Security & Networks — Summer 2023 Solutions

Q1(a): Vigenere Cipher

university = 20 13 8 21 4 17 18 8 19 24
birmingham = 1 8 17 12 8 13 6 7 0 12
Plain + Key = 21 21 25 33 12 30 24 15 19 36
(Plain + Key) mod 26 = 21 21 25 7 12 4 24 15 19 10

Re-encoded: vvzhmeyptk

Q1(b): RSA Malleability

Enc(PK, m) = m^e mod N

The following ciphertext c' will produce an encryption of 2m:
c' = (2^e * c) mod N

Enc(PK, 2m) = (2m)^e mod N
= (2^e * m^e) mod N
= (2^e * c) mod N

Q1(c): CBC-MAC Forgery

We request a MAC for m0:
E_K(m0) = t0

We consider:
m1 = m0 XOR t0

Then the output of the second block cipher call is:
E_K(m1 XOR t0) = E_K(m0 XOR t0 XOR t0) = E_K(m0) = t0

The tag for m0||m1 = m0||(m0 XOR t0) is t0.
The forgery pair is message: m0||(m0 XOR t0) with the tag: t0

Q2(a): Self-Signed Certificates

This is not secure, as it would allow an attacker to run Man-in-the-Middle attacks using self-signed certificates. The attacker could decrypt all traffic using its own certificate and re-encrypt and forward the traffic to the intended party.

Q2(b): Forward Secrecy and Malware Checking

This is not possible because the malware-checking server lacks the ephemeral/session keys. By the definition of forward secrecy (session keys will not be compromised even if long-term secrets used in the session key exchange are compromised), the malware-checking server will not be able to decrypt the traffic without the session keys.

The webserver could decrypt and re-encrypt the traffic with a key for the malware-checking server. Alternatively, the webserver could also transfer the required session keys to the malware-checking server.

Q2(c): CSRF via MAC

This scheme does not prevent CSRF attacks because the MAC in the hidden form field is constant for each form (as it is the MAC of the unique identifier only). Thus, the attacker can log in to the webpage, retrieve the form field first, and send along the relevant value.

A simple solution is to add a nonce to the plaintext, which will guarantee

different MAC outputs on every request.

Q3(a): Stack Layout and Buffer Overflow

(i) Stack layout (top = low address):

```
-- TOP --
password_buffer (16 bytes)
canary          ( 1 byte )
authenticated   ( 4 bytes)
old EBP         ( 4 bytes)
return address  ( 4 bytes)
*password       ( 4 bytes)
-- BOTTOM --
```

(ii) The attack is a buffer overflow in the strcpy call (line 6). strcpy does not check the length of password_buffer (16 bytes) and will continue writing into the canary and authenticated fields.

(iii) The input needs to be at least 18 bytes. The first 16 bytes can be random; the 17th byte must be 'a' to pass the canary check, followed by something non-zero to make authenticated non-zero.
Example: "aaaaaaaaaaaaaaaaaa" (18 x 'a')