Memahami Session Hijacking

Serangan dan Penanggulangan dengan Praktik Go & CLI

Apa itu Session Hijacking?

Definisi

Pengambilalihan sesi pengguna secara tidak sah untuk mendapatkan akses ke sistem atau data.

Cara Kerja

Penyerang mencuri atau memprediksi token sesi (session ID) yang valid milik pengguna.

Konsekuensi

Pencurian identitas, penipuan finansial, pencurian data sensitif, dan pengambilalihan akun penuh.

Vektor Serangan Umum

Spoofing (Penebakan)

Penyerang mencoba menebak session ID yang valid. Sulit dilakukan jika tokennya acak dan panjang.

Session Fixation

Penyerang "memaksa" pengguna untuk menggunakan session ID yang sudah diketahui oleh penyerang.

Session Stealing

Mencuri token sesi melalui sniffing (MITM) di jaringan yang tidak aman (HTTP) atau melalui XSS.

Serangan 1: Session Fixation

Langkah Serangan

1. Penyerang membuat session ID dan mengirimkannya ke korban:

```
# Penyerang mengirim link
http://example.com/?session_id=ATTACKER_ID
```

2. Korban mengklik link, masuk (login). Server kini mengasosiasikan `ATTACKER_ID` dengan akun korban.

3. Penyerang menggunakan `ATTACKER_ID` untuk mengakses akun korban.

Pencegahan (Golang)

Selalu regenerasi token sesi SETELAH pengguna berhasil login.

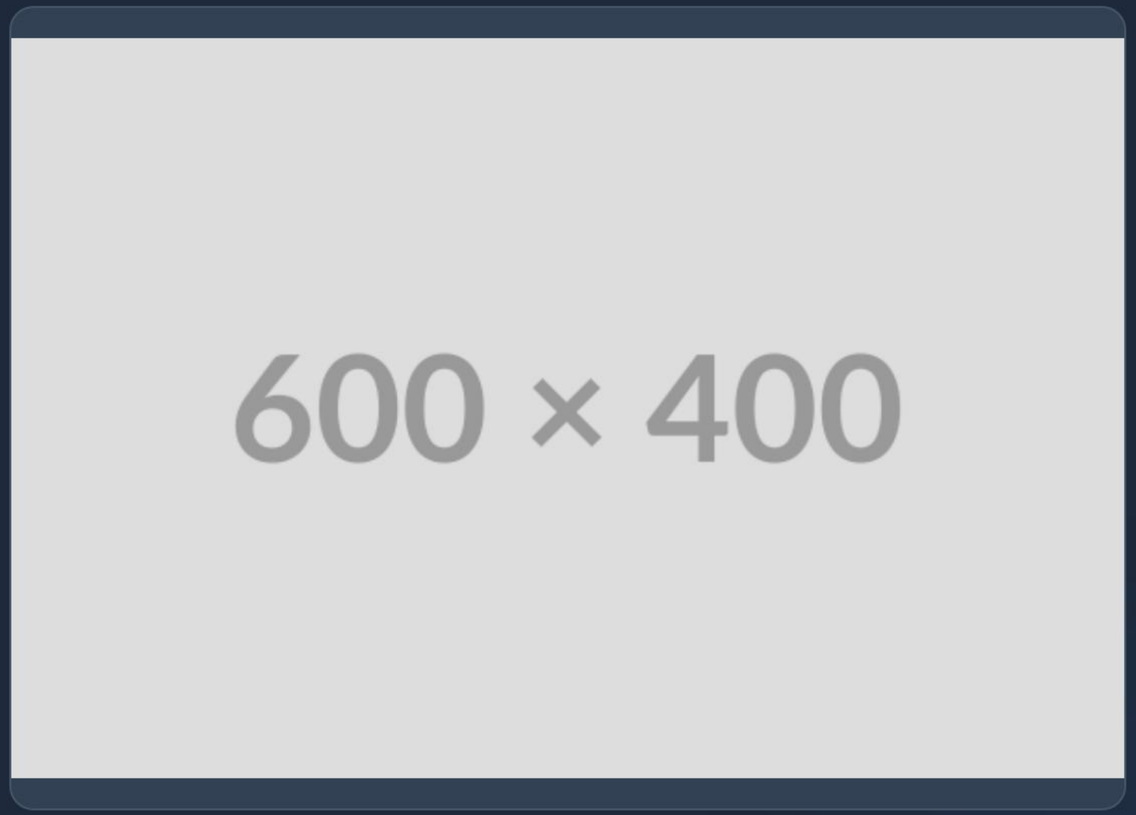
```go
// Menggunakan library alexedwards/scs
func loginHandler(w http.ResponseWriter, r *http.Request) {
    // ... validasi username/password ...

    // PENTING: Regenerasi token
    err := sessionManager.RenewToken(r.Context())
    if err != nil {
        http.Error(w, err.Error(), 500)
        return
    }

    // Simpan data di sesi baru
    sessionManager.Put(r.Context(), "userID", 123)
}
```

Serangan 2: Session Stealing (MITM)

Pencurian Sesi (Man-in-the-Middle)

Jika aplikasi berjalan di HTTP (bukan HTTPS), cookie sesi dikirim sebagai teks biasa.

- Penyerang di jaringan yang sama (misal, Wi-Fi publik) dapat "mengendus" (sniff) traffic.
- Alat seperti Wireshark atau `tcpdump` dapat menangkap paket jaringan.
- Penyerang mengekstrak `Cookie: session_id=...` dari header HTTP dan menggunakannya.
- Ini adalah alasan mengapa HTTPS adalah WAJIB.



Praktik (CLI): Sniffing Cookie

Di jaringan yang tidak aman, penyerang menjalankan `tcpdump` untuk mendengarkan traffic ke `example.com` di port 80 (HTTP).

```
# Jalankan sebagai root untuk menangkap paket
sudo tcpdump -i wlan0 -A 'port 80 and host example.com'
```

Penyerang akan melihat output seperti ini saat korban menjelajah:

```
...
GET /dashboard HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 ...
Cookie: session_id=PLAINTEXT_COOKIE_VALUE_123
Accept: text/html,...
...
```

Praktik (CLI): Menggunakan Cookie

Penyerang kini menggunakan cookie yang dicuri dengan `curl` untuk meniru identitas korban.

```
# Mengirim request ke halaman dashboard
curl "http://example.com/dashboard" \
     -H "Cookie: session_id=PLAINTEXT_COOKIE_VALUE_123"
```

Hasil:

```
Selamat Datang, [Nama Korban]!
Ini adalah data rahasia Anda...
```

Penyerang berhasil masuk sebagai korban tanpa perlu password.

Pencegahan (Golang): Secure Cookies

Gunakan atribut cookie di Go (`net/http`) untuk mitigasi. Ini adalah pertahanan terpenting Anda.

```
func loginHandler(w http.ResponseWriter, r *http.Request) {
    // ... setelah validasi ...
    sessionToken := "your_random_secure_token"

    cookie := http.Cookie{
        Name:     "session_id",
        Value:    sessionToken,
        Path:     "/",
        MaxAge:   3600, // 1 jam

        // PENTING UNTUK KEAMANAN:
        HttpOnly: true, // Mencegah akses via JavaScript (XSS)
```

Penjelasan Atribut Cookie

- **HttpOnly:** Mencegah skrip sisi klien (JavaScript) mengakses cookie. Ini adalah pertahanan utama terhadap pencurian cookie melalui XSS.
- **Secure:** Memastikan browser hanya mengirim cookie melalui koneksi HTTPS. Ini menggagalkan serangan sniffing (`tcpdump`) di jaringan yang tidak aman.
- **SameSite:** Mengontrol apakah cookie dikirim dengan permintaan lintas situs. `SameSite=Lax` atau `Strict` memberikan perlindungan kuat terhadap serangan Cross-Site Request Forgery (CSRF).
- **MaxAge / Expires:** Mengatur waktu kedaluwarsa cookie. Sesi yang berumur pendek mengurangi jendela waktu bagi penyerang untuk menggunakan cookie yang dicuri.

Alat Bantu Serangan (Attack Tools)

Wireshark / tcpdump

Penganalisis paket jaringan. Digunakan untuk mengendus (sniff) traffic HTTP dan mencuri cookie teks biasa.

Burp Suite

Proxy pencegat web. Digunakan untuk memanipulasi permintaan, memutar ulang (replay) sesi, dan menguji kerentanan.

Firesheep

(Sudah usang) Plugin Firefox yang mengotomatiskan pencurian cookie di jaringan Wi-Fi terbuka. Membuktikan betapa mudahnya serangan ini.

Ringkasan Deteksi & Pencegahan

Deteksi (Detection)

- **IDS (Intrusion Detection System):** Memantau anomali jaringan, seperti session ID yang tiba-tiba muncul dari IP yang berbeda.
- **WAF (Web Application Firewall):** Dapat dikonfigurasi untuk memfilter pola serangan yang diketahui.
- **Monitoring Aplikasi:** Memantau perubahan mendadak pada User-Agent atau pola akses yang tidak biasa untuk sebuah session ID.

Pencegahan (Prevention)

- **Gunakan HTTPS (Wajib!):** Mengenkripsi semua traffic, membuat sniffing tidak berguna.
- **Atribut Cookie Aman (Go):** Terapkan `HttpOnly`, `Secure`, dan `SameSite`.
- **Regenerasi Token (Go):** Selalu buat session ID baru setiap kali pengguna login atau mengubah tingkat privilege.

Questions?

Terima Kasih