OCPP Fuzzing Zafiyet Analizi Simülasyonu Gerekli Bileşenler

Amaç: Bu rapor, OCPP tabanlı şarj istasyonu (CP) ve Merkezi Yönetim Sistemi (CSMS) implementasyonlarında bulunması gereken bileşenler, **Fuzzing (Kasıtlı Hata Beslemesi)** konfigürasyonları, tespit kuralları (F1, F2, F3), test senaryoları ve önerilen uygulama adımlarını detaylı ve uygulanabilir biçimde sunar.

1. Özet

Simülasyonun amacı, bir CSMS veya CP (OCPP v2.0.1) implementasyonunun **sağlamlığını (robustness)** test etmektir. Kasıtlı olarak bozulmuş, standart dışı ve beklenmedik OCPP mesajları ("Fuzzing") göndererek, hedef yazılımın çökme (crash), donma (hang), hafıza sızıntısı veya beklenmedik durum değişikliği gibi zafiyetlerini tespit etmektir. Bu rapor, "OCPPStorm" makalesinden ilham alarak, bir Fuzzing ortamı, kuralları ve test senaryolarını içerir.

2. Gerekli Bileşenler

1. **Hedef Sistem (CSMS veya CP Emülatörü):**
    a. Test edilecek olan OCPP v2.0.1 (veya 1.6) sunucusu veya istemci yazılımı. (Örn: Açık kaynaklı bir CSMS implementasyonu).
2. **Fuzzer Aracı (Test Orkestrasyonu):**
    a. OCPP JSON şemalarını bilen, jenerasyon tabanlı (generation-based) bir fuzzer.
    b. Mesaj alanlarını (tip, uzunluk, değer) kasıtlı olarak bozabilen bir mutasyon motoru içermelidir (Örn: boofuzz, radamsa veya özel Python script'i).
3. **Ağ Konfigürasyonu:**
    a. Fuzzer'ın hedef sisteme (WSS veya WS üzerinden) bağlanabilmesi için gerekli ağ altyapısı.
4. **SIEM / Log Toplama:**
    a. Hedef sistemin (CSMS/CP) uygulama loglarını (application logs), sistem loglarını (syslog, journald) ve potansiyel çökme dökümlerini (core dumps) merkezi olarak toplayacak bir sistem (ör. Elastic Stack, Splunk).
5. **Monitoring & Dashboard (Kaynak İzleme):**
    a. Fuzzing sırasında hedef sistemin CPU, RAM ve ağ kullanımını izlemek için Grafana/Prometheus gibi araçlar. (Kaynak tüketimi zafiyetlerini bulmak için kritiktir).
6. **PKI / İç Sertifika Otoritesi (CA) (Opsiyonel):**

a. Eğer hedef sistem WSS (Güvenli WebSocket) ve mTLS zorunlu kılıyorsa, Fuzzer'ın geçerli bir istemci sertifikasına sahip olması için gereklidir.

(Not: Bu projede MitM Proxy veya Sanal CAN Arayüzü gibi bileşenlere, Fuzzing doğrudan uygulama katmanını hedef aldığı için, gerek yoktur.)

3. Kritik Yazılım ve Konfigürasyonlar

3.1 Fuzzer Konfigürasyonu (Mutasyon Stratejisi)

- **Jenerasyon-Tabanlı Fuzzing:** Fuzzer, geçerli OCPP JSON mesaj şablonlarını bilmeli.
- **Tip Mutasyonu:** Sayı (Integer) beklenen alanlara metin (String), idTag gibi alanlara negatif sayılar veya özel karakterler göndermeli.
- **Uzunluk Mutasyonu:** vendorId gibi kısa metin beklenen alanlara çok uzun (örn. 10MB) string'ler göndererek Buffer Overflow veya kaynak tüketimi denemeli.
- **Değer Mutasyonu:** meterValue gibi alanlara negatif değerler veya null göndermeli.
- **Format Mutasyonu:** Geçerli JSON yerine bozuk JSON, XML veya rastgele binary veri göndermeli.
- **Protokol Dışı Komut:** StartTransaction yerine InvalidCommand gibi protokol dışı komutlar denemeli.

3.2 Hedef Sistem Konfigürasyonu

- **Verbose Loglama:** Hedef sistem, Fuzzing denemeleri sırasında tüm hataları (exceptions, stack traces) detaylıca loglayacak şekilde yapılandırılmalı.
- **Hata Ayıklama (Debugging):** Mümkünse, hedef sistem hata ayıklama modunda çalıştırılmalı veya çökme dökümlerini (core dumps) kaydedecek şekilde ayarlanmalıdır.
- **Rate Limiting (Test Amaçlı Devre Dışı Bırakma):** Fuzzer'ın engellenmemesi için test sırasında hedef sistemin hız sınırlama mekanizmaları geçici olarak devre dışı bırakılabilir.

3.3 Loglama ve Zaman Senkronizasyonu

- Tüm bileşenler (Fuzzer, Hedef, SIEM) aynı NTP sunucusuna senkronize olmalı.
- Zorunlu log alanları: timestamp, source: "Fuzzer-01", fuzzed_message_type, fuzzed_payload_hash, target_response_code, target_error_log.

4. Tespit / Anomali Kuralları (Fuzzing Başarısı)

Aşağıdaki kurallar, Fuzzing'in başarılı olduğunu (yani bir zafiyet bulduğunu) tespit etmek için SIEM üzerinde uygulanacaktır.

- **Kural F1 — Hedef Çökmesi (Crash / Unhandled Exception)**
  - **Tanım:** Fuzzer'dan bir mesaj gönderildikten sonraki 5 saniye içinde, hedef sistemin uygulama loglarında "Unhandled Exception", "Stack Trace", "Segmentation Fault" veya "Core Dumped" gibi kritik hata loglarının gözlenmesi.
  - **Aksiyon:** Yüksek öncelikli alarm, Fuzzing'i durdur, çökme dökümünü kaydet, başarılı fuzzed_payload'u işaretle.
- **Kural F2 — Yanıt Vermeme (Hang / DoS)**
  - **Tanım:** Fuzzer'ın gönderdiği mesaja karşılık hedef sistemden 10 saniye boyunca hiçbir TCP/WebSocket yanıtı gelmemesi veya bağlantının kopması (Hedef sistemin donduğunu veya çöktüğünü gösterir).
  - **Aksiyon:** Yüksek öncelikli alarm, hedef sistemin durumunu kontrol et, fuzzed_payload'u işaretle.
- **Kural F3 — Kaynak Tüketimi (Resource Exhaustion)**
  - **Tanım:** Belirli bir fuzzed_payload (örn. çok uzun string) gönderildiğinde, hedef sistemin CPU veya RAM kullanımının 5 saniye boyunca %95'in üzerine çıkması.
  - **Aksiyon:** Orta öncelikli alarm, kaynak tüketimini doğrulayın, fuzzed_payload'u işaretle.

(Not: Orijinal belgedeki K1 , K2 ve K3 kuralları MitM ve CAN korelasyonu için olduğundan bu projeyle ilgisizdir.)

5. Metrikler ve Analiz Kuralları

- **Test Senaryosu / Saniye:** Fuzzer'ın test hızı.
- **Bulunan Eşsiz Çökme Sayısı (Unique Crashes):** Fuzzer'ın tetiklediği farklı çökme (hata) sayısı (Projenin ana başarı metriği).
- **Kod Kapsamı (Code Coverage) (İleri Düzey):** Fuzzer'ın hedef yazılımın yüzde kaçını test edebildiği.
- **Anomali Oranı:** Başarısız (F1, F2, F3 tetikleyen) istek sayısı / Toplam istek sayısı.

6. Log Biçimi Örneği (Fuzzer Logu)

JSON


```
{
  "timestamp": "2025-11-09T13:00:05Z",
  "source": "Fuzzer-01",
  "target_system": "CSMS_A",
  "session_id": "fuzz-sess-456",
  "fuzzed_message_type": "BootNotification",
  "fuzzed_payload_hash": "sha256:...",
  "fuzzed_payload_preview":
"{\"chargePointModel\":\"ABC\",\"chargePointVendor\":\"[...10MB_DATA...]\"}",
  "anomaly_flags": ["F2_Timeout","F1_Crash_Detected"],
  "target_response_log": "CRITICAL: Server crashed due to unhandled payload.
ValueError: String length limit exceeded..."
}
```


7. Önerilen Test Senaryoları (Fuzzing Vektörleri)

1. **Baseline Testi:** Normal, geçerli BootNotification -> StartTransaction -> StopTransaction akışı. (Hedefin çalıştığını doğrulamak için).
2. **Tip Manipülasyonu:** connectorId (sayı) alanına metin ("TEST") göndermek.
3. **Uzunluk Manipülasyonu (Buffer Overflow Denemesi):** idTag alanına 100.000 karakterlik bir metin göndermek.
4. **Değer Manipülasyonu:** meterStart alanına negatif bir sayı (-1000) göndermek.
5. **Format Manipülasyonu:** Geçerli JSON yerine bozuk JSON veya XML göndermek.
6. **Protokol Dışı Komut:** Authorize yerine InvalidCommand göndermek.
7. **Sıra Dışı Komut:** BootNotification göndermeden önce StartTransaction göndermeyi denemek.
8. Her senaryo için beklenen sonuç (örn. "çökme" veya "protokol hatası mesajı") dokümante edilmelidir.

8. Örnek Uygulama Adımları (Adım Adım)

1. **Ortamı Kur:** Hedef CSMS yazılımını bir sunucuya veya Docker konteynerine kur.
2. **Fuzzer'ı Hazırla:** Python'da OCPP JSON şablonlarını içeren Fuzzer script'ini yaz.
3. **Loglamayı Yapılandır:** Hedef sistemin tüm uygulama ve sistem loglarını SIEM'e yönlendir.
4. **SIEM Kurallarını Tanımla:** F1 (Crash), F2 (Hang) ve F3 (Resource) kurallarını SIEM üzerinde oluştur.
5. **Baseline Ölçümü Al:** Geçerli OCPP mesajları göndererek sistemin normal çalıştığını doğrula.
6. **Fuzzing'i Başlat:** Test senaryolarını (Bölüm 7) sırayla çalıştır.
7. **Sonuçları Kaydet:** SIEM üzerinde tetiklenen F1/F2/F3 alarmlarını, hangi payload'un (mesajın) buna neden olduğunu ve hedef sistemin loglarındaki hata dökümünü (stack trace) kaydet.

235542010-Sena Ateş