Computer Security (Sheet #1)

Marius Gavrilescu

1. If we allowed everyone to edit the password file, everyone could change another user's password to what they wanted and could then log in as that user, so every file that is accesible by one user could be accessed by any user.

   Restricting access to low-numbered internet ports means a user-controlled app cannot pretend to be a common system service. If this was possible then an user app could wait for a service to go down and then pretend to be that service. If that service was (for example) sshd, the user app would get login information from clients thinking it is the real sshd. If that service was smtp, the user app could read email it should not be allowed to read.

   Programs can be declared as 'setuid', which means that they will run as the user who owns the executable and not as the user who runs the program. A setuid program still knows who actually called it, so it can enforce access control: for example when passwd is run by anyone it needs write access to the password file, but if it is called by a non-superuser it will only permit changing that user's password.

   This is a simple way to permit regular users to access these functions, but it comes at the cost of duplicating access control (as each setuid app needs to make its own access control decisions: it has full access to everything, but it shouldn't give the user who calls it access to everything!). This feature also enables some security bugs: if a 'setuid' app does not have the correct permissions a regular user might be able to overwrite it with an app of their choice (thereby giving themselves superuser permissions).

2. With n = 37 (a prime number), for a given ciphertext c each key will decrypt to a different ciphertext (to prove this, assume the opposite: two keys $k_1 \neq k_2$ encrypt m to the same thing; this means $m \cdot (k_1 - k_2)$ must be a multiple of n. But k_1, m_2, m are all smaller than n and n is prime, so this is not possible).

   That means that knowing the ciphertext offers no information about the message, so we have perfect security.

   With n = 27 = 3 * 3 * 3, messages that are multiples of 3 will always encrypt to multiples of 3. If the ciphertext is 2, we know the message can't be a multiple of 3. Thus the ciphertext can offer some information about the message, so we do not have perfect security.

3. The output of the first round is $< y, S(y) \oplus K_1 \oplus x >$. The output of the second round is $< S(y) \oplus K_1 \oplus x, S(S(y) \oplus K_1 \oplus x) \oplus K_2 \oplus y >$

   If we have a plaintext-ciphertext pair, that means we know $A = S(y) \oplus K_1 \oplus x$ and $B = S(S(y) \oplus K_1 \oplus x) \oplus K_2 \oplus y$ for some x and y. But then we can compute $S(y)$ which allows us to obtain K_1 by xoring A with $S(y) \oplus x$, and then we can compute $S(S(y) \oplus K_1 \oplus x)$ which allows us to obtain K_2 by xoring B with $S(S(y) \oplus K_1 \oplus x) \oplus y$, thus breaking the key.

4. The ciphertext is still scrambled: the initial permutation, E-boxes, S-boxes and P-boxes still apply every round. However, as the all-zeroes key is a weak key in DES the decryption function we obtain is identical to the encryption function which means double-encrypting does not scramble the plaintext at all!

5. Fix the message m. If $k > n$, there will necessarily exist multiple keys that encrypt m to the same thing. This is because there are only 2^n possible encryptions of m (every ciphertext), yet there are 2^k keys. Because of this, if we know the encryption of m there can be multiple keys that encrypt m to that ciphertext.

   For the meet-in-the-middle attack we need to store $< k, E_k(m) >$ for all keys k. Each pair uses $n + k$ bits of storage, and we have 2^k such pairs. Total storage space is $2^k(n + k)$. For DES, this is $2^{56} \cdot 120$ bits, or about 1.08 exabytes.

6. The receiver looks at the last byte. Its value indicate exactly how many bytes of padding there are, so the receiver can remove this many bytes from the end. Since we always pad (even if the input size is a multiple of 8), the last byte will always indicate the padding.

   For a random 64-bit block, if the last byte is K then the block is correctly padded if $1 <= K <= 8$ and the previous $K - 1$ bytes also have value K. The probability of a certain byte having a certain value is $\frac{1}{256}$, so the probability of the block being padded correctly is $\frac{1}{256} + \frac{1}{256^2} + \ldots + \frac{1}{256^8}$ which is about 0.4%.

7. Here c_1 is the IV, so c_2 and c_3 will be decrypted correctly. Then $c_4^{'}$ will decrypt to $m_3^{'} \oplus c_3^{'} \oplus c_3$ and in general $c_{k+1}^{'}$ will decrypt to $m_k^{'} \oplus c_k^{'} \oplus c_k$.

   This means the attacker can xor the plaintext with any string of their choice. If the attacker has a good idea of what the message (or part of it) then they can change parts of the message.

   For example if the message is known to be "The attack shall take place at XX:YY" we can xor the second digit of the hour with 1 to change the time by one hour.

   The use of a message authentication code would prevent this problem, as the receiver could then detect if the message was altered by some attacker.