NETWORK SECURITY HW6

GUR TELEM 206631848

1. SSL

1.1. Does comply with PFS?. The protocol does comply with PFS.

The long term password is `pass` (the one the student and the server agreed on).

And the short term password is calculated from the `client_random`, `server_random` and `pre_master_secret` (which creates the `master_secret`).

Since the DH keys are random each time, meaning that leaked DH won't leak previous `pre_master_secret` which is required to calculate the `master_secret`.

Since the short term password of one session doesn't compromise other short term password, the protocol complies with PFS.

If `pass` is leaked, it still won't help because the `pass` is only used to "trust" the agreed upon `master_secret` and it doesn't play a part in its calculation. Meaning that while it may help initiated a new session, imitating a legitimate user, it won't compromise data transferred in previous connections.

1.2. Is the protocol secured? The protocol is secured.

For an attacker to be able to pretend to be a client they need to know a pair of y and the corresponding $E_{f(pass)}(g^y \mod p)$. The former is to be able to use the DH key to send a `pre_master_secret` that is visible for the attacker, and the latter is so the server would trust the client.

Since the attacker doesn't know what $f(pass)$ is, they can't just calculate such a pair for a random y.

Unless $f(pass)$ is leaked (aka password equivalent), the attacker is pretty helpless (even offline brute-force attack wouldn't help much because there are two unknown parameters that we need to guess to be able to recognize a good pair: a pair with the latter previously used).

Thus an attacker is unable to impersonate a legitimate client.

The `pre_master_secret` isn't visible to the attacker as we previously mentioned, thus the attacker is unable to produce a `master_secret` that will be able to decrypt traffic.

1.3. Is the SDL resilient to dictionary attack by a passive attacker? Yes.

As explained in the previous section, we have two unknown parameters that we need to "guess". One of which is pseudo random and won't appear in a dictionary (hashed `pass` and y). The second is a hashed password. And even if we found a pair for which we got a previously seen $E_{f(pass)}(g^y \mod p)$, we still don't know if it's an accidental pair or if the $f(pass)$ and y are actually the same one used to calculate the forementioned value.

Meaning that since we don't have a value that is solely dependent on `pass`, we can't use a dictionary attack.

1.4. Is the SDL resilient to dictionary attack by an active attacker? The protocol is susceptible to a dictionary attack by an active attacker.

An active attacker can impersonate a server:

(1) Wait for a `ClientHello`
(2) Return a $g^x \mod p$ s.t. the x is known to him (along with `server_random` that we also need to remember)
(3) Get $E_{f(pass)}(g^y \mod p)$ and E_k(pre_master_sercert) from the client
(4) Get the `FinishedClient` message from the client

Now for the dictionary attack:

(1) For each $pass$ in the dictionary:
    (a) Calculate $f(pass)$ (since f itself isn't a secret)
    (b) Decrypt $E_{f(pass)}(g^y \mod p)$ using $f(pass)$
    (c) Calculate $g^{yx} \mod p$ using the value from the previous section and the random x we chose after getting `ClientHello`
    (d) Use $g^{yx} \mod p$ to decrypt E_k(pre_master_sercert)
    (e) Calculate `master_secret` using the `pre_master_sercert` that we calculated and the `*_randoms` that were easily visible
    (f) Try to decrypt the `FinishedClient` which is already protected by the `master_secret`.
    (g) If we got a valid message, then we found the password.

The difference here is that we could know the x and thus we had a value in hand which is solely dependent on `pass`.

1.5. **Is the modified protocol resilient to dictionary attack by an active attacker?** In this case a dictionary attack (an offline one) wouldn't work since we don't have a value to confirm our guess against.

Since the client would probably give up after a few failed attempts, we wouldn't be able to do a proper online dictionary attack as neither the client, nor the server would agree to talk to as for long enough.

2. IKE

A. Doesn't retain IKE's security
B. Retains security ; Doesn't reduce the amount of exchanged data
C. Retains security ; Reduces amount of exchanged data

2.1. **The cookie will be a single bit.** A.

Security isn't retained as one of the purposes of the cookie is to prevent DoS attacks that use IP spoofing by preventing doing the expensive operation which is calculating the exponent in the DH.

An attacker would be able to DoS attack a responder by sending a lot of messages with interchangeably 0 and 1 as cookies. The responder will ignore the incorrect cookie value but for the correct one it will continue to the next step which is the exponent for the DH. The attacker would be able to spoof his IP as he doesn't need the actual response from the responder (they simply send two packets, one for each possible cookie).

2.2. **Don't send cookie in Aggressive Mode.** C.

In aggressive mode, the server doesn't verify there isn't a DoS attack using the cookie.

Since the protocol doesn't need to handle the case of overlapping sessions, then the cookie has no purpose left for it and it can be eaten away (bad pun intended).

2.3. **Using 0 instead of sending M_{ID} in Quick Mode.** C.

The purpose of M_{ID} is to identify an activation of Quick Mode (as the IKE protocol doesn't use TCP and thus doesn't have a "session" per se). Since we don't need to support concurrent activations (no concurrent sessions), we don't need it.

Thus it doesn't hurt the security and obviously it's one less piece of data being exchanged.

2.4. **Double hashing.** B.

The length of the result of $PRF_{SKEYID}(x)$ is always the same, regardless of the x. Meaning that the amount of exchanged data isn't changed.

Since the result of a double hash is as difficult to guess as a single one (or more, since an attacker needs to hash twice for each try), it doesn't hurt the security of the protocol.

The purpose of the hashes is to prevent an attacker from replacing the algorithms specified with weaker ones. And as mentioned above, that purpose is still being held.

3. Wireless

3.1. **BREAP.**

a. How does each side verify the identity of the other?

The AS verifies the supplicant's identity after step 5 in which the client sends $password_{Supplicant}$ encrypted under the AS's pubkey.

If the supplicant wasn't legit, they wouldn't know the password.

The client verifies the AS after step 6 where the AS signed the supplicant's challenge with the AS's privkey.

No one but the AS and supplicant know what $challenge_{supplicant}$ is because it was sent encrypted under the pubkey of the server.

No one but the AS would be able to able to sign that challenge.

b. Is the protocol exposed to DoS attacks on the server? Explain

Yes.

Just before the AS sends a response in step 6, they need to decrypt (using the RSA privkey) the message the client encrypted. Since before decrypting, they don't know if the client is legit, an attacker would be able to send random text in step 5 and the server would have to decrypt all of the messages before realizing they're garbage.

c. Does comply with PFS? Explain

No.

The protocol is susceptible to exposed keys compromising past communication.

The only unknown in the communication is the privkey of the AS, if that is exposed, then the attacker would be able to decrypt all of the $challenge_{supplicant}$s and then calculate the XOR with $challenge_{AS}$ and thus to get the PMK for every session.

d. Is the protocol secured? Explain

The protocol is secured.

The supplicant's challenge is only known to the AS and supplicant since even when the AS respond with the signed thingy, it's only signing the hash of the challenge. Thus not exposing the challenge of the supplicant.

An attacker can't get their hands on the PMK so the protocol is secured.

e. Does the protocol protects from Rogue Access Point attack? Explain

Protected.

With the assumption that the AS sends the Authenticator the PMK in a secure manner, a rogue AP wouldn't be able to inspect the traffic as they won't have access to the PMK (as explained above).

If at any of the steps, either the AS, Authenticator and/or the supplicant get unexpected data, it might be that something (possibly rogue AP) tempered with the packets.

To actually detect:

Assuming the layout is $supplicant \leftrightarrow RogueAP \leftrightarrow Authenticator$: to detect the RogueAP the client can send a spoofed packet with the src set to the $Authenticator$ and the destination set to itself, if the TTL didn't change, then there's a Rogue AP on the network.

If the RogueAP does decrement the TTL, then we can simply send the initial TTL of the packet encrypted and the Authenticator will check if it was lowered more than once (meaning there's another endpoint between the supplicant and the Authenticator).

We can generalize the solution to fit a more complex network layout if each AP in the network is aware of the structure of the layout and is sharing a common key (or a key with each other AP).

3.2. **AS and Authenticator don't have a secured line.** Well, an attacker would be able to see the PMK going between the Authenticator and the AS and then they would be able to not only decrypt messages between the supplicants and the Authenticator (after the connection is initialized) but they will also be able to create those messages and request data from the Authenticator.

Meaning, that after an attacker saw a PMK going from the AS to the Authenticator, they would be able to impersonate the supplicant for which the PMK was originally created, including getting access to the network from the Authenticator.