MD5 Road: The Revenge

Descripción del Reto

El año pasado Mike no fue capaz de ver venir algunos obstáculos, aunque algunos de vosotros le conseguisteis ayudar. Este año, el desafío es mayor, el camino es más largo, pero vosotros tenéis más experiencia... ¿no?

Análisis Inicial

Al conectarnos al servidor, recibimos un banner ASCII que incluye un hash MD5 (32 caracteres hexadecimales) etiquetado como `bus_id`. El servidor nos da 1000 oportunidades ("avisos") para enviar un "obstáculo" en formato hexadecimal, y por cada obstáculo enviado, el servidor responde con otro hash MD5.

El código fuente del servidor revela la lógica interna:

1. Se genera un `secret = urandom(52)` que tenemos que recuperar
2. Se calcula `bus_id = md5(secret).hexdigest()` (tenemos este valor)
3. En un bucle de 1000 iteraciones:
    - El servidor recibe un `obstacle` del usuario
    - Calcula `H = md5(secret + obstacle + secret[:randint(0, len(secret))]).hexdigest()`
    - Envía `H` de vuelta al cliente
4. Finalmente, pide una suposición del secreto para dar la flag

La parte crucial es `secret[:randint(0, len(secret))]`. En cada iteración, se añade un sufijo aleatorio que es un prefijo del propio secreto, con longitud variable entre 0 y 52 bytes. Esta aleatoriedad complica los ataques de fuerza bruta convencionales.

La Vulnerabilidad: Ataque de Extensión de Longitud

MD5 es vulnerable a ataques de extensión de longitud. Si conocemos:

- El hash de una clave secreta: `H = hash(key)`
- La longitud de la clave: `L = len(key)`

Podemos calcular `hash(key + padding + data_adicional)` para cualquier `data_adicional` sin conocer la clave original.

En nuestro caso:

- Conocemos `bus_id = md5(secret)`
- Sabemos que `len(secret) = 52 bytes`
- Podemos usar herramientas como `hashpumpy` para realizar este ataque

Estrategia de Explotación

Nuestra estrategia es manipular el `obstacle` que enviamos para que el hash calculado por el servidor coincida con la estructura de un ataque de extensión de longitud.

Paso 1: Calcular el Padding

El padding que MD5 añadiría a `secret` (52 bytes) sería:

```
pad_secret = b'\x80\x00\x00\x00\xa0\x01\x00\x00\x00\x00\x00\x00'
```

(12 bytes en total, incluyendo el byte 0x80, bytes de relleno, y la longitud original en bits).

Paso 2: Enviar el Padding como Obstáculo

Cuando enviamos `obstacle = pad_secret`, el servidor calculará:

```
H_servidor = md5(secret + pad_secret + secret[:L_s])
```

donde `L_s` es una longitud aleatoria entre 0 y 52.

Paso 3: Aprovechar la Estructura

Observamos que `secret + pad_secret + secret[:L_s]` tiene exactamente la estructura de un mensaje procesado por un ataque de extensión de longitud sobre `md5(secret)`, donde `secret[:L_s]` son los datos añadidos.

Plan de Implementación

Fase 1: Recolección de Hashes

- Conectarse al servidor y extraer `bus_id`
- Calcular `pad_secret`
- Enviar `pad_secret` (en hexadecimal) como `obstacle` 1000 veces
- Almacenar todos los hashes únicos recibidos

Fase 2: Recuperación Byte a Byte

- Inicializar `known_prefix = b''`
- Para cada posición del byte en el secreto (0 a 51):
  - Probar cada valor posible (0 a 255)
  - Formar el prefijo candidato: `current_guess = known_prefix + bytes([b])`
  - Verificar usando extensión de longitud si este prefijo produce un hash que coincide con alguno observado
  - Si hay coincidencia, hemos encontrado el byte correcto

Fase 3: Obtener la Flag

- Enviar el secreto recuperado al servidor
- Recibir la flag

Implementación (Script Python)

```
#!/usr/bin/env python3

import sys
import logging
from pwn import *
from hashpumpy import hashpump

# Configuración
HOST = "ctf.hackademics-forum.com"
PORT = 42836
```

```
context.log_level = 'info'

# Constantes
SECRET_LEN = 52
pad_secret = b'\x80\x00\x00\x00\xa0\x01\x00\x00\x00\x00\x00\x00'  # Padding MD5 para 52 bytes
obstacle_hex = pad_secret.hex()
PROMPT_DELIMITER = b'Avisa de un obst\xc3\xa1culo: '
FINAL_PROMPT_DELIMITER = b'Secreto: '

# Función para encontrar un byte específico
def find_specific_byte(k, known_prefix, observed_hashes, bus_id):
    log.info(f"Buscando byte {k}...")
    for b in range(256):
        current_guess = known_prefix + bytes([b])
        try:
            computed_hash, _ = hashpump(bus_id, b'A', current_guess, SECRET_LEN)
            computed_hash_lower = computed_hash.lower()
            if computed_hash_lower in observed_hashes:
                log.success(f"Encontrado byte {k}: {hex(b)}")
                return b
        except Exception as e:
            continue
    return -1

# Main Script
try:
    conn = remote(HOST, PORT)
    # Extraer bus_id del banner
    banner_data = conn.recvuntil(PROMPT_DELIMITER, timeout=10)
    bus_id = ""
    for line in banner_data.split(b'\n'):
        parts = line.strip().split(b'|')
        if len(parts) >= 3 and len(parts[^1]) == 32:
            potential_id = parts[^1].decode('ascii', errors='ignore')
            if all(c in '0123456789abcdefABCDEF' for c in potential_id):
                bus_id = potential_id.lower(); break
    log.success(f"Extraído bus_id: {bus_id}")
```

```
    # Fase 1: Recolección de hashes
    observed_hashes = set()
    log.info("Recolectando 1000 hashes...")
    for i in range(1000):
        conn.sendline(obstacle_hex.encode())
        response_line = conn.recvline(timeout=2)
        if not response_line: continue
        response = response_line.strip().decode()
        if len(response) == 32 and all(c in '0123456789abcdefABCDEF' for c in response):
            observed_hashes.add(response.lower())
        if i < 999:
            prompt = conn.recvuntil(PROMPT_DELIMITER, timeout=2)
            if not prompt.endswith(PROMPT_DELIMITER): break
    log.success(f"Recolección finalizada. Hashes únicos: {len(observed_hashes)}")

    # Fase 2: Recuperación byte a byte
    known_prefix = b''
    for k in range(SECRET_LEN):
        byte_val = find_specific_byte(k, known_prefix, observed_hashes, bus_id)
        if byte_val == -1: raise ValueError(f"Fallo al recuperar byte {k}.")
        known_prefix += bytes([byte_val])
        log.info(f"Prefijo actual ({k+1}/{SECRET_LEN}): {known_prefix.hex()}")

    # Fase 3: Enviar secreto y obtener flag
    log.success("¡Secreto recuperado!")
    conn.recvuntil(FINAL_PROMPT_DELIMITER, timeout=5)
    conn.sendline(known_prefix.hex().encode())
    result = conn.recvall(timeout=10).decode(errors='ignore')
    log.success("Respuesta final:")
    print(result.strip())

except Exception as e:
    log.error(f"Error: {e}")
finally:
    if 'conn' in locals() and conn and conn.connected:
        conn.close()
```

Ejecución y Resultado

Al ejecutar el script:

1. Se conecta al servidor y extrae el `bus_id`
2. Recolecta los 1000 hashes (generalmente obteniendo 53 únicos, correspondientes a sufijos de longitudes 0 a 52)
3. Procede a recuperar el secreto byte a byte
4. Finalmente, envía el secreto recuperado y recibe la flag

Flag: `hfctf{h45h_l3ngTH_3xt3ns10n_att4Ck_n0_14S_4Ll0w3D!!}`