Pasty – Writeup (Nullcon CTF)

Resumen

- **Servicio:** Pastebin “seguro” con firmas criptográficas propias.
- **Objetivo:** Acceder al paste con id `flag`.
- **Flag:** `ENO{cr3at1v3_cr7pt0_c0nstruct5_cr4sh_c4rd5}`

Reconocimiento

1. **Frontend:** Crear paste en `/` (POST a `create.php`) → el servidor devuelve una URL de la forma:

   ```
   http://52.59.124.14:5005/view.php?id=<HEX_ID>&sig=<HEX_SIG>
   ```

2. **Verificación:** Para ver un paste se requieren `id` y `sig`; el servidor comprueba la firma antes de mostrar el contenido.
3. **Archivo `sig.php`:** Contiene la lógica de firma “custom” que debemos analizar.

Análisis del esquema de firma (`sig.php`)

```
function _x($a,$b)
{$r='';for($i=0;$i<strlen($a);$i++)$r.=chr(ord($a[$i])^ord($b[$i]));return
$r;}
function compute_sig($d,$k){
  $h=hash('sha256',$d,1);                                    // 32 bytes
  $m=substr(hash('sha256',$k,1),0,24);                       // 24 bytes
(clave derivada)
  $o='';
  for($i=0;$i<4;$i++){
    $s=$i<<3;
    $b=substr($h,$s,8);
    $p=(ord($h[$s])%3)<<3;
    $c=substr($m,$p,8);
    $o.=($i?_x(_x($b,$c),substr($o,$s-8,8)):_x($b,$c));
  }
  return $o;
}
```

- **Entrada:** `$d` = dato a firmar (en la app, el **id** del paste como string, p. ej. `"e9a04902a7dc8a4d"` o `"flag"`), `$k` = clave secreta.
- **Salida:** 32 bytes (firma en bruto; en la URL se muestra en hex, 64 caracteres).

Paso a paso del algoritmo

1. `h = SHA256(d)` en binario → 32 bytes.
2. `m = primeros 24 bytes de SHA256(k)` → se interpretan como **3 bloques de 8 bytes**: `m0 = m[0:8]`, `m1 = m[8:16]`, `m2 = m[16:24]`.
3. Para cada bloque `i = 0..3`:
   - `b_i = h[8*i : 8*i+8]` (8 bytes de `h`).
   - `p = (h[8*i] % 3) * 8` → valor en `{0, 8, 16}` → se elige uno de los tres bloques de `m`: `c_i = m[p:p+8]` (es decir, `m0`, `m1` o `m2`).
   - Salida del bloque:
     - `i = 0`: `o_0 = b_0 XOR c_0`
     - `i > 0`: `o_i = (b_i XOR c_i) XOR o_{i-1}` (encadenamiento tipo CBC con XOR).
4. Firma final: `sig = o_0 || o_1 || o_2 || o_3` (32 bytes).

La vulnerabilidad no está en SHA256, sino en que **el material de clave efectivo son solo 24 bytes (3×8)** y **cada bloque de la firma revela uno de esos tres subbloques** cuando tenemos el par `(d, sig)` y podemos calcular `h = SHA256(d)`.

Idea del ataque

- Para cada par `(id, sig)` que obtenemos al crear pastes:
  - Conocemos `id` (y por tanto `h = SHA256(id)`).
  - La firma nos da `o_0, o_1, o_2, o_3` (4×8 bytes).
  - Para cada bloque `i` podemos despejar el subbloque de clave usado:
    - `o_0 = b_0 XOR c_0` → `c_0 = b_0 XOR o_0`.
    - `o_i = (b_i XOR c_i) XOR o_{i-1}` → `c_i = b_i XOR o_i XOR o_{i-1}`.
  - El índice del subbloque usado en el bloque `i` es `idx = h[8*i] % 3`, así que estamos recuperando `m0`, `m1` o `m2` según el caso.
- Creando bastantes pastes, los `id` (y por tanto los `h`) varían; estadísticamente en pocas peticiones habremos visto los tres índices para suficientes bloques como para tener **m0, m1 y m2 completos**.
- Con los 24 bytes recuperados podemos calcular **la misma firma** que calcularía el servidor para cualquier `d`, en particular para `d = "flag"`, y acceder al paste de la flag.

Implementación

1. **Recolección de pares (id, sig):**
   Crear pastes (POST a `create.php`); el servidor responde con **302** y en el header `Location` viene algo como:

```
index.php?
msg=created&url=http%3A%2F%2F...%2Fview.php%3Fid%3D<ID>%26sig%3D<SIG>
```

Decodificando el parámetro `url` se extraen `id` y `sig` (hex).

2. **Recuperación de m0, m1, m2:**
   Para cada `(id, sig)`:
   - Calcular `h = SHA256(id)`.
   - Partir `sig` en 4 bloques de 8 bytes → `o_0..o_3`.
   - Para cada bloque `i`, calcular `c_i` como arriba y asignar `m_blocks[h[8*i] % 3] = c_i` hasta tener las tres posiciones rellenadas.
3. **Falsificación de firma para `"flag"`:**
   - Calcular `h = SHA256("flag")`.
   - Aplicar el mismo bucle del `compute_sig` usando los `m_blocks` recuperados y obtener la firma en hex.
4. **Petición al paste de la flag:**
   `GET view.php?id=flag&sig=<firma_calculada>`.

El script `exploit.py` automatiza estos pasos; con ~80 pastes suele ser suficiente para recuperar los tres bloques y obtener la flag.

Conclusión

El reto muestra los riesgos de **diseños criptográficos propios**: aunque se use SHA256, un esquema que:

- Deriva solo 24 bytes de clave y los reutiliza en pocos bloques,
- Expone la “salida” por bloques en una estructura lineal (XOR + encadenamiento),

permite **recuperar todo el material de clave efectivo** a partir de mensajes firmados legítimos y luego **firmar cualquier mensaje**, incluido `id=flag`. La moraleja: usar primitivas estándar (HMAC, Ed25519, etc.) en lugar de construcciones ad hoc.