Autenticacion en Criptografia

Valentina S. Vispo

Octubre 2023

Índice

1 Desafio: Ataque de extensión de longitud

1.1 Enunciado

El desafío consiste en crear una falsificación de un mensaje autenticado con una MAC creada a partir de una función de hash y un secreto.

Existen 3 pares clave-valor:

- $user = user@example.com$
- $action = show$
- $mac = 55d7c19d9e4d5427d5cadd309b58a5fdfb0f78b9b40671d65c73df5ed4476784$

La falsificación debe ser una query string válida que incluya el par $admin = true$.

Para calcular el MAC, se construye un mensaje formado por la concatenación del resto de los pares clave-valor, ordenados alfabéticamente por clave, y eliminando los símbolos = y &. En el caso anterior, el mensaje sería: $actionshowuseruser@example.com$

Sobre este mensaje se calcula un MAC de la siguiente forma:

$$MAC = SHA-256(secreto||mensaje)$$

La query string falsificada, con las siguientes propiedades:

- Debe ser una query string válida, con pares atributo-valor de la forma $atributo = valor$ separados por el carácter &
- Debe contener un campo $user$
- Debe contener un campo $admin$ con el valor $true$
- Debe contener un MAC correcto, que corresponda con los datos enviados

1.2 Analisis

El servidor dado una query enviado por el post, procede a ordenar las claves en orden alfabetica para luego eliminar los & y los =.

Supongamos que queremos crear un MAC (Message Authentication Code) a partir de un secreto y una funcion de hash.

Una posible implementacion podria ser, dados una funcion de hash H un mensaje M y un secreto K:

$$MAC_K(M) = H(K||M)$$

Como lo que salgo del hash es uno de los estados intermedios, no es seguro!!

La idea es que un atacante que solo conoce M y $MAC_K(M)$ pero ignora el secreto, no puede generar un MAC valido para un mensaje modificado M'.

El problema es que hay muchas funciones de hash para lo cual no es cierto. Afectan a ciertas funciones de hash en las que el ultimo estado se usa como resultado (ataque de extension de longitud).

Afectan a ciertas funciones de hash en las que el ultimo estado se usa como resultado.

Ocurren cuando se usa la funcion de hash como MAC, aplicando $H(secreto||mensaje)$.

Si el atacante econoce el mensaje y el hash, puede construir una extenson del mensaje y calcular un hash valido aun cuado no conozca el secreto.

Es decir que puede calcular de la siguiente forma:

$$H(secreto||mensaje||agregado) \text{ — aun cuando no se conozca el secreto}$$

Entonces, dado un mensaje M, el padding orignal P, como atacante podemos construir un mensaje:

$$M' = M||P||X \text{ y calcular } MAC_K(M') = H'(X)$$

H': es la misma funcion de hash que H solo que usando $MAC_K(M)$ como inicializador (IV)

1.3 Solucion

Pasos

1. Obtener el mensaje de la request (GET). Convertilo de bytes a string

```
b'user=valentina.vispo@mi.unc.edu.ar&action=show&mac=3496
   bd0d75e25eed0a4508c6f6c41941fde5a8583c826e32b2369a86bf5cd0bf'
```

2. Eliminar los caracteres & y dividirlos en los terminos de clave-valor

```
terms = msg_str.split('&')

# user=valentina.vispo@mi.unc.edu.ar
# action=show
# mac=3496bd0d75e25eed0a4508c6f6c41941fde5a8583c826e32b2369a86bf5cd0bf
```

2. Luego, se elimina los caracteres = y se encodea a ascii.
3. En particular para el termino que contiene la mac, se elimina $mac =$
4. Se extiende el mensaje a partir del MAC original, obteniendo asi el $fake\_mac$ y el pad

$fakeMac = SHA-256(extension, MAC, longitud)$

```
#* Nuevo_MAC = SHA-256(extension, MAC, longitud)
fake_mac, pad = extend_message(action, user, mac)
# ...
def extend_message(action: bytes, user: bytes, mac: str) -> bytes:
    """
    mensaje || relleno || extension
    """
    #* Tiene 16b
    fake_secret = b'1234567891111111'
    to_hash = action + user + fake_secret
    original_msg_with_pad = add_pad_sha256(to_hash)
    print(len(original_msg_with_pad))

    pad = original_msg_with_pad[len(to_hash):]

    break_msg = b'admintrueuservalentina.vispo@mi.unc.edu.ar'
    iv = iv_block(mac)
    fake_mac = SHA256(break_msg, iv, len(original_msg_with_pad))
```

```
        print(fake_mac.hexdigest())
        return fake_mac.hexdigest(), pad
    # ...
```

5. Al *pad* se le realiza una limpieza con la funcion *quote* para poder ser utilizado en la url

```
    from urllib.parse import quote
    # ...
    pad_parse = quote(pad)
```

6. La falsificación debe ser una query string válida que incluya el par *admin* = *true*

```
    # ...
    # fake_mac =
       eec7ed400eced2c2c468d0f5541798d96da715fd300a64cf33ee903d49b0263a
    query_without_mac = f'a=ctionshowuservalentina.vispo@mi.unc.edu.ar'
    extension = f'user=valentina.vispo@mi.unc.edu.ar&admin=true'
    break_query = f'{query_without_mac}{pad_parse}&{extension}&mac={
       fake_mac}'
    # ...
```

7. Se envia la respuesta con *break_query*

```
    Query: a=ctionshowuservalentina.vispo@mi.unc.edu.ar
       %80%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%
       D8&user=valentina.vispo@mi.unc.edu.ar&admin=true&mac=
       eec7ed400eced2c2c468d0f5541798d96da715fd300a64cf33ee903d49b0263a

    Ganaste
```

1.3.1 Como correr la solución

```
python3 challenges/7_Secret_Prefix_MAC.py
```

1.3.2 Evitar el problema

Deberia **agregar el** *secret* **al final de todo** para que no pueda ser utilizado por un atacante. Otra forma de evitar el problema es utilizar una funcion de hash que sea **resistente al ataque de extension de longitud**.

1.4 Herramientas

```
def add_pad_sha256(message: bytes) -> bytes:
    length = len(message)
    bitlen = length * 8
    complete_block_length = length - length % 64
    first_part = message[:complete_block_length]
    last_block = message[complete_block_length:]
    padding_blocks = SHA256._pad_message(last_block, bitlen)
```

```
        return first_part + b"".join(padding_blocks)
```

1.5 Referencias

1. Enunciado ejercicio #https://ciberseguridad.diplomatura.unc.edu.ar/cripto/doc/spm.html
2. https://dl.packetstormsecurity.net/0909-advisories/flickr_api_signature_forgery.pdf
3. Notas de clase - borrador