrniUJCUhP
8VGMSd48tfuhXh8TVCKsQ28LI3Tyi8KmFPoPvFR7O6huq9vaStGp41+90GJILu0+
mOpMkzOWH/r9AhZ1RU2DtdfnlQ3/C5xgzasjgKMxSLV0H1cVunF5eWZumdOhU+ob
e+j7gEeRaFz6X+rk044zoYf8rhB2G6LT2IJV6P5XAdRTQBz1B9/UwrhQ3aoPBqsM
QQw3bUjUtCTapwpxPZ8IfVoRcMFzxA/VV0QqBIzoClJ8zjYbQQKCAQEA+u+GzWv5
```

```
7l9WG4PYc+P95OHsywDREh4u7LhNjGXzxU4UiqRkl6VoZebzk7yGq8uX+itiRNFu
Z71TFJp2uzg0L+CAm5HwjTs8UYloi0JNALuinXHzpS3+sp8W30XH0yaTqYqa3yTx
XxKFjXc/5jCmFOVpCwhEJ1Uur48mbiS7TMVpOzQEEASTskxlY9Rt0MZWEkWwutnw
NYqyEsFQ0wVzGp16UVhCvMaANHI915STB80Ys2xN3GF/B/dwEgKErO2qrAEz4F4Y
KHJ+JgebfF+weKOMkyVkBOmJ719/CoD/hLC5IZC6++WfD8gBiwoxV2pC9H3hWeeI
I7HMyyaxok1HUwKCAQEA4iJxRKNiovMBnceUvGj0IoCZyCwKeXuA4/thr8l+2ApH
7qjL2DMn4JS4BUnbNLiMbngJM45RI7ys0ANbTMgc2Gy11cAOH5o6c4BsaaKPOZGv
MMcEUuIh3DdIG2TQZpgwiU/kvOWKWo33OLFUo+aS4kOTziOeOV1KSf4Nl2ZRmbhO
gx2km9DVnhxPIjt5J/ktn0HHAXE5/Bd65OaQhcSRvpzU2q3c84KV3J5SBFXGIAlx
14HSUi10Awb0KOTt2QWHbbk4COSe7YIJP/tzsxfZM7qkpvx6kpn+F1XEBCEnmR0u
E9ToZyQnKNQmhv8BakRTOCPsLfuvOE+C9EHapZiAUQKCAQAUlJaKvINYEIugYBTd
lGJbZkgkciGzibQxiAAcNrRihz/aCxeQ9Gj2ipWJlVm6N010oCBEIqUabwWkV/LK
8hYdox0JJVQEUUpmKY3gdgSYvcrdfN4NuxL7lV6r2y5DXBOaQRromEAPmxZG/vPH
rk/AwPv3gqsMSsk0btopCGwwJLS3vVFj+uweIHPkVyTWjZ27i+mtuXgg/AoUzbQS
So0hLq88gq+eie0z3/bAepgAeMrA1G4iWACyJ5ISeBBnmmp4BvU5Pp5emt1Lwy32
amavzkIWQ2fLm1fLwRpLQz8xo8jbPuKHDFMaWT3/KEvZroZlRPm5hOq+erOgKcFB
3XKvAoIBAQDdcgEdyColjHb+vZ1HzDeXOfxea9JuGKWlnFyTOmy+v4KlkiLcu2vH
n5t9gk/plvfejinklO+cYX2RzlewHx8wSXTft02dYPjwdsizwX8kTygSSjJPwCaM
co5oVRdIAK03KkfDO7165B/T/HP4dSlN7gNmELc3UcYYI3PH2Wj7ceNgvryd4anv
RaWwzjDdFkS4+j8ZiHnSBmRQmADbHh3jXc2LwErpI+4BuAB1QlHcuaMD+Zuu4dgD
xuF+0oCgz6tJpeHbw5Zm27qXL3Sj4yzOXW40IHcf7TFIRLLJoHYYmNywiwRzTJIU
h3ybIkm0eQ5Noc/9T8TNDgAdlge5tlehAoIBAQDx8UG6WgFXkw53w9w7WkjeeAnq
loZTrOdXjabxlYhluuxhluSO3plM1fDKizQP26VFR8EvaVBZb/X9ej8c+hwI/pSl
7fFjOLWYkX3WkG8qANUutSHjJPXS6q+y2gKEm7g6WnnzIaFR8wAHuBSmY23ZhCy/
vBTlOS/eKZ1cs+2it5HmaPzUJbQZ7dtp37gCnay02g5+MmvUTuSLZVAaprvPDv7L
VhOfnlrX7YpLwcsBNmhHsRVA74kvlIlX36XjsqnIZWKW5P1aZ8xEofPAOGf2OkP7
bwWtIBOp/wx65SQ/HWBxOf3A9F/BNMNKT4oVgPq9RQDlaYrOnF11e/pQ6VXx
-----END RSA PRIVATE KEY-----
```

NOTE: THIS KEY IS NOW OUT IN THE WILD. MAKE SURE YOU DO NOT KEEP TRUST IN THIS KEY AFTER DOING THE ASSIGNMENT!!!!

1. Make sure you can see whats going on in this certificate and key. Generate the public key corresponding to this private key. Looking at the certificate, what is the contact email for the CA?

2. Choose a domain you want to impersonate. Generate a certificate signing request for that domain.

3. Sign this CSR using the cs558key to generate a certificate for the domain you want to impersonate.

4. Using `openssl verify` check that your certificate is valid with respect to the cs558 CA certificate

5. Set up a local TLS server of some kind. There are many choices. The guide above uses bud. You can also use the docker container you built for the heartbleed assignment and just replace the default server's keys and certificates. A third option is to the the `openssl s_server` tool, which

has tons of options and some non-trivial amount of documentation. You want this server to generate TLS connections (as the server) using the certificates and server key that you have generated in the previous steps. It can serve any arbitrary html content.

6. Set up a proxy on your local machine to redirect requests (at least for your target domain) to your local webserver. Changing `/etc/hosts` will do this on some OS's. On macOS, you can also change this in system preferences. Windows should have a similar proxy tool in the preferences (sorry, your professor knows nothing about Windows. If this is a problem, let me know and I will do some investigation).

7. Add the cs558 certificate to the appropriate keystore (either the system keystore or your browser's keystore).

8. Open your browser and connect to the target domain! Your connection should be redirected to your local server. Take a screenshot of your browser connected to the website with the certificate information it that it has verified expanded (eg. on Chrome, click on the lock next to the URL and select certificates. On Firefox, click on the lock, click on the right arrow, then more information, and finally view certificate).

Deliverables, Checklist

You should submit a PDF. Be sure that you submission includes:

- Answers to the review questions
- Attacks on the three protocols described above. Let us know what messages the attacker will send to the relevant parties.
- Your diary and screenshot for tricking your browser