1. Wstęp i Cel Projektu

1.1. Opis Systemu

Advanced Honeypot System to zaawansowane narzędzie z zakresu cyberbezpieczeństwa, należące do kategorii **Low-Interaction Honeypot** (Honeypot niskiej interakcji). System emuluje działanie podatnych usług sieciowych w celu zwabienia potencjalnych atakujących, rejestracji ich działań oraz analizy wektorów ataku.

1.2. Cel Projektu

Głównym celem projektu jest stworzenie bezpiecznego środowiska do:

- **Wykrywania intruzów:** Działanie jako "sensor" w sieci wewnętrznej (tzw. Canary).
- **Threat Intelligence:** Zbieranie danych o metodach ataków, używanych hasłach i skanowanych zasobach.
- **Edukacji:** Analiza zachowania botnetów i hakerów w kontrolowanym, izolowanym środowisku.
- **Spowolnienie ataku:** Zasymulowanie prawdziwych odpowiedzi, serwerów świadczących określone usługi w celu zwiększenia autentyczności oraz taktyka „pułapki ze smoły”.

2. Architektura Techniczna

System został zaprojektowany w nowoczesnej architekturze mikroserwisów z wykorzystaniem konteneryzacji Docker, która pozwala na utworzenie dowolnej ilości honeypotów, dzięki temu możemy stworzyć pełną infrastrukturę fałszywych pseudo-serwerów, na dowolnych portach - odpowiednio konfigurując plik docker-compose.yml.

2.1. Komponenty Systemu

- **Honeypot Core (Python 3.9):** Wielowątkowy serwer oparty na gniazdach sieciowych (Sockets). Obsługuje emulację protokołów SSH, FTP, HTTP, Telnet, SMTP, MySQL. Działa jako maszyna stanów, generując fałszywe odpowiedzi zgodne ze specyfikacją RFC.
- **Baza Danych (SQLite + WAL):** Przechowuje logi zdarzeń, payloady ataków oraz metadane sesji. Wykorzystuje tryb Write-Ahead Logging dla zapewnienia wysokiej wydajności przy równoczesnym zapisie i odczycie przez wiele kontenerów.
- **Dashboard Analityczny (Streamlit):** Interfejs webowy do wizualizacji danych w czasie rzeczywistym. Zawiera statystyki ataków oraz szczegółowe logi.

- **Moduł Threat Intelligence:** Klasyfikuje ataki (np. SQL Injection, Brute Force).

2.2. Przykładowe Fragmenty Kodu

Poniżej przedstawiono kod odpowiedzialny za symulację interakcji klasy TelnetHandler odpowiedzialny za symulację fałszywej powłoki Linuxa (Fake Shell). Jest to przykład implementacji "Low-Interaction", gdzie komendy są tylko parsowane tekstowo, a nie wykonywane w systemie.

Plik: connection_handler.py (Fake Shell)

Tabela 1

```
class TelnetHandler(ConnectionHandler):
   def handle(self):
      try:
         banner = self.service_config.get('banner', 'Ubuntu 20.04 LTS\r\nlogin: ')
         self.client_socket.send(banner.encode())

         session_data = {'session_id': self.session_id}
         self.log_activity("TELNET_CONNECTION", session_data=session_data)

         username = None
         password = None

         while not username or not password:
            data = self.client_socket.recv(1024)
            if not data: return

            decoded = data.decode('utf-8', errors='ignore').strip()
            if not decoded: continue

            if not username:
               username = decoded
               session_data['username'] = username
               self.log_activity(f"Username: {decoded}", session_data=session_data)
               self.client_socket.send(b"Password: ")
            elif not password:
               password = decoded
               session_data['password'] = password
               self.log_activity(f"Password: {decoded}", session_data=session_data)

               if username == "admin" and password == "admin":
                  welcome_msg = (
                     b"\r\nWelcome to Ubuntu 20.04.3 LTS (GNU/Linux 5.4.0-88-
generic x86_64)\r\n\r\n"
                     b" * Documentation:  https://help.ubuntu.com\r\n"
                     b" * Management:     https://landscape.canonical.com\r\n"
                     b" * Support:        https://ubuntu.com/advantage\r\n\r\n"
                     b"Last login: " +
datetime.now().strftime("%a %b %d %H:%M:%S %Y").encode() + b" from
```

```
192.168.1.10\r\n"
                    )
                    self.client_socket.send(welcome_msg)
                    self._fake_shell(session_data)
                    return
                else:
                    self.client_socket.send(b"\r\nLogin incorrect\r\nlogin: ")
                    username = None
                    password = None

        except Exception as e:
            logging.error(f"Błąd w TelnetHandler: {e}")
        finally:
            self.client_socket.close()
```

Kod odpowiedzialny za symulację interakcji serwera FTP klasy FTPHandler odpowiedzialny za celowe opóźnienie odpowiedzi serwera FTP o kilka sekund. Dla człowieka to mała niedogodność jednak dla bota, próbującego zgadnąć hasło, oznacza spowolnienie ataku z tysiecy prób na minute do zaledwie kilkunastu, poprzez wywoływanie np. time.sleep(1.0)

Tabela 2

```
class FTPHandler(ConnectionHandler):
    def handle(self):
        try:
            banner = self.service_config.get('banner', '220 FTP Server ready.\r\n')
            self.client_socket.send(banner.encode())

            session_data = {'session_id': self.session_id}
            self.log_activity("FTP_CONNECTION", session_data=session_data)

            username = None
            authenticated = False

            while True:
                data = self.client_socket.recv(1024)
                if not data:
                    break

                decoded = data.decode('utf-8', errors='ignore').strip()
                if not decoded:
                    continue

                command = decoded.upper().split()[0] if decoded else ""

                if command == "USER":
                    username = decoded.split(maxsplit=1)[1] if len(decoded.split()) > 1 else
```

```
"unknown"
                session_data['username'] = username
                self.log_activity(decoded, session_data=session_data)


                time.sleep(1.0)
                self.client_socket.send(b"331 Password required.\r\n")

            elif command == "PASS":
                password = decoded.split(maxsplit=1)[1] if len(decoded.split()) > 1 else
"unknown"
                session_data['password'] = password
                self.log_activity(decoded, session_data=session_data)

                time.sleep(2.0)

                if username == "admin" and password == "admin":
                    authenticated = True
                    self.client_socket.send(b"230 User logged in.\r\n")
                else:
                    self.client_socket.send(b"530 Login incorrect.\r\n")

            elif command == "SYST":
                self.client_socket.send(b"215 UNIX Type: L8\r\n")

            elif command == "PWD":
                if authenticated:
                    self.client_socket.send(b"257 \"/\" is the current directory\r\n")
                else:
                    self.client_socket.send(b"530 Please login with USER and PASS.\r\n")

            elif command == "TYPE":
                self.client_socket.send(b"200 Type set to I.\r\n")

            elif command == "PASV":
                self.client_socket.send(b"227 Entering Passive Mode
(127,0,0,1,200,100).\r\n")

            elif command == "LIST":
                if authenticated:
                    self.log_activity("LIST command received (Fake Success)",
session_data=session_data)
                    self.client_socket.send(b"150 Opening ASCII mode data connection
for file list\r\n")
                else:
                    self.client_socket.send(b"530 Please login with USER and PASS.\r\n")

            elif command == "QUIT":
                self.log_activity(decoded, session_data=session_data)
                self.client_socket.send(b"221 Goodbye.\r\n")
```

```
                break

            else:
                self.log_activity(decoded, session_data=session_data)
                self.client_socket.send(b"502 Command not implemented.\r\n")

    except Exception as e:
        logging.error(f"Błąd w FTPHandler: {e}")
    finally:
        self.client_socket.close()
```

3. Funkcjonalności i Emulowane Usługi

System nasłuchuje na następujących portach (konfigurowalnych w docker-compose.yml). Aplikacja na potrzeby demonstracji nasługuje na wysokich portach (powyżej 1000), aby nie były potrzebne uprawnienia administratora do jej uruchomienia poprzez środowisko skonteneryzowane, co generuje problemy konfiguracji, na ten moment zakładamy:

Tabela 3

Usługa	Port	Opis Funkcjonalności
SSH	2222	Emulacja bannera OpenSSH. Rejestruje próby logowania (login/hasło).
HTTP	8080	Emulacja serwera Apache. Wykrywa skanowanie paneli admina i ataki SQL Injection w URL.

FTP	2121	Pozwala na logowanie (admin/admin), ale blokuje kanał przesyłu danych (Data Channel), uniemożliwiając wgranie malware'u.
Telnet	2323	Pełna interakcja "Fake Shell". Symuluje powłokę Linuxa (Ubuntu), pozwala na wpisywanie komend (ls, whoami).
SMTP	2525	Emulacja serwera pocztowego. Przyjmuje maile, pozwalając na przechwycenie treści spamu i phishingu.
MySQL	3306	Emulacja handshake'u bazy danych. Rejestruje próby autoryzacji.

Istnieje możliwośc uruchomienia na portach rzeczywistych poprzez odpowiednie zmapowanie w pliku docker-compose.yml np.

```
ports:
  - "22:2222"   # Haker uderza w 22 (SSH), trafia do 2222 w kontenerze
  - "80:8080"   # Haker uderza w 80 (HTTP), trafia do 8080 w kontenerze
  - "21:2121"   # Haker uderza w 21 (FTP), trafia do 2121 w kontenerze
```

Zamiast dotychczasowych:

```
ports:
  - "2222:2222"   # SSH
  - "8080:8080"   # HTTP
  - "2121:2121"   # FTP
```

3.1. Mechanizm Decepcji i Spowalniania Ataku

Kluczową cechą systemu jest nie tylko wykrywanie, ale i aktywne angażowanie atakującego (Deception Technology). System został zaprojektowany tak, aby **zawsze odpowiadać** na zapytania sieciowe w sposób przewidywalny dla protokołu, co ma na celu:

- **Wydłużenie czasu ataku:** Każda sekunda, którą bot lub haker spędza na interakcji z fałszywą powłoką (np. w Telnecie) lub próbie złamania hasła, to czas, w którym nie atakuje on rzeczywistej infrastruktury, zastosowanie metod wydłużających np. Time.sleep().
- **Zwiększenie kosztu ataku:** Zmuszamy atakującego do analizy fałszywych danych (np. pliku secret_passwords.txt, który w rzeczywistości nie istnieje), marnując jego zasoby.
- **Zmylenie przeciwnika:** Poprzez emulację różnych usług (SSH, MySQL, WWW) na jednym adresie IP, tworzymy obraz "łatwego celu", odciągając uwagę od prawdziwych, lepiej zabezpieczonych serwerów.

4. Symulator Ataków (attacker.py)

W celu przetestowania systemu bez konieczności oczekiwania na rzeczywiste ataki z Internetu, stworzono moduł attacker.py. Jest to skrypt automatyzujący proces testowania, który symuluje zachowanie botnetu.

Skrypt wykonuje losowe ataki na zdefiniowane usługi, wykorzystując słowniki popularnych loginów i haseł oraz znane payloady (np. XSS, SQL Injection).

Tabela 4

```
TARGET_IP = "127.0.0.1"
SSH_PORT = 2222
HTTP_PORT = 8080
FTP_PORT = 2121
TELNET_PORT = 2323

USERNAMES = ["admin", "root", "user", "support", "oracle", "test", "ftp",
"anonymous"]
PASSWORDS = ["123456", "password", "admin123", "qwerty", "letmein", "toor",
"ftp", "anonymous"]

def attack_ssh():
    user = random.choice(USERNAMES)
    pwd = random.choice(PASSWORDS)
    print(f"[SSH] Atak: {user}:{pwd}...", end=" ")

    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.settimeout(3)
        s.connect((TARGET_IP, SSH_PORT))

        s.recv(1024)

        payload = f"{user}\n{pwd}"
        s.send(payload.encode())
        s.close()
        print("Wysłano.")
    except Exception as e:
        print(f"Błąd: {e}")

def attack_http():
    paths = ["/admin", "/login", "/wp-admin", "/config.php", "/",
             "/?id=1' OR '1'='1", "/index.php?page=../../../etc/passwd",
             "/?name=<script>alert('XSS')</script>"]
    path = random.choice(paths)
    url = f"http://{TARGET_IP}:{HTTP_PORT}{path}"
    print(f"[HTTP] Skanowanie: {url}...", end=" ")

    try:
        with urllib.request.urlopen(url, timeout=3) as response:
            pass
        print("Sukces (200 OK).")
```

```
    except Exception as e:
        print(f"OK (honeypot odpowiedział)")

def attack_ftp():
    user = random.choice(USERNAMES)
    pwd = random.choice(PASSWORDS)
    print(f"[FTP] Atak: {user}:{pwd}...", end=" ")

    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.settimeout(3)
        s.connect((TARGET_IP, FTP_PORT))

        response = s.recv(1024).decode('utf-8', errors='ignore')

        s.send(f"USER {user}\r\n".encode())
        response = s.recv(1024).decode('utf-8', errors='ignore')

        s.send(f"PASS {pwd}\r\n".encode())
        response = s.recv(1024).decode('utf-8', errors='ignore')

        commands = ["SYST\r\n", "PWD\r\n", "LIST\r\n", "QUIT\r\n"]
        for cmd in random.sample(commands, 2):
            s.send(cmd.encode())
            time.sleep(0.1)
            try:
                s.recv(1024)
            except:
                pass

        s.close()
        print("Wysłano.")
    except Exception as e:
        print(f"OK (połączono)")

def attack_telnet():
    user = random.choice(USERNAMES)
    pwd = random.choice(PASSWORDS)
    print(f"[TELNET] Atak: {user}:{pwd}...", end=" ")

    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.settimeout(5)
        s.connect((TARGET_IP, TELNET_PORT))

        time.sleep(0.2)
        try:
            s.recv(1024)
        except:
            pass
```

```
        s.send(f"{user}\n".encode())
        time.sleep(0.2)
        try:
            s.recv(1024)
        except:
            pass

        s.send(f"{pwd}\n".encode())
        time.sleep(0.2)
        try:
            s.recv(1024)
        except:
            pass

        commands = ["ls\n", "whoami\n", "pwd\n", "cat /etc/passwd\n", "uname -a\n",
"exit\n"]
        for cmd in random.sample(commands, 3):
            s.send(cmd.encode())
            time.sleep(0.1)
            try:
                s.recv(1024)
            except:
                pass

        s.close()
        print("Wysłano.")
    except Exception as e:
        print(f"OK (połączono)")

if __name__ == "__main__":
    print("=" * 60)
    print(" SYMULACJA ATAKÓW NA HONEYPOT")
    print("=" * 60)
    print(f"Target: {TARGET_IP}")
    print(f"Porty: SSH:{SSH_PORT}, HTTP:{HTTP_PORT}, FTP:{FTP_PORT},
Telnet:{TELNET_PORT}")
    print("Naciśnij Ctrl+C, aby przerwać.")
    print()

    attack_functions = [attack_ssh, attack_http, attack_ftp, attack_telnet]

    try:
        for i in range(30):
            attack_func = random.choice(attack_functions)
            attack_func()

            time.sleep(random.uniform(0.3, 1.0))

    except KeyboardInterrupt:
```

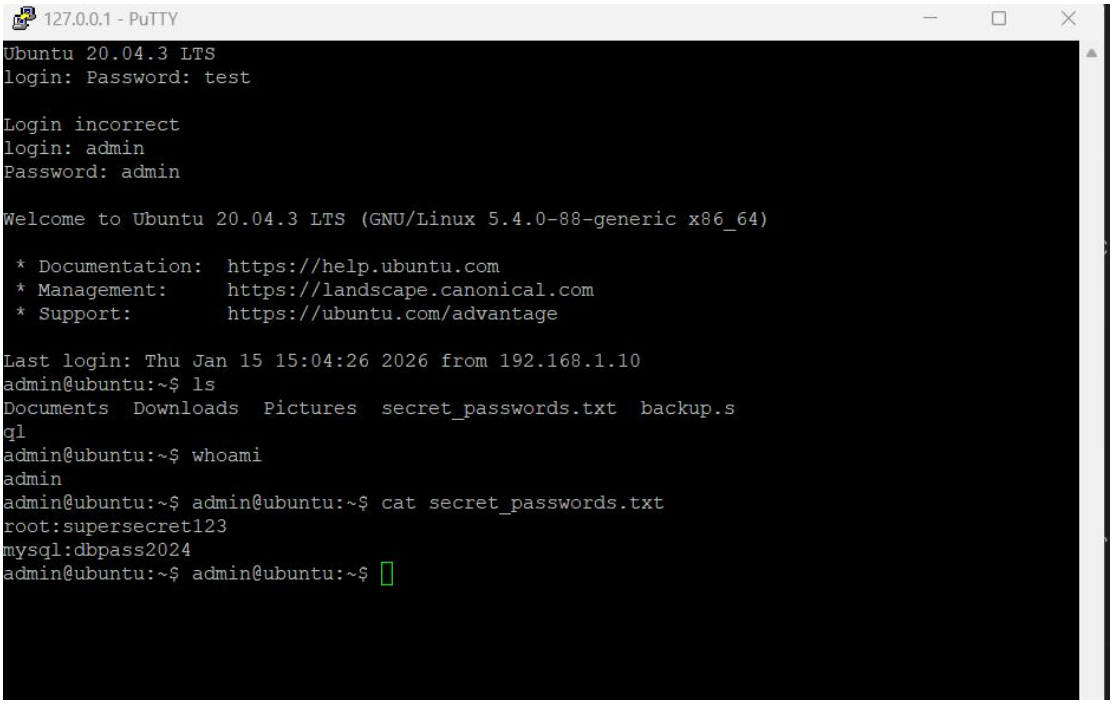
```
    print("\n")
    print("=" * 60)
    print("      Symulacja zatrzymana")
    print("=" * 60)

print()
print("✓ Zakończono symulację ataków")
print("    Sprawdź dashboard: http://localhost:8501")
```

5. Scenariusze Testowe i Wyniki

5.1. Scenariusz 1: Symulacja "Fake Shell" (Telnet)

Cel: Wykazanie, że system potrafi utrzymać interakcję z atakującym, symulując działanie systemu operacyjnego.

- **Narzędzie:** PuTTY
- **Akcja:** Logowanie (admin/admin) i eksploracja systemu plików.



Rysunek 1. Konsola Putty, przeprowadzona próba emulacji systemu ubuntu i typowych komend.

Wynik w Dashboardzie: System poprawnie zarejestrował całą sesję, w tym każdą wpisaną komendę.

Szczegółowe Logi

Wyszukaj po IP

Pokaż wierszy

50

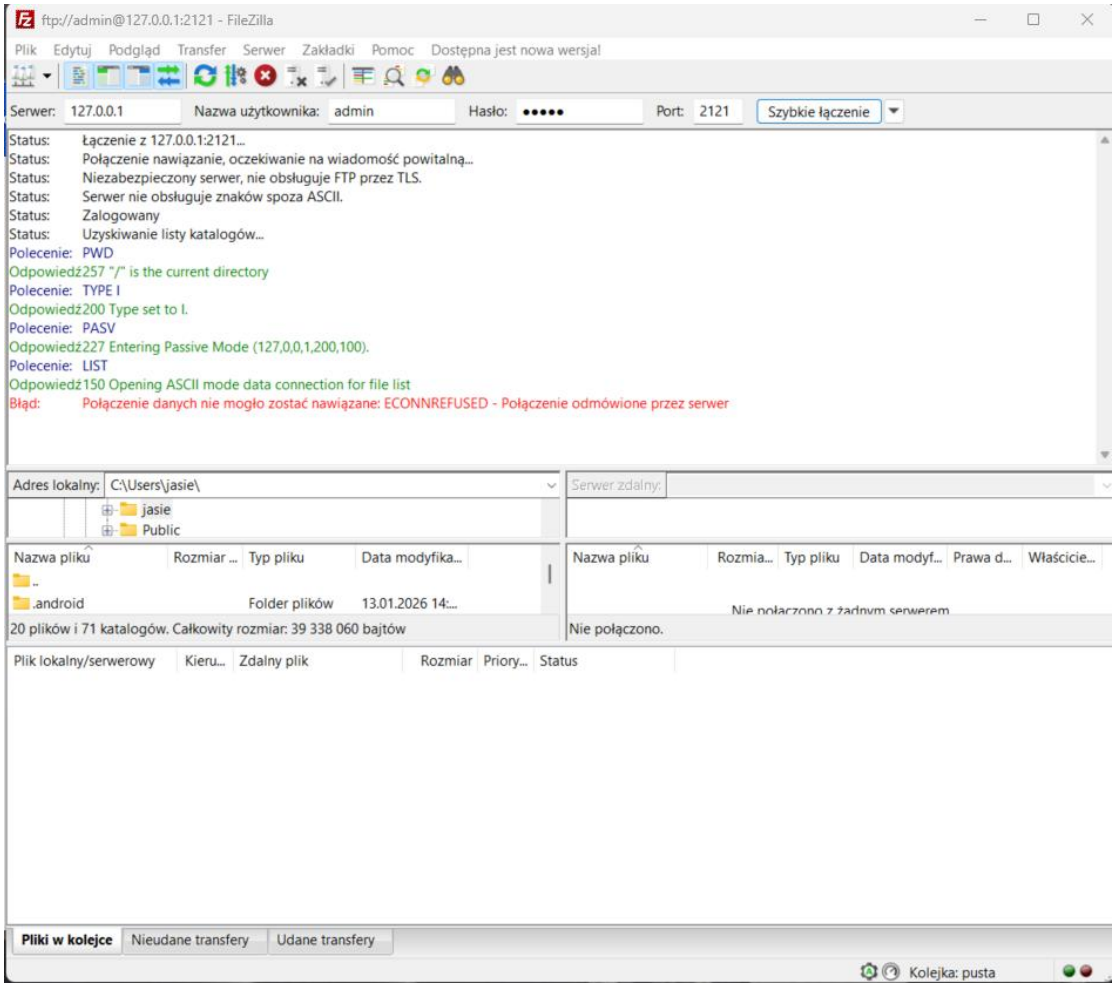
	timestamp	service_name	source_ip	country	threat_level	payload
0	2026-01-15 15:04:26	Fake Telnet	172.19.0.1	None	1	Password: admin
1	2026-01-15 15:04:24	Fake Telnet	172.19.0.1	None	1	Username: admin
2	2026-01-15 15:04:22	Fake Telnet	172.19.0.1	None	1	Password: test
3	2026-01-15 15:04:19	Fake Telnet	172.19.0.1	None	1	Username: ▯'▯▯▯
4	2026-01-15 15:04:18	Fake Telnet	172.19.0.1	None	1	TELNET_CONNECTION
5	2026-01-15 15:04:00	Fake Telnet	172.19.0.1	None	1	Username: admin
6	2026-01-15 15:03:58	Fake Telnet	172.19.0.1	None	1	Password: .
7	2026-01-15 15:02:58	Fake Telnet	172.19.0.1	None	1	Username: ▯'▯▯▯
8	2026-01-15 15:02:57	Fake Telnet	172.19.0.1	None	1	TELNET_CONNECTION
9	2026-01-15 15:01:55	Fake Telnet	172.19.0.1	None	1	Password: admin

Rysunek 2.Logi systemu.

5.2. Scenariusz 2: Blokada Wycieku Danych (FTP)

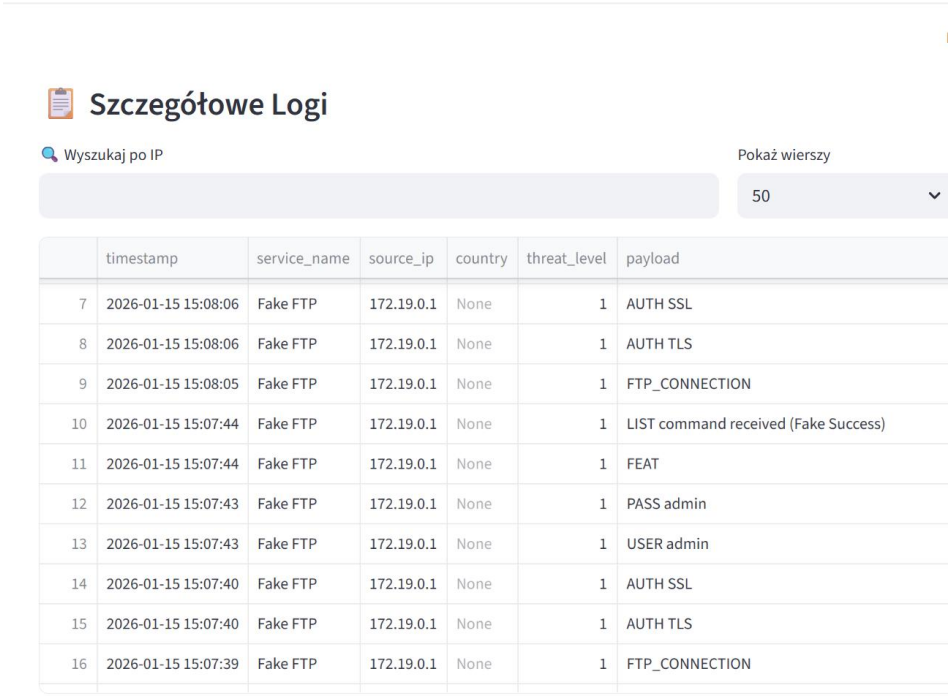
Cel: Wykazanie, że system pozwala na logowanie, ale blokuje transfer plików (Data Channel).

- **Narzędzie:** FileZilla
- **Akcja:** Próba wylistowania plików po zalogowaniu.



Rysunek 3. Ekran programu filezilla, służącego do przeprowadzenia manualnego testu ataku

Dashboard:



	timestamp	service_name	source_ip	country	threat_level	payload
7	2026-01-15 15:08:06	Fake FTP	172.19.0.1	None	1	AUTH SSL
8	2026-01-15 15:08:06	Fake FTP	172.19.0.1	None	1	AUTH TLS
9	2026-01-15 15:08:05	Fake FTP	172.19.0.1	None	1	FTP_CONNECTION
10	2026-01-15 15:07:44	Fake FTP	172.19.0.1	None	1	LIST command received (Fake Success)
11	2026-01-15 15:07:44	Fake FTP	172.19.0.1	None	1	FEAT
12	2026-01-15 15:07:43	Fake FTP	172.19.0.1	None	1	PASS admin
13	2026-01-15 15:07:43	Fake FTP	172.19.0.1	None	1	USER admin
14	2026-01-15 15:07:40	Fake FTP	172.19.0.1	None	1	AUTH SSL
15	2026-01-15 15:07:40	Fake FTP	172.19.0.1	None	1	AUTH TLS
16	2026-01-15 15:07:39	Fake FTP	172.19.0.1	None	1	FTP_CONNECTION

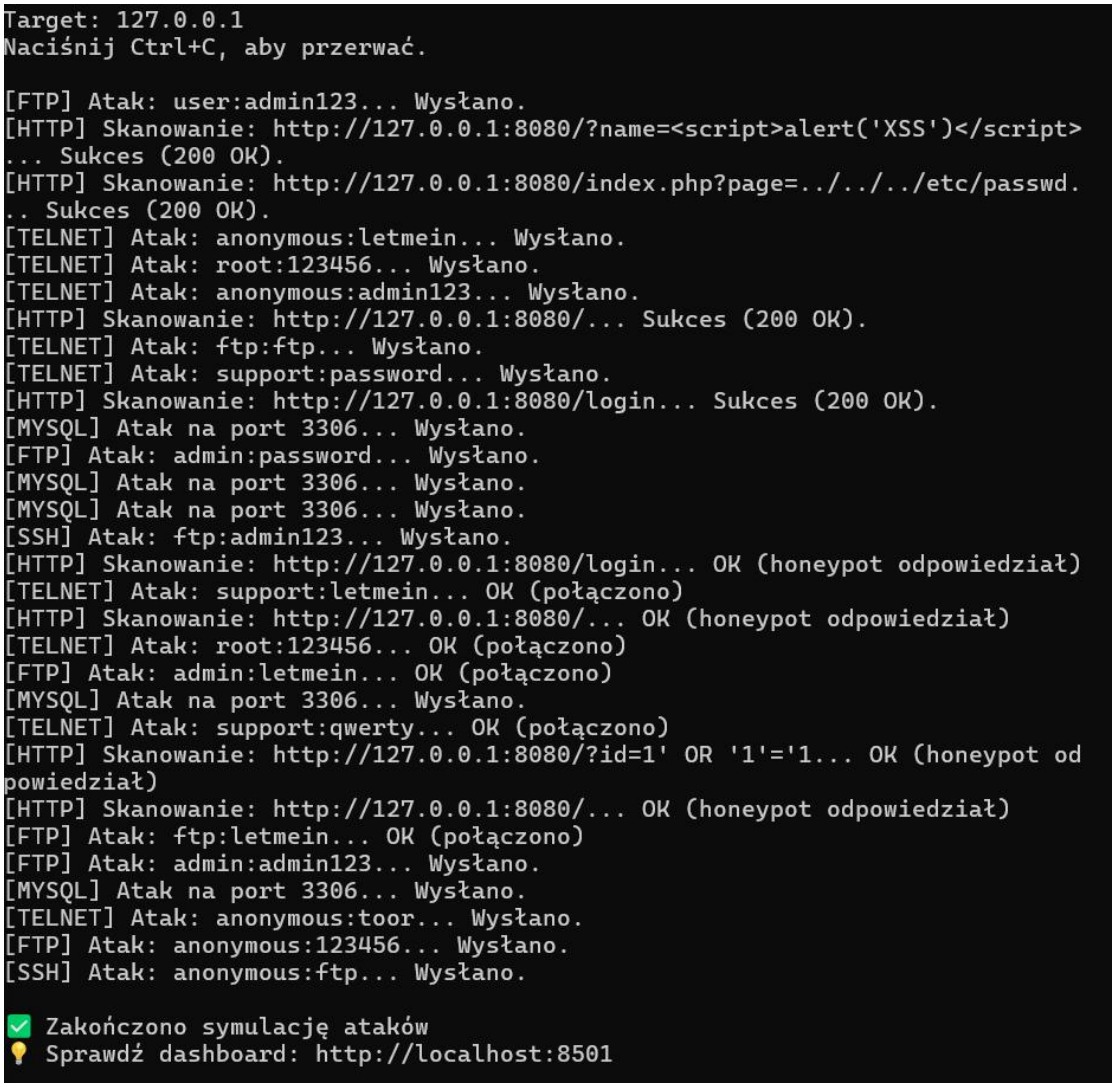
Rysunek 4 Dashboard

Klient musiał odczekać wyznaczony czas aby zostac poinformawanym o pomyslnym zalogowaniu, jednak w momencie uruchomienia procedur zwiazanych z pobraniem katalogów, przestaje być obsługiwany.

5.3. Scenariusz 3: Reakcja na przygotowane testy w pliku attacker.py

Cel: Wykazanie skuteczności modułu Threat Intelligence w klasyfikacji różnorodnych typów ataku generowanych automatycznie.

- **Narzędzie:** Skrypt attacker.py (uruchamiany w terminalu)
- **Akcja:** Uruchomienie symulacji botnetu, który wykonuje serię ataków na usługi: SSH, HTTP, FTP oraz MySQL. Nie wszystkie requesty mają mieć przewidzianą pozytywną odpowiedź ponieważ zawierają np. błędne hasło.



Rysunek 5. Konsola programu attacker.py

Wynik w Dashboardzie: System w czasie rzeczywistym odbiera zdarzenia i automatycznie przypisuje im odpowiednie kategorie zagrożeń.

Przetestowanie kolejnych usług (http) polega na dokładnie tym samym, czyli zgłoszeniu próby połączenia na port który nie powinien otrzymywać połączeń.

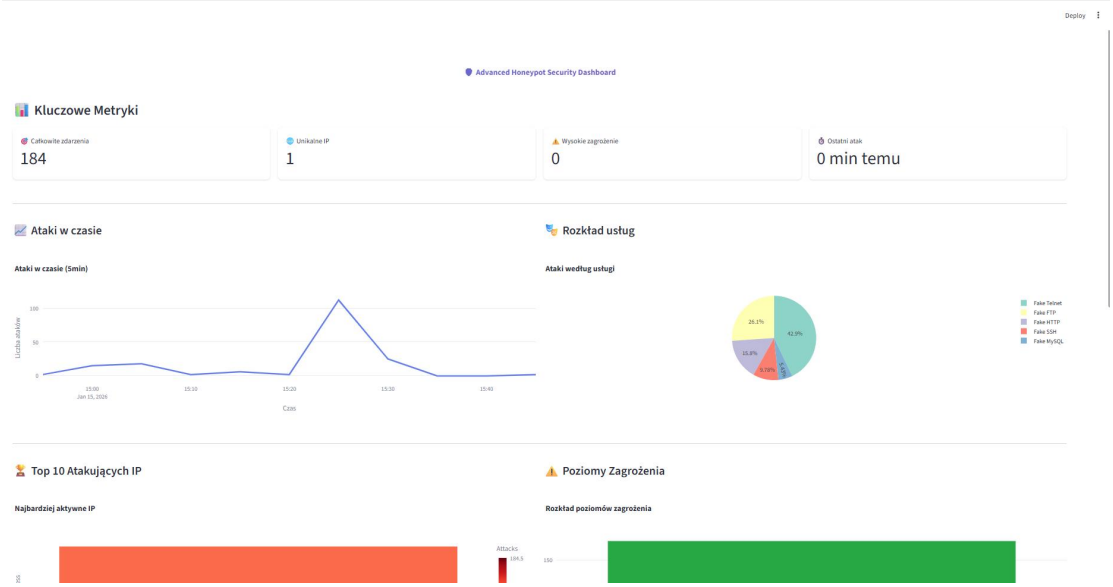
Szczegółowe Logi

Wyszukaj po IP

Pokaż wierszy 50

	timestamp	service_name	source_ip	country	threat_level	payload
0	2026-01-15 15:30:40	Fake SSH	172.19.0.1	None	1	anonymous ftp
1	2026-01-15 15:30:39	Fake SSH	172.19.0.1	None	1	CONNECTION_ATTEMPT
2	2026-01-15 15:30:36	Fake FTP	172.19.0.1	None	1	PASS 123456
3	2026-01-15 15:30:34	Fake FTP	172.19.0.1	None	1	USER anonymous
4	2026-01-15 15:30:33	Fake FTP	172.19.0.1	None	1	FTP_CONNECTION
5	2026-01-15 15:30:33	Fake Telnet	172.19.0.1	None	1	Password: toor
6	2026-01-15 15:30:32	Fake Telnet	172.19.0.1	None	1	Username: anonymous
7	2026-01-15 15:30:32	Fake Telnet	172.19.0.1	None	1	TELNET_CONNECTION
8	2026-01-15 15:30:32	Fake MySQL	172.19.0.1	None	1	MySQL auth attempt: [redacted]
9	2026-01-15 15:30:31	Fake MySQL	172.19.0.1	None	1	MYSQL_CONNECTION

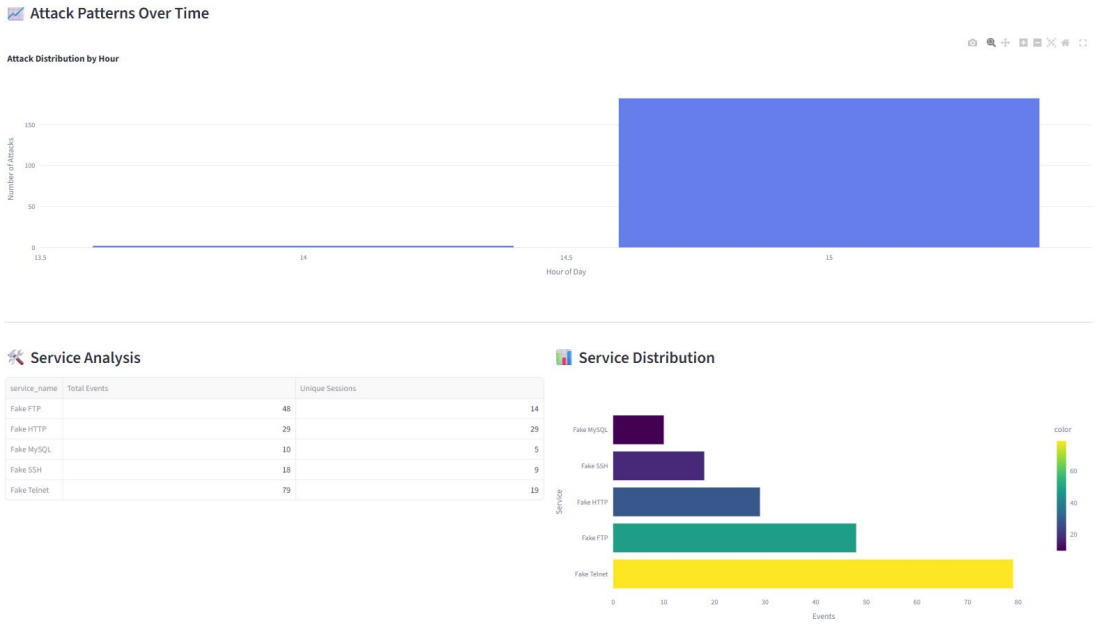
Rysunek 6 Logi systemu



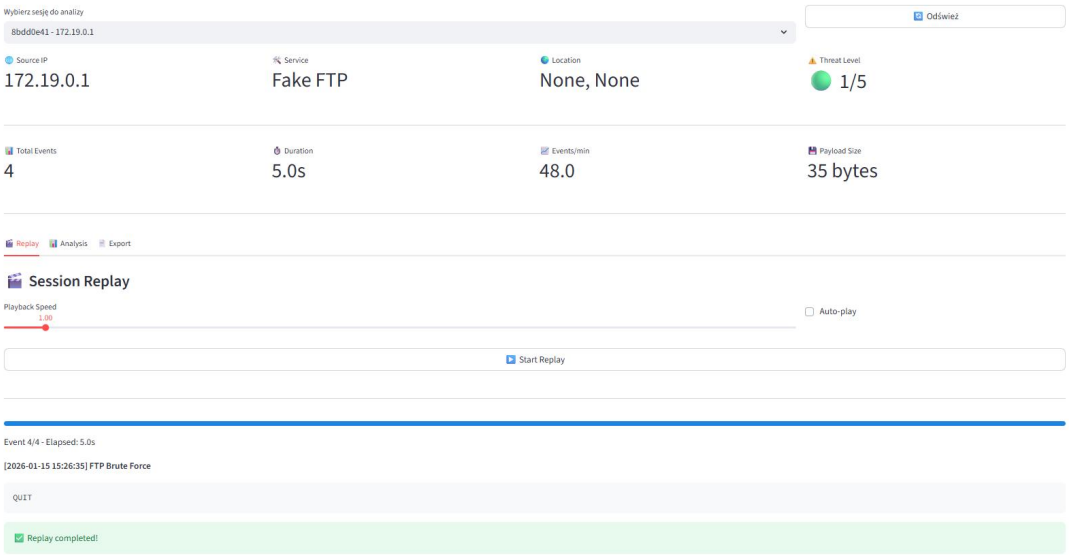
Rysunek 7. Ogólne dane ataków.

6. Dodatkowe funkcjonalności.

System przedstawia statystyki o atakach w różnej formie, dodatkowo umożliwia przeprowadzenie symulacji powtórki ataku aby zobrazować w jakiej kolejności i odstępach czasu atak był wykonywany (w celu wykrycia np. czy atakującym był człowiek czy bot).



Rysunek 8 Ekran statystyk w formie wykresów i tabel.



Rysunek 9. Prosty symulator przeprowadzonych ataków.

7. Podsumowanie

Zrealizowany projekt to funkcjonalny, skalowalny i bezpieczny system pułapki. Dzięki zastosowaniu Dockera jest łatwy we wdrożeniu na dowolnym serwerze VPS. Architektura "Low-Interaction" minimalizuje ryzyko przejęcia systemu przez hakera,

jednocześnie dostarczając wartościowych danych analitycznych niezbędnych w procesie Incident Response.

8. Link do repozytorium z projektem.

https://github.com/jantom38/honeypot_doc