Przemysław Klejno, 09.01.2026

SPRAWOZDANIE Z PROJEKTU HONEYPOT SMB

---

1. Cel i zakres projektu

Celem projektu było stworzenie narzędzia typu **Honeypot** dla protokołu **SMB**. Jako że jest to trudny do zaimplementowania samemu protokół, zdecydowano się użyć gotowego obrazu SMB i przekazywać do niego ruch poprzez proxy, co pozwoliło przechwycić komunikację, sklasyfikować ruch i wyciągnąć wnioski. Rozwiązanie pośredniczy w komunikacji TCP między klientem a docelowym, potencjalnie podatnym serwerem (SMB - samba `4.13.7`). System ma za zadanie:

1. Przechwytywać ruch sieciowy w czasie rzeczywistym.
2. Analizować przesyłane pakiety pod kątem znanych sygnatur ataków (SQL Injection, Command Injection, próby przesyłania złośliwego oprogramowania).
3. Wykrywać anomalie wolumetryczne (skanowanie portów, ataki Brute Force).
4. Logować szczegółowe informacje o zdarzeniach do bazy danych SQLite.

2. Architektura systemu

System został zrealizowany w języku **Python** z wykorzystaniem biblioteki **asyncio**, co pozwala na wydajną, nie blokującą obsługę wielu połączeń jednocześnie. Działa w modelu **Man-in-the-Middle (MITM)**.

Schemat przepływu danych:

1. **Nasłuch (Listen):** Proxy oczekuje na połączenia na porcie 445 (domyślnie).
2. **Analiza wolumetryczna:** Po nawiązaniu połączenia sprawdzana jest historia IP w celu wykrycia skanowania.
3. **Forwarding:** Ruch jest przekazywany bajt po bajcie do docelowego serwera (TARGET HOST).
4. **Inspekcja:** Każdy pakiet jest analizowany przez moduł heurystyczny.
5. **Logowanie:** Wyniki analizy trafiają do kolejki asynchronicznej, a następnie do bazy danych.

3. Struktura danych (Baza Danych)

Do przechowywania danych zastosowano bazę **SQLite**. Zaimplementowano dwie kluczowe tabele:

3.1. Tabela `logs` (Dziennik Zdarzeń)

Jest to główna tabela przechowująca szczegółowe informacje o każdym pojedynczym pakiecie lub połączeniu, które zostało zaklasyfikowane przez system. Służy do pełnej audytowalności ruchu sieciowego (Audit Trail).

- **`id` (INTEGER, PK):** Unikalny identyfikator zdarzenia (klucz główny, autoincrement).
- **`timestamp` (TEXT):** Dokładny czas wystąpienia zdarzenia w formacie ISO 8601 (UTC).
- **`day` (TEXT):** Data zdarzenia (YYYY-MM-DD). Kolumna ta służy do szybkiego partycjonowania i filtrowania danych bez konieczności parsowania pełnego znacznika czasu.
- **`src_ip` (TEXT):** Adres IP źródła (atakującego).
- **`src_port` / `dst_port` (INTEGER):** Port źródłowy i docelowy połączenia TCP.
- **`protocol` (TEXT):** Nazwa protokołu warstwy aplikacji (w tym projekcie stała wartość: 'SMB').
- **`event_type` (TEXT):** Kierunek przepływu danych lub typ zdarzenia. Przechowuje wartości z Enuma: `clientToServer` (atak), `serverToClient` (odpowiedź), `connection_open` (skanowanie).
- **`raw` (TEXT):** Surowy payload pakietu zakodowany w **Base64**. Pozwala to na bezpieczne przechowywanie danych binarnych (np. złośliwego oprogramowania) bez ryzyka uszkodzenia struktury bazy.
- **`parsed` (TEXT):** Znormalizowana, czytelna dla człowieka wersja payloadu (tekstowa), służąca do szybkiego podglądu.
- **`classification` (TEXT):** Wynik analizy heurystycznej (np. `sql_injection`, `brute_force`, `scanning`, `benign`).
- **`confidence` (REAL):** Poziom pewności klasyfikacji w zakresie 0.0 - 1.0 (gdzie 1.0 to 100% pewności).
- **`details` (TEXT):** Pole typu JSON przechowujące dodatkowe metadane techniczne (np. wykryte wzorce ataku, listę znalezionych nazw użytkowników). Zastosowanie JSON zapewnia elastyczność schematu dla różnych typów ataków.
- **`headers` (TEXT):** Pole zarezerwowane na nagłówki (dla kompatybilności z przyszłymi modułami HTTP), obecnie przechowuje pysty obiekt JSON.

3.2. Tabela `daily_summary` (Agregacja Statystyczna)

Tabela ta służy do optymalizacji wydajności raportowania. Zamiast każdorazowo przeliczać tysiące rekordów z tabeli `logs`, system aktualizuje ten podsumowujący widok w czasie rzeczywistym (mechanizm *Upsert*).

- **`day` (TEXT, PK):** Data, której dotyczy podsumowanie. Jest to klucz główny.
- **`total_events` (INTEGER):** Łączna liczba zdarzeń zarejestrowanych tego dnia.
- **`by_class` (TEXT):** Obiekt JSON zawierający histogram klasyfikacji ataków (np. `{"sql_injection": 5, "scanning": 20}`). Pozwala to na łatwe generowanie wykresów kołowych lub słupkowych na dashboardzie.
- **`first_seen` (TEXT):** Czas pierwszego zarejestrowanego zdarzenia w danym dniu.

- `last_seen` **(TEXT):** Czas ostatniego zarejestrowanego zdarzenia (służy do monitorowania aktywności honeypota).

4. Opis implementacji

4.1. Obsługa typów wyliczeniowych (Enum)

W celu poprawy czytelności kodu i uniknięcia błędów typu "magic strings", wprowadzono enum TrafficDirection do określania kierunku ruchu. Zapewnia to spójność danych w bazie.

```
Python

class TrafficDirection(str, Enum):
    CLIENT_TO_SERVER = "clientToServer"
    SERVER_TO_CLIENT = "serverToClient"
```

4.2. Mechanizmy detekcji (Heurystyka)

System analizuje ruch w dwóch kierunkach:

A. Kierunek Client -> Server (`clientToServer`):

- **SQL Injection:** Wykrywanie słów kluczowych takich jak `UNION SELECT`, `DROP TABLE`.
- **Remote Code Execution (RCE):** Wykrywanie prób wywołania powłoki (`/bin/sh`, `powershell`, `cmd.exe`).
- **Przesyłanie plików binarnych:** Detekcja nagłówków plików wykonywalnych Windows (tzw. *MZ header*).
- **Analiza UTF-16:** Próba ekstrakcji nazw użytkowników z pakietów binarnych SMB.

B. Kierunek Server -> Client (`serverToClient`):

- **Weryfikacja ataku:** Sprawdzanie, czy serwer zwrócił błąd bazy danych (potwierdzenie SQLi).
- **Detekcja Brute Force:** Analiza kodów błędów SMB. Wykrycie sekwencji bajtów `0xC000006D` (STATUS_LOGON_FAILURE) zlicza nieudane logowania. Przekroczenie progu (5 prób) klasyfikuje zdarzenie jako `brute_force`.

4.3. Asynchroniczność i wydajność

Aby operacje zapisu na dysku (SQLite) nie spowalniały ruchu sieciowego, zastosowano wzorzec **Producer-Consumer**:

- Wątek sieciowy wrzuca logi do `asyncio.Queue`.
- Osobny worker (`log_worker`) pobiera logi z kolejki i zapisuje je w bazie w osobnym wątku (`run_in_executor`).

5. Konfiguracja i uruchomienie

System jest przystosowany do konteneryzacji (Docker-friendly) poprzez wykorzystanie zmiennych środowiskowych:

Zmienna	Opis	Wartość domyślna
LISTEN_HOST	Adres nasłuchu proxy	0.0.0.0
LISTEN_PORT	Port nasłuchu	445
TARGET_HOST	Adres serwera podatnego	vuln_smb
DB_PATH	Ścieżka do pliku bazy danych	/app/data/honeypot.db

6. Wyniki Testów i Obserwacje

Podczas weryfikacji działania systemu przeprowadzono serię symulowanych ataków z wykorzystaniem narzędzi systemowych (`netcat`, `nmap`) oraz ręcznego generowania payloadów. Poniżej przedstawiono zaobserwowane zachowanie systemu Honeypot Proxy:

6.1. Wykrywanie SQL Injection

- **Wektor ataku:** Wysłanie ciągu znaków `admin' UNION SELECT 1, password FROM users --` na port 445.
- **Obserwacja:** Moduł DPI (Deep Packet Inspection) przechwycił pakiet w kierunku *Client-to-Server*. Analiza łańcucha znaków wykryła sygnaturę `UNION SELECT`.
- **Rezultat:** System poprawnie zaklasyfikował zdarzenie jako `sql_injection` z pewnością (confidence) 0.95.

6.2. Wykrywanie Command Injection (RCE)

- **Wektor ataku:** Próba wstrzyknięcia komendy systemowej: `GET /index.php?cmd=/bin/sh HTTP/1.1`.
- **Obserwacja:** Analizator wykrył wystąpienie ciągu `/bin/sh` w dekodowanym payloadzie.

- **Rezultat:** Ruch został oznaczony jako krytyczny `command_injection` (confidence 1.0), co świadczy o próbie zdalnego wykonania kodu.

6.3. Wykrywanie Skanowania Portów (Analiza Wolumetryczna)

- **Wektor ataku:** Uruchomienie pętli wysyłającej 25 szybkich połączeń TCP (SYN/ACK) w czasie poniżej 60 sekund.
- **Obserwacja:** Początkowe połączenia były klasyfikowane jako `benign` (łagodne). Po przekroczeniu progu `SCANNING_THRESHOLD` (20 połączeń z jednego IP), system zmienił klasyfikację kolejnych pakietów.
- **Rezultat:** Zdarzenia typu `connection_open` otrzymały klasyfikację `scanning`, a w polu `details` odnotowano licznik połączeń (np. `"count": 21`).

6.4. Wykrywanie Brute Force (Analiza odpowiedzi serwera)

- **Wektor ataku:** Wielokrotne próby logowania z błędnym hasłem.
- **Obserwacja:** Proxy analizowało ruch powrotny (*Server-to-Client*). Wykryto bajty odpowiadające kodowi błędu SMB `STATUS_LOGON_FAILURE`.
- **Rezultat:** Pierwsze 4 błędy zostały oznaczone jako `auth_failed`. Piąta nieudana próba z tego samego IP zmieniła klasyfikację na `brute_force`, potwierdzając aktywny atak słownikowy.

6.5. Ruch Prawidłowy (Negocjacja SMB)

- **Działanie:** Standardowe podłączenie klienta SMB.
- **Obserwacja:** Pakiety binarne niezawierające znanych sygnatur ataków.
- **Rezultat:** Zdarzenia zostały zalogowane jako `unknown` lub `benign` z pewnością 0.0. Jest to pożądane zachowanie, tworzące pełny ślad audytowy (Audit Trail) bez generowania fałszywych alarmów (False Positives).

To są świetne dane! Idealnie pokazują działanie mechanizmu progowego (threshold).

Przygotowałem dla Ciebie **profesjonalną analizę tych konkretnych wyników**, którą możesz wkleić bezpośrednio do swojego sprawozdania lub prezentacji.

7. Analiza Wyników Testów (Na podstawie danych rzeczywistych)

Poniższa sekcja przedstawia analizę logów zebranych podczas testów penetracyjnych przeprowadzonych w dniu 2026-01-09.

7.1. Analiza Detekcji Wolumetrycznej (Skanowanie Portów)

Test polegał na szybkim nawiązaniu serii połączeń TCP (pętla 1-25) z adresu IP **192.168.65.1** (Docker Host Gateway).

Fragment logów z bazy danych:

ID	Timestamp	Klasyfikacja	Licznik (count)	Komentarz Analityczny
28	21:00:51.346	benign	18	Ruch w normie
29	21:00:51.355	benign	19	Ruch w normie (granica)
30	**21:00:51.364**	**scanning**	**20**	**PRZEKROCZENIE PROGU (Alarm)**
31	21:00:51.371	scanning	21	Kontynuacja ataku
32	21:00:51.378	scanning	22	Kontynuacja ataku

Wnioski:

Zarejestrowane logi potwierdzają poprawność działania algorytmu detekcji wolumetrycznej zdefiniowanej w zmiennej SCANNING_THRESHOLD = 20.

- Pakiety o **ID 18-29** zostały zaklasyfikowane jako **benign** (ruch łagodny), ponieważ licznik połączeń w oknie czasowym (60s) wynosił < 20.
- Pakiet o **ID 30** jest punktem przełomowym. W tym momencie licznik osiągnął wartość 20, co natychmiast zmieniło klasyfikację na **scanning**.
- Wszystkie kolejne połączenia (ID 31-37) dziedziczą status ataku, a licznik w polu **details** poprawnie rośnie.

7.2. Podsumowanie Skuteczności Systemu (Statystyka)

Tabela podsumowująca (agregacja po kolumnie **classification**) wykazuje następujący rozkład zdarzeń:

Typ Zdarzenia	Ilość	Interpretacja

sql_injection	**4**	System poprawnie wykrył sygnatury SQL (np. UNION SELECT) w payloadzie.
command_injection	**2**	Wykryto próby wstrzyknięcia komend powłoki (np. /bin/sh).
scanning	**8**	Końcowa faza testu obciążeniowego (połączenia nr 20-27).
benign	**23**	Początkowa faza testu obciążeniowego oraz inne neutralne pakiety negocjacji SMB.

Całkowita skuteczność:

System zarejestrował łącznie 37 zdarzeń. W tym 14 zdarzeń zostało sklasyfikowanych jako zagrożenie (True Positives), co stanowi 37.8% całego przechwyconego ruchu. Pozostały ruch (62.2%) to ruch tła lub faza "rozgrzewki" przed atakiem wolumetrycznym.

8. Wnioski i Podsumowanie

8.1. Stopień Realizacji Celu

Zrealizowany projekt **SMB Honeypot Proxy** spełnił wszystkie założenia funkcjonalne. Udało się stworzyć działający system typu *Man-in-the-Middle*, który transparentnie pośredniczy w ruchu sieciowym, jednocześnie analizując go pod kątem zagrożeń. System skutecznie oddziela atakującego od chronionej infrastruktury (lub celowo wystawionej podatnej usługi), działając jako pierwsza linia detekcji.

8.2. Skuteczność Metod Detekcji

Testy penetracyjne wykazały, że hybrydowe podejście do detekcji jest najbardziej efektywne:

1. **Analiza Sygnaturowa (DPI):** Okazała się niezawodna w wykrywaniu ataków warstwy aplikacji, takich jak **SQL Injection** czy **Command Injection** (RCE). Dekodowanie payloadu w locie pozwoliło na identyfikację konkretnych fraz (np. `UNION SELECT`, `/bin/sh`) z wysokim wskaźnikiem pewności (0.95 - 1.0).

2. **Analiza Behawioralna (Heurystyka):** Mechanizm progowy (`threshold`) okazał się kluczowy do wykrycia **skanowania portów**. Zdarzenia te, same w sobie niegroźne (brak złośliwego payloadu), zostały poprawnie sklasyfikowane jako atak po przekroczeniu progu 20 połączeń w ciągu minuty.
3. **Analiza Kontekstowa:** Śledzenie odpowiedzi serwera (`STATUS_LOGON_FAILURE`) pozwoliło na odróżnienie zwykłej pomyłki użytkownika od ataku **Brute Force**.

8.3. Wydajność Architektury Asynchronicznej

Wykorzystanie biblioteki `asyncio` w języku Python było decyzją kluczową dla wydajności systemu.

- Zastosowanie modelu **non-blocking I/O** pozwoliło na obsługę wielu równoległych połączeń (podczas testów skanowania) bez widocznego wzrostu opóźnień (latency).
- Implementacja wzorca **Producer-Consumer** z wykorzystaniem kolejek (`asyncio.Queue`) do zapisu logów wyeliminowała problem "wąskiego gardła" przy operacjach dyskowych (I/O bazy danych nie blokuje I/O sieciowego).

8.4. Bezpieczeństwo i Izolacja (Docker)

Konteneryzacja rozwiązania zapewniła wysoki poziom bezpieczeństwa hosta.

- Nawet w przypadku udanego ataku RCE na podatny kontener (`vuln_smb`), atakujący pozostaje zamknięty wewnątrz izolowanego środowiska Docker, nie mając dostępu do systemu operacyjnego gospodarza (macOS/Linux).
- Mechanizm mapowania wolumenów pozwolił na trwałe przechowywanie dowodów (baza SQLite) nawet w przypadku awarii lub restartu kontenerów.

8.5. Wnioski z Procesu Implementacji

Podczas prac nad projektem zidentyfikowano istotne wyzwania techniczne:

- **Zarządzanie Pętlą Zdarzeń (Event Loop):** W środowisku Docker/Python 3.9+ kluczowe okazało się inicjalizowanie obiektów asynchronicznych (takich jak kolejki) wewnątrz aktywnej pętli (`main`), a nie w przestrzeni globalnej, co zapobiegło błędom typu `RuntimeError`.
- **Protokoły Binarne:** SMB jest protokołem złożonym i gadatliwym. Implementacja pełnego parsera jest trudna, dlatego analiza heurystyczna na poziomie surowych bajtów okazała się kompromisem między dokładnością a wydajnością.

8.6. Możliwości Rozwoju

Projekt stanowi solidną bazę do dalszej rozbudowy. Potencjalne kierunki rozwoju to:

- Implementacja obsługi szyfrowanego ruchu SMBv3 (wymagałoby to zarządzania certyfikatami).

- Integracja z systemami SIEM (np. ELK Stack) poprzez wysyłanie logów w formacie JSON na zewnętrzny serwer.
- Dodanie mechanizmu "Active Response", który automatycznie blokowałby adres IP atakującego na poziomie firewalla po wykryciu ataku o wysokim priorytecie.