Programming Project 2 - Short Answers

Leo Martel

Wednesday, Mar 11, 2015

1. This scheme is clearly insecure, because the client's response to the server's challenge does not require knowledge of the client's secret key. $SHA-256$ is a deterministic, publicly-available algorithm; an adversary could simply connect to the server and respond with $SHA-256(l)$ to impersonate any client it wishes, violating mutual authentication.

2. By pinning $cs255ca.pem$, our client trusts the cs255 CA (and no other CA). Since the TLS client only accepts connections to servers with certificates signed by this CA, any server without such a cert will be rejected at the $tls.connect$ step. Thus the only possible issuer in the "field validation" step is $cs255ca$, which we trust has no reason to lie in its certificates, so we can safely assume the "issuer" field is correct without checking it.

3. We could put the "salt" half of the private key directly into setup-cipher, rather than running it through the KDF with the password first. This would still run, since the salt is a length 128 bitarray just like the sk-der is, but this implementation would ignore the client's password. The scheme would be vulnerable to an attacker compromising the client's computer and stealing their private key; the correct implementation requires that an attack also knows or guesses the client's password (which should not be stored anywhere on disk).

   Alternatively, an implementation might forget to salt the password, instead hashing password by itself to use as the cipher's secret key. To save cycles, the implementation might also write the hashed password to disk between runs, rather than recomputing it each time. In this case, the protocol would be vulnerable to a rainbow-table attack; if an attacker stole the hashed password, they could use a precomputed lookup table to potentially find the user's password, enabling them to attack the protocol without needing to guess or steal the password from the user separately.

4. (a) One simple advantage of symmetric key encryption here is that it is faster. Since we need one encryption and one decryption for each connection, using either RSA (in which encryption is slow) or secure ElGamal (in which neither step is that fast) will be slower than using a simple symmetric-key MAC. This could be a big advantage, especially if our protocol needs to support many short-lived connections.

   (b) The disadvantage of symmetric-key challenge-response is that in order to ensure security, a client must use a distinct key with each server it wishes to connect to. It's easy to see why: if a client C shares a symmetric key k with two servers A and B, then server B can connect to A by impersonating C using k, because by definition A and C both have access to k undera symmetric-key protocol. Since $A \neq C$ can impersonate C when connecting to B, the mutual authentication of the server is violated.

   Thus, each client must store a distinct secret key for each server it wishes to connect to. On the other hand, under the public-key based system each clients needs only to store a single private key to connect to any number of servers securely. Symmetric key grows this constant space requirement to a linear one, a disadvantage.

5. (a) The server would need to observe standard password best-practices; rather than storing the password in plaintext, the server would store a unique salt and a hashed, salted password for each user. This way, if the server's database is compromised, an attacker cannot impersonate all of the server's clients. They would first need to brute force the hashes–and due to the salting, each brute force attack would only reveal one client's password, even if some clients chose the same password.

(b) Using the bad certificate, the attacker connects to the client, claiming to be the server. The client checks the bad certificate, which validates, and then sends their password to the attacker. The attacker now simply records the plaintext of the password.

Later on, the attacker can connect to the server directly and impersonate the client by using the stolen password. The bad cert is no longer needed!