Writeup: Signed to See

Una plataforma de almacenamiento seguro protege sus archivos con una firma muy segura... O puede que no...

Análisis del código

Al examinar el archivo `index.php` proporcionado, podemos identificar los siguientes elementos clave:

- El sistema utiliza una clave secreta (`$secret`) obtenida de una variable de entorno
- Solo se permiten solicitar dos archivos: `test.txt` y `flag.txt`
- Se requieren dos parámetros POST: `file` (nombre del archivo en hexadecimal) y `key` (firma)
- El proceso de validación incluye:
    - Decodificar `file` de hexadecimal a texto
    - Calcular una firma SHA1 con `sha1($secret . $reqFile)`
    - Verificar que el archivo solicitado termine en `.txt`
    - Comprobar que el archivo esté en la lista de permitidos
    - Verificar que la firma proporcionada coincida con la calculada

La página web nos proporciona información adicional crucial:

- Archivos disponibles: `["test.txt", "flag.txt"]`
- Firma de test.txt: `c9d4b01ce16b640782af2864a47547d88fc02cab`

Identificación de la vulnerabilidad

La forma en que se calcula la firma (`sha1($secret . $reqFile)`) es vulnerable a un **Hash Length Extension Attack (HLEA)**. Este tipo de ataque aprovecha las propiedades internas de funciones hash como SHA-1 y MD5.

Las funciones hash como SHA-1:

- Son construcciones Merkle–Damgård
- Procesan datos en bloques y mantienen un estado interno
- Si conocemos el hash de `SECRET + DATA` y la longitud total, podemos calcular el hash de `SECRET + DATA + PADDING + EXTRA_DATA` sin conocer el valor de `SECRET`

Lo crucial en este caso es que:

- Conocemos `hash = sha1($secret + "test.txt")`
- Conocemos `data = "test.txt"` (8 bytes)
- No conocemos `$secret` ni su longitud
- Queremos obtener `flag.txt`

Estrategia de explotación

1. **Adivinar la longitud del secreto**: Necesitamos probar diferentes longitudes (1-64 bytes típicamente)
2. **Aplicar HLEA**: Usando una herramienta como hashpump, hlextend o hashpumpy
3. **Construir el payload**:
    - Añadir `flag.txt` al final de `test.txt` (con padding)
    - Calcular la nueva firma
    - Enviar la petición y verificar respuesta

La clave está en que aunque calculamos la firma con toda la cadena, el script PHP extrae el último archivo que coincide con `.txt` usando `preg_match`. Así podemos hacer que lea `flag.txt` aunque nuestro payload comience con `test.txt`.

Implementación del exploit

```
import requests
import binascii
from hlextend import Sha1

url = "http://[IP_DEL_RETO]/index.php"
```

```
known_hash = "c9d4b01ce16b640782af2864a47547d88fc02cab"
known_data = b"test.txt"
data_to_append = b"flag.txt"
max_key_length = 64

for key_len in range(1, max_key_length + 1):
    try:
        # Calcular nueva firma y mensaje usando HLEA
        ext = Sha1()
        new_hash, new_message_bytes = ext.extend(data_to_append, known_data, key_len, known_hash)

        # Preparar datos POST
        post_key = new_hash
        post_file_hex = binascii.hexlify(new_message_bytes).decode('ascii')

        payload = {'file': post_file_hex, 'key': post_key}
        response = requests.post(url, data=payload, timeout=10)

        # Verificar éxito
        if "Invalid file or signature!" not in response.text:
            print(f"[+] ¡Éxito con longitud de clave {key_len}!")
            print("[+] Respuesta (Flag):")
            print(response.text)
            break
    except Exception as e:
        print(f"[*] Error con longitud {key_len}: {e}")
```

Resultado

Al ejecutar el script, eventualmente encontramos la longitud correcta del secreto y recibimos el contenido de `/files/flag.txt`:

Flag: `hfctf{haSh_Ext3nd3r_FlaggED}`