Miłosz Kutyła, Jakub Ossowski,
Jan Walczak, Patryk Jankowicz

Politechnika Warszawska

Sprawozdanie z implementacji projektu bezpiecznej architektury pt. Nowa aplikacja webowa

28 grudnia 2023

Spis treści

1. Wstęp

1.1. Scenariusz

Firma, dla której pracuje zespół WOJK, poprosiła nas o pomoc z zabezpieczeniem nowej aplikacji www. Aplikacja składa się z:

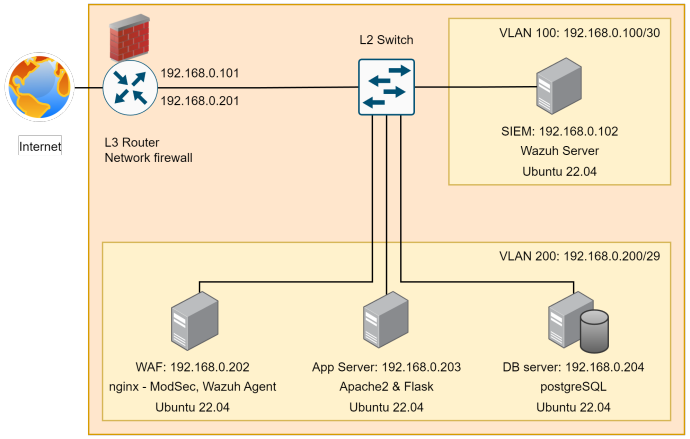
- serwera aplikacyjnego, który serwuje usługę,
- bazy danych, która gromadzi dane powiązane z działaniem aplikacji.

Zadaniem zespołu jest przedstawienie architektury sieciowej, która będzie uwzględniała najlepsze praktyki bezpieczeństwa m.in.:

- zastosowanie szyfrowanego połączenia tam, gdzie jest to wymagane,
- zastosowanie filtrowania ruchu w kontekście zdefiniowanego ruchu,
- zastosowania weryfikacji bezpieczeństwa danych użytkowników wprowadzanych do aplikacji,
- zbieranie logów z nieprawidłowych zdarzeń w sieci,
- odpowiednie przygotowanie maszyn, na których będzie działać aplikacja www.

1.2. Docelowa architektura

W ramach części projektowej utworzyliśmy schemat architektury docelowej sieci uwzględniając kwestie bezpieczeństwa związane z separacją poszczególnych komponentów usługi www oraz kwestie bezpieczeństwa związane z siecią i serwerami. Architekturę sieci na schemacie niskopoziomowym przedstawia rysunek 1.



Rysunek 1: Sugerowana topologia sieci: schemat niskopoziomowy

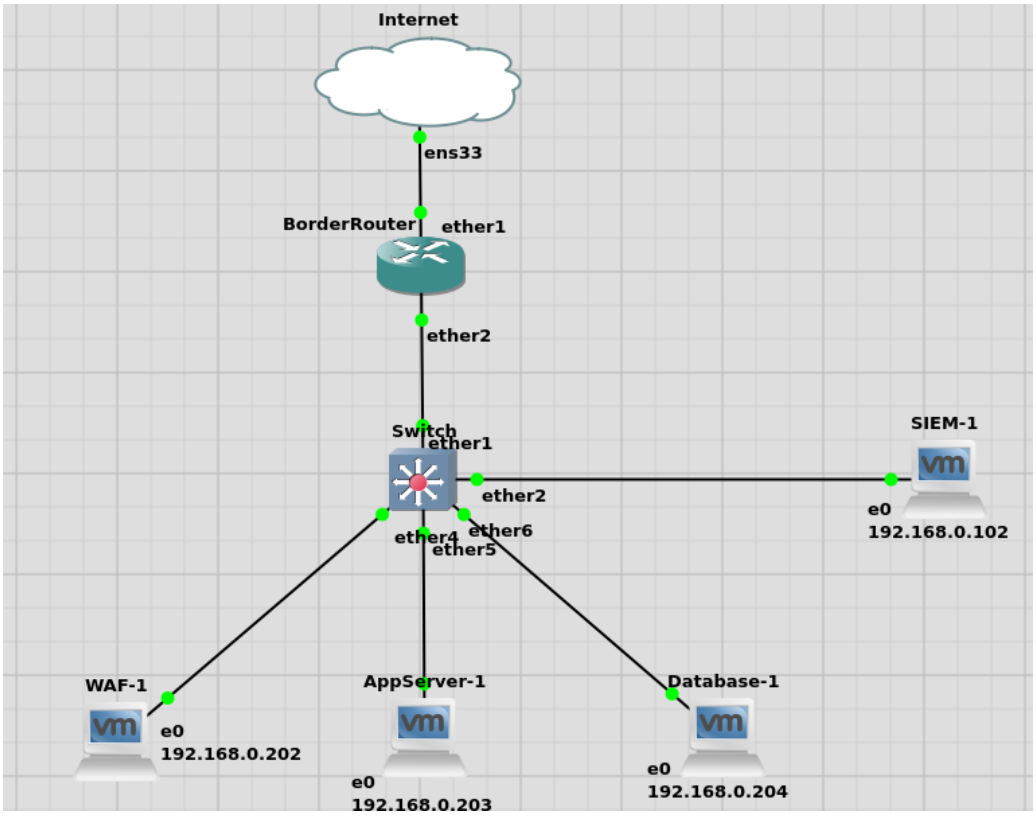
1.3. Wymagania – laboratorium

W ramach laboratorium należy wdrożyć stworzony projekt. Wybór aplikacji www pozostaje w gestii wykonujących, przy czym wybrana usługa musi rozdzielać warstwę aplikacyjną od warstwy danych (nie jest dopuszczalne umieszczenie dwóch tych warstw na jednym hoście). W szczególności wdrożenie musi obejmować następujące wymagania bezpieczeństwa:

- Komunikacja z serwerem aplikacyjnym powinna odbywać się po zaszyfrowanym kanale z wykorzystaniem TLS 1.2.
- Należy wdrożyć firewalla aplikacji www w trybie reverse proxy, który będzie:
  - ◇ Blokował wyciek danych w odpowiedzi na zapytanie `GET`.
  - ◇ Logował Command Injection w zapytaniu `POST` (np. wykrywanie ciągu znaków `ifconfig`).
  - ◇ Blokował atak siłowy na mechanizm uwierzytelnienia.
- Informacje o wykrytych atakach z firewalla aplikacji www przesyłał do rozwiązania SIM/SEM/SIEM (np. logowanie zdarzeń w OSSEC).
- Należy zastosować utwardzanie do wszystkich użytych hostów.

2. Topologia sieci

Implementacja topologii sieci pozostała niezmieniona w stosunku do topologi przedstawionej w części projektowej. Całość przygotowana została w emulatorze sieci GNS3. Do emulacji hostów wykorzystane zostały maszyny wirtualne, które po odpowiednim skonfigurowaniu zostały zaimportowane do programu GNS3. Następnie zostały podpięte do infrastruktury sieciowej, w skład której wchodzą router brzegowy i switch firmy MikroTik.



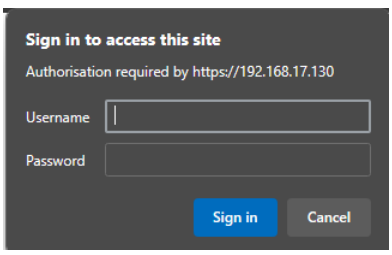
Rysunek 2: Topologia utworzonej sieci w GNS3

2.1. Serwer aplikacyjny

Serwer aplikacyjny, zgodnie z założeniami, został zaimplementowany na maszynie wirtualnej z systemem Ubuntu 22.04 LTS (Jammy Jellyfish). Maszyna w wersji Server (brak interfejsu graficznego) została pobrana ze strony `https://www.linuxvmimages.com`.

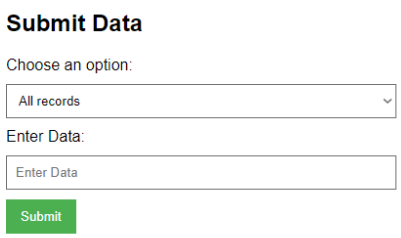
Serwer aplikacyjny został zaimplementowany przy pomocy serwera `apache2` i modułu `Flask` w języku Python. Serwer umożliwia:

- uwierzytelnienie użytkownika, co przedstawia rysunek 3,

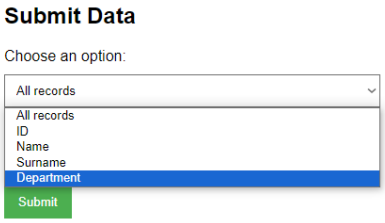


Rysunek 3: Mechanizm uwierzytelniania

- po uwierzytelnieniu, udostępnia stronę tworzenia i wysyłania zapytań do serwera bazodanowego, co przedstawia rysunek 4a i 4b,



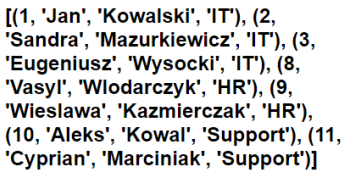
(a) Strona tworzenia i wysyłania zapytań



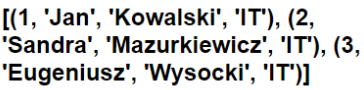
(b) Strona tworzenia i wysyłania zapytań - wybór atrybutu

Rysunek 4: Strona tworzenia i wysyłania zapytań

- po zapytaniu, przekierowuje na stronę, na której widoczne są wyniki utworzonego zapytania, co przedstawia rysunek 5a i 5b. Próba odczytania tej strony bez autoryzacji nie jest możliwa, co przedstawia rysunek 6.

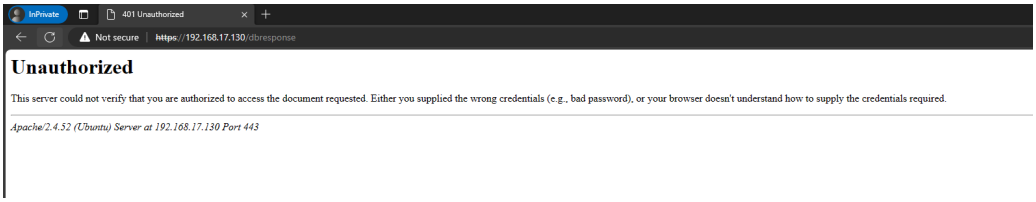


(a) Zwrócenie wszystkich rekordów



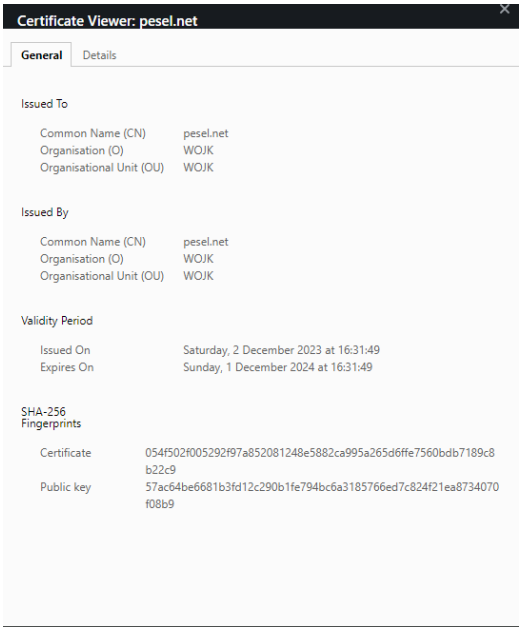
(b) Zwrócenie wszystkich pracowników z działu IT

Rysunek 5: Strona z wynikami zapytania



Rysunek 6: Próba dostępu do strony z odpowiedziami zwracanymi przez bazę danych

- zgodnie z poleceniem szyfrowanie komunikacji protokołem TLS1.2 (certyfikat na rysunku 7.)



Rysunek 7: Ustawiony certyfikat https

Pliki związane z konfiguracją serwera zostały załączone w sekcji 5.1.

2.2. Serwer bazodanowy

Serwer bazodanowy, zgodnie z założeniami, został zaimplementowany na maszynie wirtualnej z systemem Ubuntu 22.04 LTS (Jammy Jellyfish). Maszyna w wersji Server (brak interfejsu graficznego) została pobrana ze strony `https://www.linuxvmimages.com`. Serwer został zaimplementowany z wykorzystaniem usługi PostgreSQL. Udostępnia styk, przy pomocy którego nawiązywane jest połączenie z serwerem aplikacyjnym. Na serwerze umieściliśmy jedną bazę danych - Pracownicy (`employees`). Wypełniliśmy ją danymi testowymi – przedstawia to rysunek 8.

```
test_erp=# SELECT * from employees
test_erp-# ;
 id | first_name |  last_name   | department
----+------------+--------------+------------
  1 | Jan        | Kowalski     | IT
  2 | Sandra     | Mazurkiewicz | IT
  3 | Eugeniusz  | Wysocki      | IT
  8 | Vasyl      | Wlodarczyk   | HR
  9 | Wieslawa   | Kazmierczak  | HR
 10 | Aleks      | Kowal        | Support
 11 | Cyprian    | Marciniak    | Support
(7 rows)
```

Rysunek 8: Utworzona baza danych

2.3. Web Application Firewall

Web Application Firewall został zaimplementowany na maszynie wirtualnej z systemem Ubuntu 22.04 LTS (Jammy Jellyfish). Maszyna w wersji Server (brak interfejsu graficznego) została pobrana z `osboxes.org`.

Do realizacji WAF zastosowaliśmy serwer `nginx` pełniący rolę serwera proxy z modułem `ModSecurity`. Pośredniczy on w ruchu między klientem (hostem z sieci publicznej) i serwerem aplikacyjnym. Komunikacja na styku z siecią publiczną oraz komunikacja pomiędzy WAF'em i serwerem aplikacyjnym jest szyfrowana przy pomocy TLS 1.2.

Konfiguracja reguł:

- Blokowanie wycieku danych w odpowiedzi na zapytanie `GET`.

- Logowanie Command Injection w zapytaniu `POST` (np. wykrywanie ciągu znaków `ifconfig`).
- Blokowanie ataków siłowych na mechanizm uwierzytelnienia.

została przedstawiona na rysunku 9.

```
# Including basic config
Include /etc/nginx/modsec/modsecurity.conf

# --- Data Exfiltration ---
SecRule REQUEST_URI "DbResponse" "id:101, phase:4, deny, status:403, noauditlog, chain"
        SecRule REQUEST_METHOD "^GET$" "chain"
                SecRule RESPONSE_BODY "\[.*[^\s\[\]].*\]" "phase:4"

# --- Command Injection ---
SecRule ARGS "(\b(?:cat|aptitude|chage|df|pwd|ls|uname|ping|mkdir|who|top|echo|passwd|history|mv|kil
l|tail|hostname|nano|rm|lu|su|locate|pstree|sudo|apt|shutdown|dir|cd|reboot|gunzip|iptable|killall|n
etstat|usermod|touch|ssh|telnet|sleep|usermod|bash|sh|ifconfig|curl|wget|cc|echo)\b)" "id:102, phase
:2, auditlog, msg:'Command Injection', logdata:'%{matched_var}', chain"
        SecRule REQUEST_METHOD "^POST$"

# --- Brute force ---

# Tracking ip
SecAction "id:1, phase:1, noauditlog, pass, initcol:ip=%{REMOTE_ADDR}"
# Tracking username requests
SecRule REQUEST_HEADERS:Authorization "([a-zA-Z0-9]+=*)$"  "id:2, phase:1, capture, chain, noauditlo
g"
        SecRule TX:1 ^(\w+): "t:base64Decode,capture, setuid:%{TX.1}, noauditlog"

# IP and user blocking
SecRule ip:bf_block "@gt 0" "id:1031, deny, status:403, noauditlog, msg:'IP address blocked for 10 m
inutes, too many login attempts'"
SecRule user:bf_block "@gt 0" "id:1035, deny, status:403, noauditlog, msg:'Username blocked for 10 m
inutes, too many login attempts'"

#Succesfull login
SecRule RESPONSE_STATUS "^200" "id:1032, phase:5, t:none, noauditlog, pass, setvar:ip.bf_counter=0"
SecRule RESPONSE_STATUS "^200" "id:1036, phase:5, t:none, noauditlog, pass, setvar:user.bf_counter=0
"
```

```
#Failed login
SecRule RESPONSE_STATUS "^401" "id:1033, phase:5, t:none, noauditlog, pass, setvar:ip.bf_counter=+1,
 expirevar:ip.bf_counter=180"
Secrule RESPONSE_STATUS "^401" "id:1037, phase:5, t:none, noauditlog, pass, setvar:user.bf_counter=+
1, expirevar:user.bf_counter=180"

#If tries >5:
SecRule ip:bf_counter "@gt 3" "id:1034, t:none, setvar:ip.bf_block=1, expirevar:ip.bf_block=600, set
var:ip.bf_counter=0, noauditlog"

SecRule user:bf_counter "@gt 3" "id:1038, t:none, setvar:user.bf_block=1, expirevar:user.bf_block=60
0, setvar:user.bf_counter=0, noauditlog"
```

Rysunek 9: Reguły utworzone na WAF

Wyjaśnienie działania poszczególnych reguł:

1. **Data Exfiltration** - reguła w pierwszej kolejności sprawdza czy żądanie zostało wysłane na endpoint `/DbResponse`. Jeśli tak to sprawdza, czy wykorzystana została metoda HTTP GET. Jeśli tak, przechodzi do ostatniej reguły sprawdzającej body odpowiedzi od serwera, dopasowując je do wyrażenia regularnego. Jeśli w body odpowiedzi pomiędzy znakami "[" oraz "]", znajduje się jakikolwiek znak alfanumeryczny, oznacza to, że potencjalnie w odpowiedzi znajdują się jakieś dane. Wówczas taka odpowiedź jest blokowana i klient nie otrzymuje pełnej odpowiedzi, tym samym nie dopuszczając do wycieku danych. Brak odpowiedzi HTTP 403 Forbiden związany jest ze specyfiką działania ModSecurity. Firewall jest w stanie przeanalizować body odpowiedzi dopiero w fazie 4, podczas gdy wcześniej przetworzył już nagłówki w fazie 3. Z tego powodu nie jest już w stanie zmodyfikować nagłówków odpowiedzi i zamienić kodu. Finalnie klient otrzymuje odpowiedź z kodem HTTP 200 OK, ale z pustym body.
2. **Command Injection** - reguła sprawdza wszystkie argumenty zapytania (zarówno te znajdujące się w URI, jak i te znajdujące się w body). Porównuje wydobyte argumenty z wyrażeniem regularnym, które składa się ze słów zawierających najpopularniejsze komendy systemu UNIX. Jeśli regex znajdzie komendę w którymkolwiek z argumentów, wykonuje się kolejna reguła sprawdzająca, czy do zapytania wykorzystana została metoda POST. Jeżeli obie reguły zostały spełnione, zapytanie jest przepuszczane do serwera aplikacyjnego w standardowy sposób. Jednocześnie do logów zapisywana jest anomalia, która zostaje później przekazana do SIEM'a.
3. **Blokowanie Brute force** - zaimplementowany zestaw reguł blokuje zarówno adres IP, z którego zostały wielokrotnie wysłane żądania niepoprawnego uwierzytelniania, jak i konto użytkownika, w przypadku ataku siłowego pochodzącego z wielu adresów IP (np. botnet). W pierwszej kolejności reguła wychwytuje kontekst

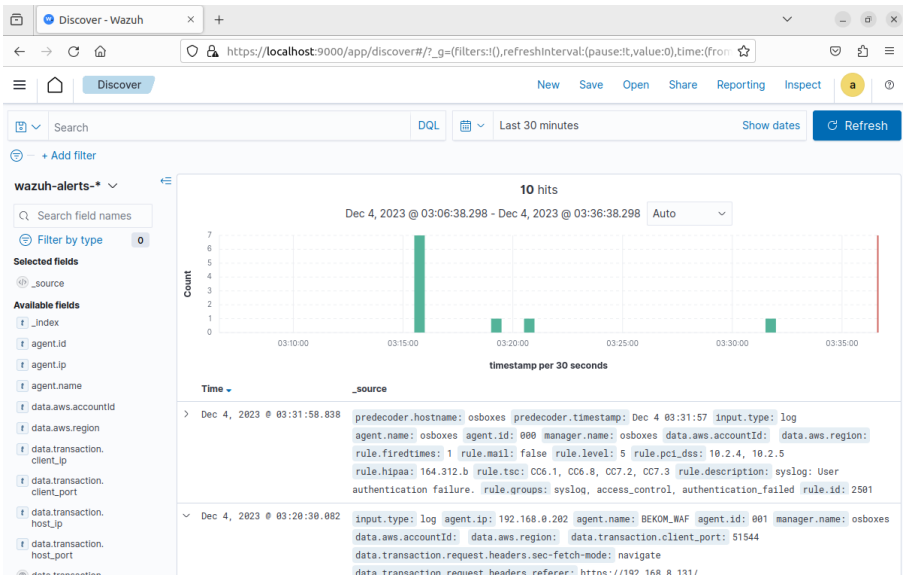
żądania i odwołuje się do odpowiedniej kolekcji. W przypadku adresu IP wartością wyróżniającą daną kolekcję był adres IP (sam w sobie). Natomiast dla nazwy użytkownika musieliśmy ją wyodrębnić z nagłówka `Authorization`, którego wartość jest parą (<nazwa użytkownika>:<hasło>) zakodowaną za pomocą Base64. Kolekcje te są identyfikowane przez podaną nazwę użytkownika. Następnie poszczególne reguły dla obu kolekcji sprawdzają odpowiedź na żądanie. Jeżeli kodem odpowiedzi jest HTTP 401, reguła zwiększa o 1 licznik niepoprawnych uwierzytelnień dla danej kolekcji (dodatkowo wartość licznika ma ważność przez 3 minuty tzn. jeśli w ciągu 3 minut nie zostanie wysłany kolejny request, licznik jest resetowany), natomiast w przypadku kodu HTTP 200 OK, resetuje licznik do 0. W ostatnim kroku reguła dla każdej kolekcji sprawdza, czy licznik niepoprawnych zapytań jest większy od 3 (pierwsze niepoprawne zapytanie ustawia licznik na 0, dozwolone 4 niepoprawne zapytania, po otrzymaniu 5. niepoprawnego, reguła blokuje każde następne). Jeśli licznik jest większy niż 3, reguła resetuje licznik i ustawia flagę blokowania na 10 minut. Stan flagi jest sprawdzany przy każdym zapytaniu i w przypadku podniesionej flagi (stan 1), zapytanie jest blokowane i do klienta zwracany jest kod HTTP 403 Forbidden.

Na serwerze został dodatkowo uruchomiony agent Wazuh, który umożliwił wysyłanie logowanych ataków do SIEMa.

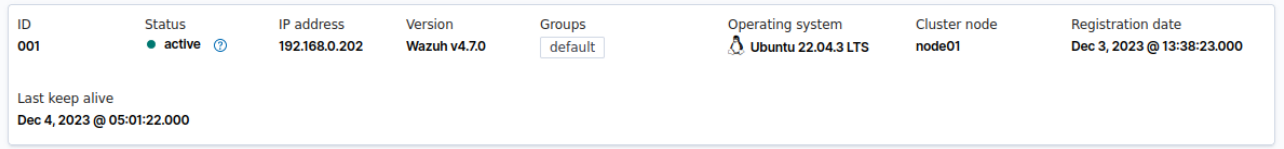
2.4. Rozwiązanie SIEM

SIEM został zaimplementowany na maszynie wirtualnej z systemem Ubuntu 22.04 LTS (Jammy Jellyfish). Maszyna w wersji Desktop (z interfejsem graficznym) została pobrana z `osboxes.org`. Wybraliśmy rozwiązanie desktopowe ze względu na chęć zaimplementowania SIEMa z interfejsem graficznym – Wazuh.

Przyjęte rozwiązanie udostępnia end-point, do którego mogą łączyć się agenci Wazuh. Agenci następnie przesyłają do serwera Wazuh logi ze skonfigurowanych plików. Interfejs graficzny Wazuh służący do analizy zdarzeń przedstawia rysunek 10, a potwierdzający podłączenie się agenta (z WAF) rysunek 11.



Rysunek 10: Webowy interfejs graficzny – Wazuh



Rysunek 11: Przyłączenie agenta – Web Application Firewall

2.5. Network Firewall

Network firewall został skonfigurowany na routerze MikroTik dostępnym w GNS3. Konfigurację firewalla zgodną z założeniami projektowymi przedstawia rysunek 12.

```
Flags: X - disabled, I - invalid; D - dynamic
 0    chain=forward action=accept protocol=tcp dst-address=192.168.0.202
      in-interface=ether1 dst-port=443

 1    chain=forward action=accept protocol=tcp src-address=192.168.0.202
      out-interface=ether1 src-port=443

 2    chain=forward action=accept protocol=tcp src-address=192.168.0.202
      dst-address=192.168.0.102 dst-port=1515

 3    chain=forward action=accept protocol=tcp src-address=192.168.0.202
      dst-address=192.168.0.102 dst-port=1514

 4    chain=forward action=accept protocol=tcp src-address=192.168.0.102
      dst-address=192.168.0.202 dst-port=1515

 5    chain=forward action=accept protocol=tcp src-address=192.168.0.102
      dst-address=192.168.0.202 dst-port=1514

 6    chain=forward action=accept protocol=tcp src-address=192.168.0.102
      dst-address=192.168.0.202 dst-port=36026

 7    chain=forward action=drop
```

Rysunek 12: Konfiguracja reguł firewalla

2.6. Hardening hostów

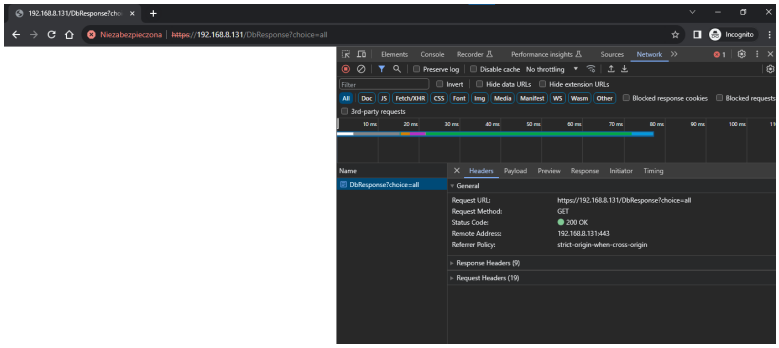
Każdy z hostów (SIEM, WAF, App Server, DB Server) został poddany hardeningowi. Każdy z nich działa na systemie operacyjnym Ubuntu 22.04 LTS (Jammy Jellyfish). W związku z tym, dokumentem referencyjnym w procesie utwardzania był `CIS_Ubuntu_Linux_22.04_LTS_Benchmark_v1.0.0`.

3. Walidacja zgodności z założeniami

Na sam koniec przeprowadziliśmy testy w celu zweryfikowania zgodności naszego rozwiązania z przedstawionymi założeniami, niezwiązanymi z architekturą sieci.

3.1. Blokowanie wycieku danych w odpowiedzi na zapytanie `GET`

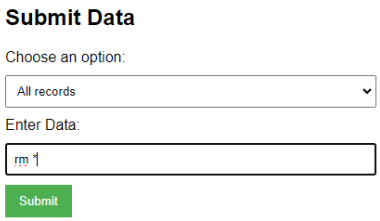
W celu zweryfikowania blokowania wycieku danych w odpowiedzi na zapytanie `GET` przez WAF, wysłaliśmy spreparowane zapytanie `GET` do serwera. Zablokowanie tego ataku w postaci wycięcia odpowiedzi (pusta strona) przedstawia rysunek 13.



Rysunek 13: Blokowanie wycieku danych w odpowiedzi na zapytanie `GET`

3.2. Logowanie Command Injection w zapytaniu `POST`

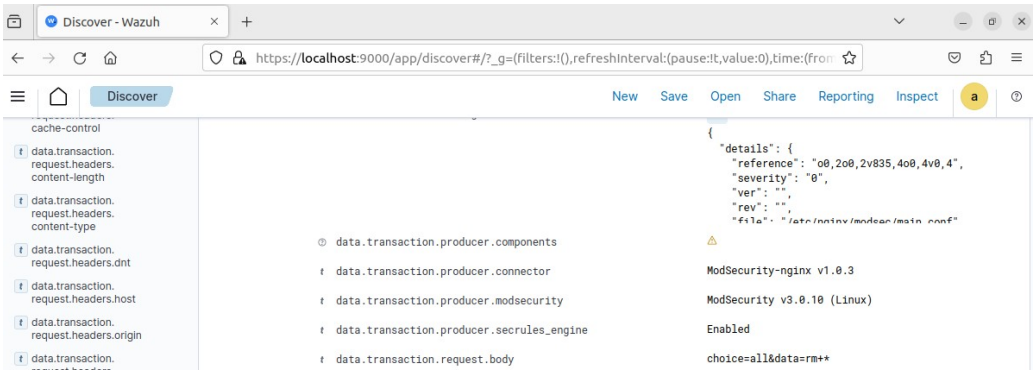
W celu zweryfikowania logowania Command Injection w zapytaniu `POST` przez WAF, przeprowadziliśmy test tego ataku próbując wykonać polecenie `rm *` (rys. 14). Zalogowanie tego zdarzenia zaobserwowaliśmy po stronie WAFa w pliku z logami – odpowiedni wpis przedstawia rysunek 15.



Rysunek 14: Próba przeprowadzenia ataku

Rysunek 15: WAF: Logowanie Command Injection w zapytaniu `POST`

Zaobserwowaliśmy również poprawne przekazanie logów do SIEMa. Zalogowanie zdarzenia z poziomu Wazuha przedstawia rysunek 16.



Rysunek 16: Przekazanie wykrytego ataku do SIEM

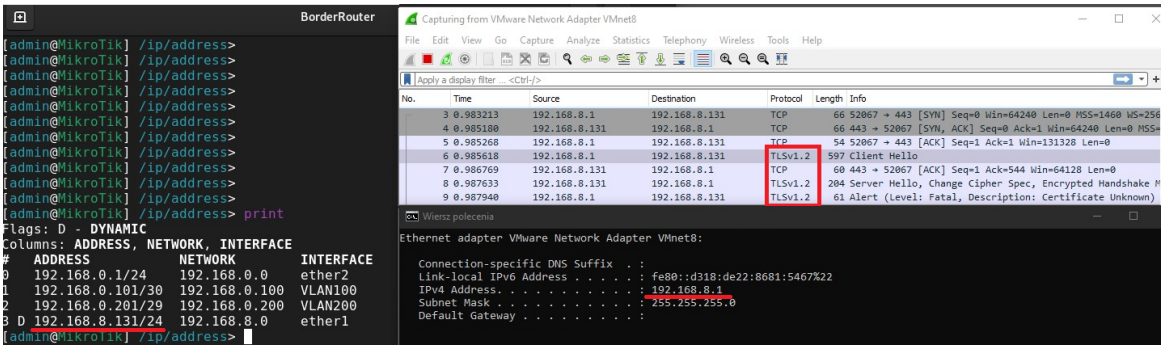
3.3. Blokowanie ataku siłowego na mechanizm uwierzytelnienia

W celu zweryfikowania blokowania ataku siłowego na mechanizm uwierzytelnienia 5-krotnie wprowadziliśmy błędne dane uwierzytelniające. Po 5. próbie dostęp do mechanizmu uwierzytelniania został zablokowany, co przedstawia rysunek 17.

Rysunek 17: Blokowanie ataku siłowego na mechanizm uwierzytelnienia

3.4. Komunikacja z serwerem aplikacyjnym po zaszyfrowanym kanale (TLS 1.2)

W celu zweryfikowania komunikowania się z serwerem aplikacyjnym po zaszyfrowanym kanale z wykorzystaniem TLS 1.2 uruchomiliśmy `Wireshark` z poziomu hosta (uruchomienie Wireshark na łączu w GNS3 nie działało poprawnie). Na rysunku 18. można zauważyć ruch TLS v1.2 pomiędzy hostem (adres 192.168.1.1) a routerem brzegowym (adres 192.168.8.131).



Rysunek 18: Komunikacja z serwerem aplikacyjnym po zaszyfrowanym kanale (TLS 1.2)

4. Wnioski i podsumowanie

Ćwiczenie laboratoryjne było bardzo złożone i sprawiło nam dużo problemów. Połączenie i skonfigurowanie kilku maszyn wirtualnych w GNS3 generowało problemy na każdym kroku realizacji - utrudnieniem był na pewno brak doświadczenia w używaniu środowiska GNS3. Dodatkowo bardzo czasochłonny okazał ię proces hardeningu. Rzetelne przeprowadzenie tego procesu zdecydowanie zwiększa bezpieczeństwo hosta, ale wykonanie go w pełnej formie na czterech maszynach mogłoby stanowić osobny projekt – wart zdecydowanie więcej punktów niż 1. Warto wspomnieć o wymaganiach sprzętowych jakie postawił projekt – włączenie 4 maszyn wirtualnych wewnątrz GNS znacznie obciążyło wykorzystywany sprzęt.

Podsumowując, ćwiczenie uznajemy za bardzo ciekawe, rozwijające i pokazujące w praktyce wykorzystanie aspektów (takich jak VLAN, podsieci ...) omawianych na teleinformatycznych przedmiotach w poprzednich semestrach.

W ramach realizacji projektu wykorzystaliśmy następujące narzędzia:

- Overleaf i Latex – do utworzenia niniejszego sprawozdania.
- draw.io – do utworzenia diagramów.
- GNS3 – do implementacji architektury sieci.
- VMWare Workstation 17 Player – jako środowisko wirtualizacyjne.
- Microsoft Word – do utworzenia macierzy komunikacji.

5. Załączniki

5.1. Serwer aplikacyjny

5.1.1. Plik konfiguracyjny Apache2

```
<VirtualHost *:80>
    RewriteEngine On
    RewriteCond %{HTTPS} !=on
    RewriteRule ^/?(.*) https://%{SERVER_NAME}/$1 [R=301,L]
</virtualhost>
<VirtualHost *:443>
    ServerName 192.168.17.130
    WSGIScriptAlias / /var/www/flaskApp/test2.wsgi
    SSLEngine on
    SSLCertificateFile /etc/apache2/certs/apache.crt
    SSLCertificateKeyFile /etc/apache2/certs/apache.key
    SSLProtocol -all +TLSv1.2
    <Directory /var/www/flaskApp>
        AuthType Basic
        AuthName "Restricted Content"
        Require valid-user
        AuthUserFile /etc/apache2/.htpasswd
    </Directory>
    ErrorLog /var/www/flaskApp/logs/error.log
    LogLevel warn
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Dodatkowo skonfigurowaliśmy przekierowanie z http na https.

5.1.2. Plik .py obsługujący funkcjonalności aplikacji web'owej

```python
from flask import Flask, render_template, request, session, redirect, url_for
import psycopg2

app = Flask(__name__)

def connect_to_db():
    conn = psycopg2.connect(
        dbname="test_erp",
        user="postgres",
        password="postgres",
        host="192.168.17.131",
        port="5432"
    )
    return conn

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/DbResponse', methods=['POST', 'GET'])
def print_data():
    if request.method == "POST":
        #param = request.args.get('choice')
        #data = request.args.get('data')
        param = request.form['choice']
        data = request.form['data']
        conn = connect_to_db()
        cur = conn.cursor()
        if param == 'all':
            cur.execute("SELECT * FROM employees")
        elif (param in ['id', 'first_name', 'last_name', 'department']):
            cur.execute(f"SELECT * FROM employees WHERE {param} = %s", (data,))
        else:
            return "error"
        rows = cur.fetchall()
        conn.close()
        return render_template('dbSite.html',sql_response=rows)
    if request.method == "GET":
        return rows

if __name__ == '__main__':
    app.run()
```

W przypadku tego skryptu warto zaznaczyć sposób tworzenia zapytań SQL'owych. W przypadku wybrania opcji 'all' - zwrócenia wszystkich rekordów wysyłane jest zapytanie utworzone na sztywno. W przypadku wyboru konkretnego atrybutu i wpisania danych wejściowych przez użytkownika np. imię pracownika, zapytanie jest tworzone w sposób sparametryzowany. Zapobiega to atakom typu SQL injection.

5.1.3. WSGI - interfejs bramy serwera webowego

```python
#!/usr/bin/python
# coding: utf-8
import sys
import logging
logging.basicConfig(stream=sys.stderr)
sys.path.insert(0, '/var/www/flaskApp')
from test3 import app as application
application.secret_key = 'asdasad'
```

5.1.4. Strona główna - kod html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Data Submission Form</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            max-width: 400px;
            margin: 0 auto;
            padding: 20px;
        }
        label {
            display: block;
            margin-bottom: 10px;
        }
        input[type="text"], select {
            width: 100%;
            padding: 8px;
            margin-bottom: 10px;
            box-sizing: border-box;
        }
        button {
            background-color: #4CAF50;
            color: white;
            padding: 10px 15px;
            border: none;
            cursor: pointer;
        }
        button:hover {
            background-color: #45a049;
        }
    </style>
</head>
<body>
    <h2>Submit Data</h2>
    <form action="/DbResponse" method="post">
        <label for="choice">Choose an option:</label>
        <select id="choice" name="choice">
        <option value="all">All records</option>
            <option value="id">ID</option>
            <option value="first_name">Name</option>
            <option value="last_name">Surname</option>
            <option value="department">Department</option>
        </select>

        <label for="data">Enter Data:</label>
        <input type="text" id="data" name="data" placeholder="Enter Data">

        <button type="submit">Submit</button>
    </form>
 </body>
</html>
```

5.1.5. Strona zwracające odpowiedzi z bazy danych - kod html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>DbResponse</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            max-width: 400px;
            margin: 0 auto;
            padding: 20px;
        }
        label {
            display: block;
            margin-bottom: 10px;
        }
        input[type="text"], select {
            width: 100%;
            padding: 8px;
            margin-bottom: 10px;
            box-sizing: border-box;
        }
        button {
            background-color: #4CAF50;
            color: white;
            padding: 10px 15px;
            border: none;
            cursor: pointer;
        }
        button:hover {
            background-color: #45a049;
        }
    </style>
</head>
<body>
    <h2> {{sql_response}} </h2>
 </body>
</html>
```