Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Instytut Telekomunikacji

Praca końcowa

na studiach podyplomowych
Inżynieria Cyberbezpieczeństwa

Projekt i wdrożenie systemu bezpieczeństwa sieciowego w środowisku lokalnym z wykorzystaniem konteneryzacji

Łukasz Dejko (1203175)

promotor
dr inz. Jędrzej Bieniasz

WARSZAWA 2025

Projekt i wdrożenie systemu bezpieczeństwa sieciowego w środowisku lokalnym z wykorzystaniem konteneryzacji

Streszczenie. Celem niniejszej pracy jest zaprojektowanie, wdrożenie oraz omówienie funkcjonalności kompleksowego systemu bezpieczeństwa sieciowego w środowisku lokalnym. System ten integruje mechanizmy ochrony warstwy DNS, filtrowanie ruchu HTTP, monitorowanie aktywności sieciowej, detekcję zagrożeń w czasie rzeczywistym oraz centralne gromadzenie i wizualizację danych telemetrycznych. Kluczowym założeniem projektu było wykorzystanie otwartoźródłowego oprogramowania oraz konteneryzacja usług przy użyciu platformy Docker, co zapewnia modularność, izolację oraz możliwość łatwego zarządzania środowiskiem.

Zastosowane komponenty umożliwiają kontrolę nad dostępem do sieci, blokowanie podejrzanych domen, analizę pakietów w czasie rzeczywistym oraz monitorowanie stanu systemu i usług. W pracy szczegółowo opisano konfigurację elementów odpowiedzialnych za filtrowanie DNS, wykrywanie intruzów, analizę logów oraz ich prezentację w formie czytelnych wykresów i pulpitów zarządczych. System został uruchomiony na serwerze domowym z systemem Linux oraz dodatkowo na platformie Raspberry Pi, co pokazuje elastyczność rozwiązania.

Projekt udowadnia, że także w warunkach domowych możliwe jest wdrożenie systemu bezpieczeństwa opartego na najlepszych praktykach znanych z infrastruktury korporacyjnej. Przedstawione podejście może stanowić punkt wyjścia do dalszego zgłębiania zagadnień związanych z bezpieczeństwem rozproszonych środowisk cyfrowych oraz wdrażaniem polityk bezpieczeństwa opartych na danych telemetrycznych i analizie zagrożeń.

Dodatkowo przedstawiono praktyczne zastosowanie omawianego środowiska, stworzonego pod nazwą „Home Network Guardian”[35], wraz z dokumentacją dostępną na dedykowanej stronie internetowej projektu. Praca zawiera szczegółowe wykresy oraz schematy które ilustrują architekturę oraz interakcję między poszczególnymi komponentami.

Opracowane rozwiązanie stanowi efektywny, ekonomiczny i bezpieczny sposób ochrony domowej infrastruktury IT, możliwy do wdrożenia przez każdego świadomego użytkownika technologii informacyjnych.

Słowa kluczowe: bezpieczeństwo sieci, DNS, Docker, analiza logów, filtracja treści, detekcja zagrożeń, Raspberry Pi, monitoring systemu

Design and Implementation of a Network Security System in a Local Environment Using Containerization

Abstract. The purpose of this thesis is to design, implement, and analyze a comprehensive local network security system that integrates DNS-layer protection, HTTP traffic filtering, real-time network activity monitoring, threat detection, and centralized telemetry data visualization. A key objective of the project was the exclusive use of open-source software and containerization via the Docker platform, enabling modularity, service isolation, and ease of management across a distributed architecture.

The implemented solution allows full control over network access, blocking of suspicious domains, real-time traffic analysis, and detailed monitoring of system and service health. The thesis outlines the configuration of components responsible for DNS filtering, intrusion detection, log collection, and interactive dashboards. The system runs on a Linux-based home server and Raspberry Pi, demonstrating its adaptability and cost efficiency.

This work demonstrates that enterprise-grade security practices can be effectively replicated in a domestic setting using widely available technologies. The proposed solution can serve as a starting point for exploring secure digital environments and implementing policy-driven security based on telemetry data and behavioral threat analytics.

Keywords: network security, DNS, Docker, log analysis, content filtering, threat detection, Raspberry Pi, system monitoring

Spis treści

1. Wstęp

1.1. Cel pracy

Głównym celem niniejszej pracy jest zaprojektowanie i wdrożenie kompleksowego systemu bezpieczeństwa sieciowego w środowisku lokalnym z wykorzystaniem konteneryzacji. System ma za zadanie nie tylko filtrować i kontrolować ruch sieciowy, lecz również analizować logi, wykrywać anomalie oraz umożliwiać monitorowanie stanu zasobów w czasie rzeczywistym.

Projekt zakłada integrację wielu niezależnych komponentów bezpieczeństwa opartych na oprogramowaniu typu open source. Do najważniejszych należą: lokalny serwer DNS z funkcją filtrowania (Pi-hole z Unbound), serwer proxy (Squid), system wykrywania zagrożeń IDS (Suricata), platforma do centralnego logowania i analizy zdarzeń (Graylog), system monitoringu (Prometheus i Grafana), a także konteneryzowane środowisko przeglądarki (Firefox) i narzędzia do zarządzania kontenerami (Portainer).

Celem pośrednim pracy jest także przedstawienie zalet architektury opartej na Dockerze, takich jak elastyczność, skalowalność i separacja usług, oraz zastosowanie zasad bezpieczeństwa sieciowego w praktyce, w tym kontrola portów, firewall oraz zabezpieczenia hosta. Projekt ten stanowi dowód na to, że przy użyciu powszechnie dostępnych narzędzi można stworzyć bezpieczne i profesjonalne środowisko do zarządzania ruchem w sieci domowej lub małej firmie.

1.2. Uzasadnienie wyboru tematu

W dobie powszechnej cyfryzacji oraz rosnącej liczby zagrożeń związanych z bezpieczeństwem informacji, istotne staje się zapewnienie odpowiedniego poziomu ochrony nie tylko w środowiskach korporacyjnych, ale także w sieciach domowych. Coraz większa liczba urządzeń podłączonych do Internetu, takich jak smartfony, komputery, kamery IP czy inteligentne sprzęty AGD, powoduje zwiększoną ekspozycję na ataki z zewnątrz. Jednocześnie użytkownicy domowi często nie dysponują narzędziami ani wiedzą pozwalającą na skuteczne przeciwdziałanie zagrożeniom.

Wybór tematu pracy został podyktowany chęcią stworzenia systemu, który — bazując na darmowym oprogramowaniu i dostępnych komponentach sprzętowych — umożliwia wdrożenie rozwiązań znanych z profesjonalnych środowisk bezpieczeństwa IT. Zastosowanie takich narzędzi jak Pi-hole, Suricata czy Graylog pozwala na uzyskanie pełnej widoczności w ruchu sieciowym, detekcję podejrzanych aktywności oraz aktywne zarządzanie zagrożeniami w czasie rzeczywistym.

Temat pracy jest również odpowiedzią na potrzebę edukacji w zakresie budowy bezpiecznych środowisk informatycznych oraz wdrażania dobrych praktyk cyberbezpieczeństwa w ujęciu praktycznym. Dzięki konteneryzacji z użyciem Dockera możliwe jest szybkie i modularne wdrażanie usług, co znacząco ułatwia testowanie oraz skalowanie środo-

wiska. Praca ta stanowi dowód na to, że budowa nowoczesnej i bezpiecznej sieci może być osiągalna nawet w warunkach ograniczonych zasobów.

1.3. Zakres i ograniczenia pracy

Zakres pracy obejmuje zaprojektowanie oraz wdrożenie zintegrowanego systemu bezpieczeństwa sieciowego w środowisku lokalnym z wykorzystaniem otwartoźródłowych narzędzi. Główne obszary realizacji obejmują: filtrowanie DNS i HTTP, wykrywanie anomalii w ruchu sieciowym, analizę i centralizację logów, monitoring systemów oraz zarządzanie kontenerami. Wszystkie usługi zostały uruchomione w kontenerach Docker, z wyjątkiem komponentów systemowych takich jak firewall czy Filebeat, które działają bezpośrednio na hoście.

Praca zawiera szczegółowy opis konfiguracji następujących komponentów:

- serwera DNS z filtrowaniem (Pi-hole) oraz rekurencyjnego resolvera (Unbound),
- serwera proxy (Squid),
- systemu IDS (Suricata),
- platformy do analizy logów (Graylog, Elasticsearch, MongoDB),
- systemu monitoringu (Prometheus, Grafana),
- izolowanego środowiska przeglądarki (Firefox w kontenerze),
- systemu zarządzania kontenerami (Portainer),
- zapory sieciowej (iptables + ufw).

Ze względu na charakter projektu oraz ograniczone zasoby sprzętowe i czasowe, praca nie obejmuje testów penetracyjnych, implementacji systemów klasy EDR, ani automatycznych mechanizmów reakcji na incydenty. Projekt skupia się wyłącznie na rozwiązaniach dostępnych lokalnie — bez użycia usług chmurowych. Ponadto, wdrożone rozwiązania nie były poddawane audytowi zewnętrznemu, a skuteczność ich działania oceniana jest na podstawie obserwacji i dostępnych metryk.

Celem pracy nie jest stworzenie w pełni zgodnego z normami ISO środowiska bezpieczeństwa, lecz raczej przedstawienie możliwości zastosowania ogólnodostępnych narzędzi w celu zwiększenia poziomu ochrony danych i usług w domowej lub małej firmowej sieci komputerowej.

1.4. Metodyka realizacji projektu

Projekt został zrealizowany zgodnie z podejściem iteracyjnym, z podziałem na kolejne etapy obejmujące analizę wymagań, projektowanie architektury, konfigurację środowiska, wdrożenie usług, testowanie oraz dokumentację. Każdy z komponentów bezpieczeństwa został wdrażany i testowany niezależnie, a następnie integrowany z pozostałymi elementami w ramach jednej, spójnej infrastruktury kontenerowej.

Prace rozpoczęto od analizy możliwości sprzętowych oraz wyboru otwartoźródłowych narzędzi najlepiej odpowiadających założonym celom funkcjonalnym. Następnie zapro-

jektowano architekturę logiczną środowiska, w której istotne znaczenie miała separacja usług w kontenerach Docker i ich komunikacja przez dedykowane sieci wirtualne.

W kolejnych krokach skonfigurowano poszczególne komponenty systemu: serwery DNS, serwer proxy, system IDS, agregator logów oraz mechanizmy monitorujące. W celu zapewnienia spójności, dla każdej usługi utworzono oddzielne pliki konfiguracyjne[35] oraz definicje kontenerów (Docker Compose)[33]. Równolegle prowadzono testy funkcjonalne poszczególnych modułów, weryfikując poprawność działania filtracji, detekcji i logowania zdarzeń.

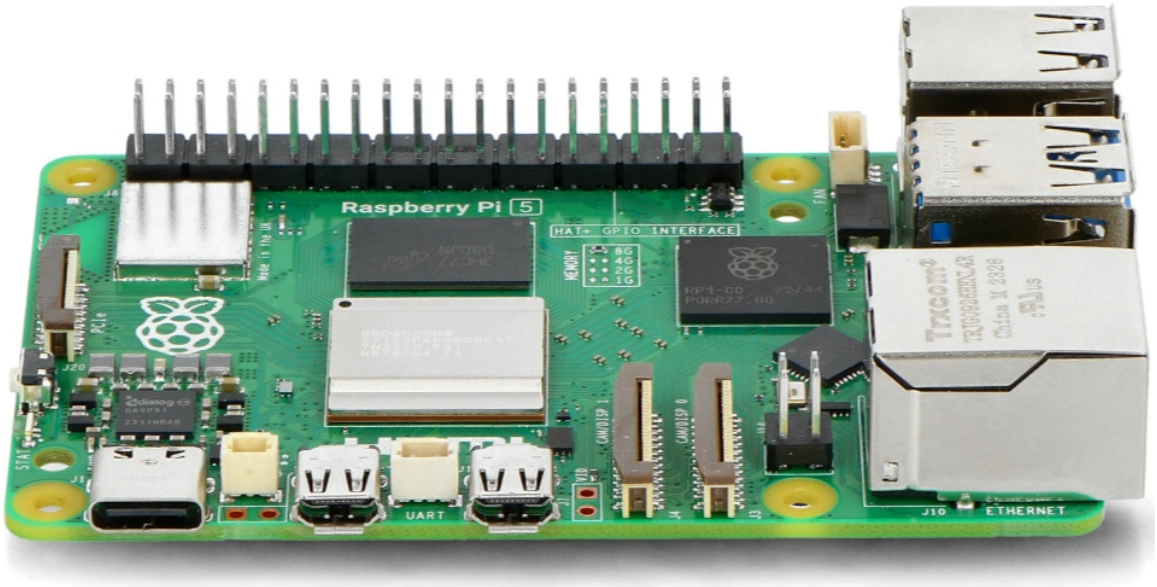
Metodyka pracy zakładała również zastosowanie dokumentacji oficjalnej (ang. vendor documentation) oraz materiałów ogólnodostępnych. W celu oceny skuteczności wdrożonych mechanizmów wykorzystywano dane z logów systemowych oraz panele monitorujące (dashboardy) stworzone w Grafanie, Graylogu i Pihole.

Całość środowiska została osadzona na serwerze z systemem Debian oraz Raspberry Pi, co pozwoliło na weryfikację działania systemu zarówno w warunkach zasobooszczędnych, jak i przy większej wydajności. Projekt zrealizowano w środowisku domowym, co podkreśla jego dostępność i możliwość zastosowania w rzeczywistych warunkach.

2. Opis środowiska i architektury systemu

2.1. Wykorzystany sprzęt i platforma systemowa

Do realizacji projektu wybrano sprzęt, który zapewnia zarówno wysoką wydajność, jak i energooszczędność, co czyni go idealnym rozwiązaniem dla środowiska typu home lab. Główną jednostką obliczeniową jest komputer jednopłytkowy **Raspberry Pi 5**[22] wyposażony w **8 GB pamięci RAM** 2.1. Jego najnowsza generacja charakteryzuje się znaczącym wzrostem mocy obliczeniowej względem poprzednich modeli, dzięki ulepszonemu procesorowi i zintegrowanemu układowi graficznemu. Taka konfiguracja pozwala na równoczesne uruchomienie wielu usług, takich jak *Pi-hole*, *Unbound*, *Suricata* oraz *Graylog*, bez zauważalnego spadku wydajności — nawet przy zwiększonym natężeniu ruchu sieciowego.



Rys. 2.1. Raspberry Pi 5 8 GB pamięci RAM

Jedną z kluczowych cech Raspberry Pi 5 jest obecność magistrali **PCIe**, która została wykorzystana do podłączenia dysku **NVMe Samsung SSD 980 o pojemności 500 GB**. Zastosowanie nośnika SSD w standardzie NVMe znacznie przyspieszyło operacje zapisu i odczytu danych w porównaniu do tradycyjnych kart microSD, co ma szczególne znaczenie w kontekście gromadzenia logów i działania systemów monitorujących.

Całość została umieszczona w dedykowanej obudowie **Argon NEO 5 M.2 NVMe PCIe** 2.2, która nie tylko zapewnia pasywne chłodzenie układów elektronicznych, ale również wspiera bezpośrednie mocowanie i integrację dysku SSD w formacie M.2. Obudowa została wykonana z aluminium, co przekłada się na wysoką trwałość oraz efektywne odprowadzanie ciepła, dzięki czemu możliwa jest długotrwała praca systemu

bez ryzyka przegrzania. Kompaktowa forma oraz estetyczne wykonanie sprawiają, że zestaw doskonale wpisuje się w warunki domowego laboratorium.

Rys. 2.2. Argon NEO 5 M.2 NVMe PCIe

Platforma systemowa oparta została na systemie **Raspberry Pi OS Lite**[21], będącym oficjalną dystrybucją Linuxa wspieraną przez Fundację Raspberry Pi. System ten zapewnia kompatybilność ze środowiskiem Docker oraz dostęp do bogatego repozytorium pakietów, co znacząco ułatwia wdrażanie usług kontenerowych. Wszystkie kontenery zostały uruchomione na silniku **Docker Engine** w wersji zgodnej z architekturą ARM64, natomiast ich zarządzanie odbywa się za pośrednictwem *Portainera*[17].

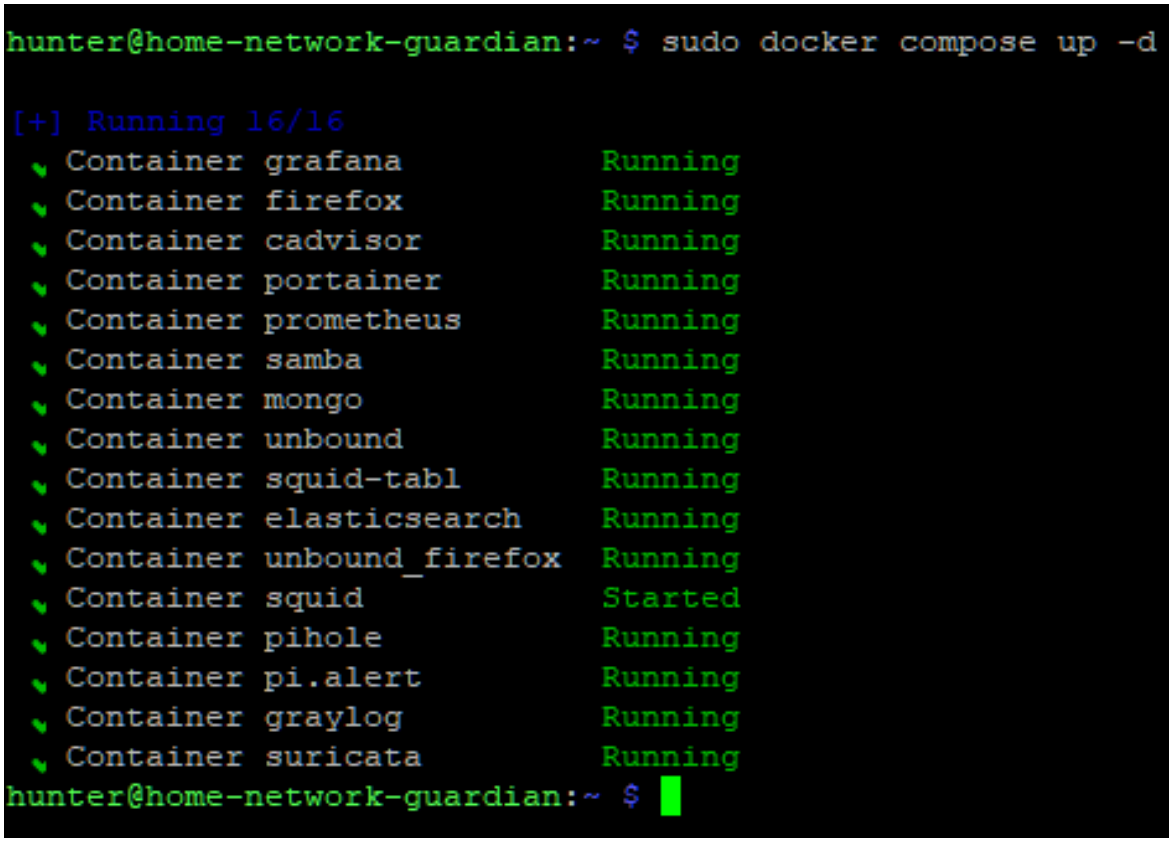
Taka konfiguracja sprzętowo-programowa spełnia wymagania wydajnościowe i funkcjonalne projektu, zapewniając jednocześnie niski pobór mocy, cichą pracę i wysoką stabilność w warunkach domowych.

2.2. Wirtualizacja i konteneryzacja (Docker + Portainer)

W ramach niniejszego projektu zdecydowano się na wykorzystanie lekkiej formy wirtualizacji, jaką jest konteneryzacja z użyciem platformy Docker[30]. Rozwiązanie to zostało wybrane ze względu na swoją elastyczność, wysoką wydajność oraz szerokie wsparcie społeczności i dostawców narzędzi open-source. Konteneryzacja umożliwia uruchamianie wielu odseparowanych usług w ramach jednego systemu operacyjnego, bez konieczności tworzenia pełnych maszyn wirtualnych. Dzięki temu znacznie redukowane są koszty za-

sobów oraz czas potrzebny na wdrożenie i zarządzanie poszczególnymi komponentami systemu bezpieczeństwa.

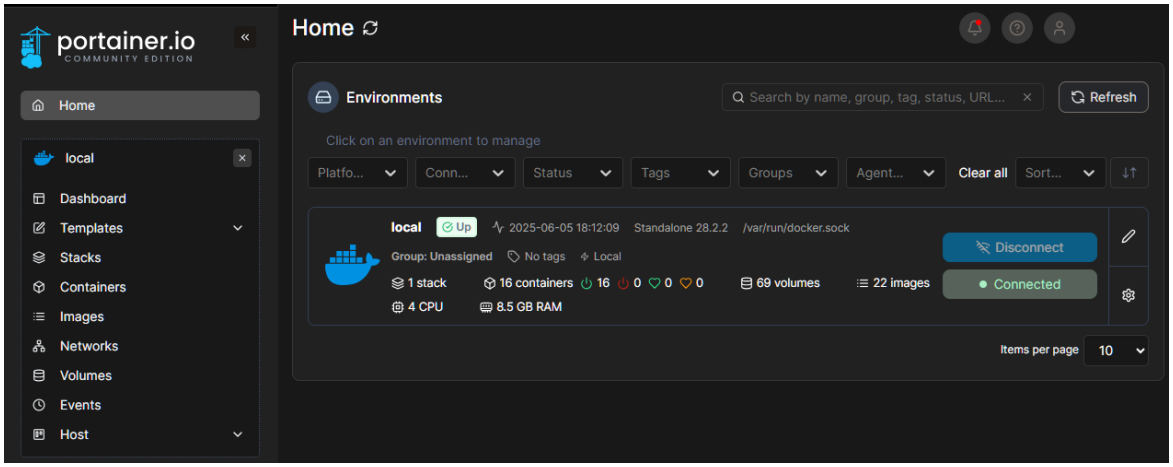
Docker pozwala na budowanie i uruchamianie aplikacji w formie kontenerów 2.3, które zawierają wszystkie niezbędne zależności — biblioteki, pliki konfiguracyjne oraz kod aplikacyjny. Każdy komponent systemu bezpieczeństwa (np. *Pi-hole*, *Unbound*, *Suricata*, *Graylog*, *Prometheus*, *Grafana*) został umieszczony w osobnym kontenerze, co znacząco zwiększa modularność i ułatwia konserwację systemu. Dodatkowo, każda usługa uruchamiana jest w odrębnej sieci wirtualnej Dockera, co umożliwia precyzyjne kontrolowanie ruchu sieciowego między kontenerami i hostem oraz zwiększa bezpieczeństwo całego środowiska.



Rys. 2.3. Lista działających kontenerów uruchomionych za pomocą `docker compose up -d`. Widoczne są m.in. usługi: Pi-hole, Unbound, Squid, Suricata, Graylog, Elasticsearch, MongoDB, Prometheus, Grafana oraz kontener dedykowany dla Firefoksa i Portainera.

Konfiguracja kontenerów została zautomatyzowana przy użyciu pliku `docker-compose.yml` [33], który umożliwia definiowanie całego stosu usług oraz ich zależności w sposób przejrzysty i powtarzalny. Taka forma zarządzania konfiguracją ułatwia proces wdrożenia oraz odtwarzania środowiska na innych urządzeniach lub po awarii.

W celu ułatwienia zarządzania środowiskiem kontenerowym wykorzystano narzędzie **Portainer** — graficzny interfejs użytkownika do zarządzania instancjami Docker 2.4. Portainer pozwala na monitorowanie kontenerów, wolumenów, obrazów, sieci oraz logów systemowych w czasie rzeczywistym. Dzięki niemu możliwe jest również szybkie uruchamianie i restartowanie usług, tworzenie nowych środowisk oraz kontrola nad dostępem użytkowników. Interfejs Portainera został zabezpieczony autoryzacją z hasłem i ograniczonym dostępem po porcie TCP/HTTPS, co stanowi dodatkowy element ochrony.



Rys. 2.4. Widok panelu Portainer – aplikacja zarządzająca kontenerami Docker. Przedstawiono środowisko lokalne z 16 aktywnymi kontenerami, 69 wolumenami i 22 obrazami. Portainer umożliwia łatwe zarządzanie stackami, kontenerami, obrazami, sieciami oraz zasobami wolumenów z poziomu przeglądarki internetowej.

Separacja usług w kontenerach umożliwia niezależne zarządzanie i aktualizację każdego komponentu bez wpływu na pozostałe elementy systemu. Jest to szczególnie istotne w kontekście bezpieczeństwa: w przypadku wykrycia podatności w jednej z aplikacji, możliwe jest szybkie zastosowanie aktualizacji tylko dla konkretnego kontenera bez konieczności przerywania działania całego systemu.

Z punktu widzenia cyberbezpieczeństwa, konteneryzacja oferuje dodatkową warstwę izolacji procesów, co utrudnia eskalację uprawnień w przypadku ewentualnego włamania. Zastosowanie polityk sieciowych Dockera pozwala ograniczyć komunikację pomiędzy usługami tylko do niezbędnych przypadków, zgodnie z zasadą najmniejszych uprawnień (ang. *principle of least privilege*).

Dzięki konteneryzacji cały system jest przenośny, łatwo skalowalny i podatny na automatyzację. Możliwe jest również wykorzystanie narzędzi takich jak *Watchtower* do automatycznego aktualizowania kontenerów, co dodatkowo zwiększa poziom bezpieczeństwa systemu poprzez ograniczenie ekspozycji na znane podatności.

Podsumowując, zastosowanie Dockera i Portainera pozwoliło na stworzenie spójnego, modularnego i łatwego w utrzymaniu środowiska, które może być rozwijane i skalowane w przyszłości bez konieczności przebudowy całej infrastruktury.

2.3. Architektura logiczna systemu bezpieczeństwa

Architektura logiczna wdrożonego systemu bezpieczeństwa opiera się na centralnym serwerze zbudowanym na platformie Raspberry Pi 5, który pełni funkcję węzła integrującego różnorodne usługi odpowiedzialne za filtrowanie, monitorowanie, analizę i kontrolę ruchu sieciowego w sieci lokalnej. Każdy z komponentów został uruchomiony w osobnym kontenerze Docker, co zapewnia ich niezależność, ułatwia aktualizację, izolację środowisk oraz zarządzanie cyklem życia usług.

Urządzenia końcowe w sieci domowej (komputery, smartfony, tablety, telewizory smart) komunikują się z Internetem poprzez serwer DNS realizowany przez **Pi-hole**. Komponent ten odpowiada za filtrowanie zapytań DNS oraz blokowanie niepożądanych domen (reklamowych, śledzących, złośliwych). Zapytania te są następnie przekazywane do **Unbound**, który jako lokalny rekurencyjny resolver zapewnia prywatność i niezależność od zewnętrznych dostawców DNS.

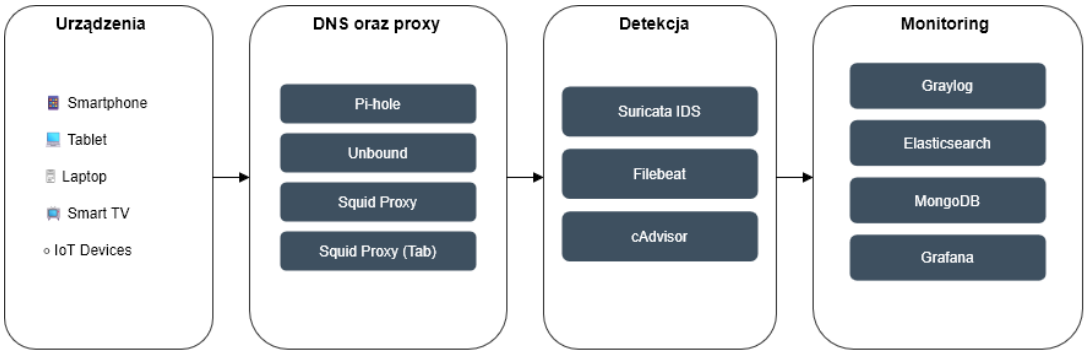
W celu kontroli i analizy ruchu HTTP/S, pakiety mogą być przekierowywane do serwera **Squid Proxy**, który pełni również funkcję bufora i kontrolera dostępu. Analiza ruchu sieciowego pod kątem zagrożeń prowadzona jest przez system **Suricata** działający jako IDS, który wykorzystuje sygnatury oraz mechanizmy heurystyczne.

Zbieraniem logów systemowych i aplikacyjnych zajmuje się **Filebeat**, który przekazuje je do systemu **Graylog**, opartego na silniku wyszukiwania **Elasticsearch** oraz bazie danych **MongoDB**. Graylog umożliwia centralną analizę, korelację i wizualizację zdarzeń. Równocześnie, dane telemetryczne i metryki systemowe są zbierane przez **Prometheus**, a następnie wizualizowane w **Grafanie** poprzez interaktywne dashboardy.

Środowisko kontenerowe zarządzane jest z poziomu aplikacji **Portainer**, która umożliwia łatwą administrację usługami działającymi w Dockerze.

Komponenty systemu są połączone za pomocą kilku odseparowanych wirtualnych sieci Dockera, co umożliwia implementację zasad izolacji i minimalizacji powierzchni ataku. Przykładowo, kontener `firefox` funkcjonuje w dedykowanej podsieci `firefox_network`, z przypisanym prywatnym DNS-em zapewnianym przez osobny kontener `unbound_firefox`. Takie podejście umożliwia granularną kontrolę dostępu między usługami oraz lepszą segmentację środowiska bezpieczeństwa.

Na rysunku 2.5 przedstawiono uproszczony schemat logiczny systemu bezpieczeństwa. Ilustruje on przepływ danych od urządzeń końcowych, przez warstwę filtrowania DNS (Pi-hole i Unbound), usługę proxy (Squid), aż po wyjście do Internetu, z jednoczesnym monitoringiem i rejestrowaniem zdarzeń.



Rys. 2.5. Architektura logiczna systemu.

Jednym z kluczowych założeń projektowych było zapewnienie logicznej i funkcjonalnej separacji usług działających w ramach systemu bezpieczeństwa, co przekłada się bezpośrednio na podniesienie poziomu ochrony sieci lokalnej. Zastosowano podejście zgodne z zasadą *security by design*, w którym każda usługa posiada wyraźnie zdefiniowaną funkcję, dostęp do tylko niezbędnych zasobów oraz ograniczoną komunikację z innymi komponentami.

Cała architektura została oparta na platformie Docker[7], która umożliwia uruchamianie poszczególnych usług w izolowanych kontenerach. Kontenery te zostały pogrupowane w logiczne segmenty sieci wirtualnych Docker, co pozwala na granularne kontrolowanie komunikacji między nimi. Przykładowo, kontener *Squid Proxy* ma dostęp do zewnętrznego Internetu oraz do kontenerów *Pi-hole* i *Unbound*, lecz nie posiada bezpośredniego połączenia z usługami logującymi, takimi jak *Graylog* czy *Prometheus*.

Separacja realizowana jest również na poziomie warstwy sieciowej dzięki wykorzystaniu wielu mostów sieciowych (ang. *Docker bridge networks*). Dla każdej grupy usług zdefiniowano osobną sieć wirtualną, co pozwala na kontrolowanie tras routingu, zamykanie nieużywanych portów oraz precyzyjne definiowanie reguł dostępu przy użyciu mechanizmów zapory ogniowej (iptables) oraz konfiguracji Dockera.

Z poziomu systemu hosta (Raspberry Pi 5), zastosowano dodatkowe zabezpieczenia w postaci lokalnego firewalla (ufw), który ogranicza dostęp do wybranych portów wyłącznie z określonych adresów IP lub interfejsów sieciowych. Dzięki temu dostęp do interfejsów zarządzających (np. Portainer, Graylog, Grafana) możliwy jest wyłącznie z wybranych urządzeń w sieci lokalnej, co znacząco ogranicza powierzchnię potencjalnych ataków.

Dodatkowo, każda aplikacja kontenerowa uruchamiana jest z jak najmniejszym zestawem uprawnień (zgodnie z zasadą najmniejszych przywilejów), bez trybu uprzywilejowanego (ang. *–privileged*) oraz bez montowania niepotrzebnych wolumenów hosta. W miejscach wymagających zapisu danych (np. logi, konfiguracje), wykorzystano dedykowane wolumeny Docker z ograniczonym dostępem do systemu plików.

Segmentacja logiczna i funkcjonalna usług nie tylko zwiększa bezpieczeństwo, ale również ułatwia zarządzanie systemem, umożliwia szybsze diagnozowanie problemów oraz

ogranicza skutki ewentualnych błędów lub kompromitacji poszczególnych komponentów. Rozdzielenie usług umożliwia też niezależne aktualizacje i restarty bez wpływu na stabilność całego środowiska.

2.4. Zestawienie komponentów, portów i ról w systemie

W tabeli 2.1 zestawiono wszystkie główne komponenty wdrożonego systemu bezpieczeństwa, wraz z informacją o wykorzystywanych portach sieciowych, przypisanej sieci Docker (lub trybie hosta) oraz pełnionej funkcji. Zestawienie to pozwala w szybki sposób zrozumieć strukturę logiczną i sieciową systemu.

Tabela 2.1. Usługi i porty w systemie bezpieczeństwa

Usługa	**Port(y)**	**Sieć**	**Funkcja**
Pi-hole	53/tcp, 53/udp, 80	internal_network squid_network	DNS filtrujący, interfejs webowy
Unbound (dla Pihole)	-	internal_network	Dedykowany DNS rekurencyjny tylko dla Pihole
Squid Proxy (LAN)	3128	squid_network	HTTP/HTTPS proxy dla urządzeń LAN
Squid Proxy (Mobile)	3129	squid_network	HTTP/HTTPS proxy dla urządzeń mobilnych
Firefox	4000, 4001	firefox_network	Izolowana przeglądarka z dostępem przez noVNC
Unbound (dla Firefox)	-	firefox_network	Dedykowany DNS rekurencyjny tylko dla Firefox
Suricata IDS	-	host	Monitorowanie i analiza ruchu sieciowego
Filebeat	5044 (out)	host (na serwerze)	Agent logów (Suricata, Squid, Pi-hole, dysk)
Graylog	9000, 5044	host	Centralne logowanie i analiza zdarzeń
Elasticsearch	9200	host	Indeksowanie i przeszukiwanie logów
MongoDB	domyślnie 27017	host	Baza danych Grayloga (konfiguracja, dashboardy)
Grafana	3000	host	Wizualizacja metryk i integracja z Prometheus
Prometheus	9090	host	Zbieranie metryk systemowych i kontenerów
Portainer	9010	portainer_network	Panel do zarządzania kontenerami Docker

3. Konfiguracja i rola poszczególnych komponentów

3.1. Docker network – konfiguracja izolacji kontenerów

W celu zapewnienia kontroli przepływu danych oraz realizacji zasady ograniczonego zaufania (*zero trust*), cały system został zorganizowany w oparciu o logicznie wydzielone sieci wirtualne Docker 3.1, działające na sterowniku typu `bridge`. Każda sieć odpowiada innej klasie usług i została zdefiniowana z osobną przestrzenią adresową (`subnet`), co zapewnia przejrzystość oraz granularną kontrolę nad komunikacją między komponentami.

Tabela 3.1. Konfiguracja sieci Docker — adresacja i parametry

Nazwa sieci	Typ	Subnet (IPv4)	Gateway (IPv4)	Właściciel
bridge	bridge (System)	172.17.0.0/16	172.17.0.1	public
host	host (System)	-	-	public
hunter_default	bridge	172.18.0.0/16	172.18.0.1	admin
firefox_network	bridge	172.30.0.0/24	172.30.0.1	admin
internal_network	bridge	172.20.0.0/24	172.20.0.1	admin
portainer_network	bridge	172.50.0.0/24	172.50.0.1	admin
squid_network	bridge	172.40.0.0/24	172.40.0.1	admin

Zdefiniowano następujące sieci:

- **internal_network** – podstawowa sieć przeznaczona dla kluczowych komponentów bezpieczeństwa: Pi-hole, Unbound (dla Pi-hole), Squid Proxy oraz Suricata. Sieć ta jest izolowana od systemu hosta, a dostęp do Internetu realizowany jest wyłącznie przez ściśle określone punkty (np. proxy).
- **firefox_network** – dedykowana sieć dla kontenera przeglądarki Firefox oraz powiązanej z nią instancji Unbound. Zapytania DNS z Firefox są obsługiwane wyłącznie przez przypisany resolver DNS (Unbound na adresie `172.30.0.2`), a przeglądarka nie posiada dostępu do pozostałych komponentów systemu.
- **squid_network** – osobna sieć dla drugiej instancji Squid Proxy, przeznaczonej do obsługi specyficznych grup urządzeń (np. dziecięcych, mobilnych), co umożliwia zastosowanie odrębnych reguł filtracji i logowania. Rozdzielenie tej instancji od głównej sieci proxy pozwala na precyzyjne zarządzanie polityką dostępu.
- **portainer_network** – wydzielona sieć przeznaczona wyłącznie dla kontenera Portainer, który zapewnia zarządzanie środowiskiem Docker. Izolacja Portainera od innych usług zapobiega możliwości jego nadużycia jako wektora dostępowego do kontenerów produkcyjnych.

Każda z tych sieci została zdefiniowana z własnym zakresem adresów IP (`subnet`), co umożliwia przypisywanie statycznych adresów poszczególnym kontenerom. Dzięki temu możliwe było np. wskazanie Unbounda jako DNS o konkretnym adresie IP z poziomu

Firefox lub ograniczenie dostępności usług DNS i HTTP jedynie do kontenerów z tej samej podsieci.

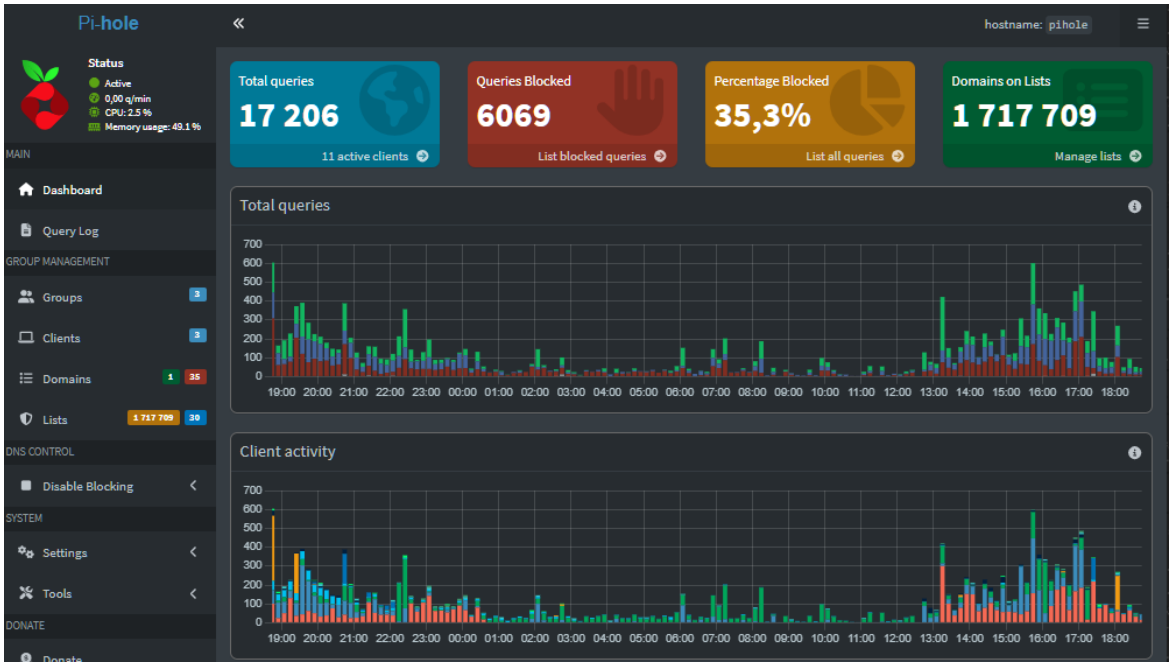
Taka architektura sieciowa pozwala również na:

- niezależne restartowanie i aktualizowanie grup kontenerów bez wpływu na pozostałe usługi,
- pełną kontrolę nad routingiem pakietów i dostępem między sieciami,
- zastosowanie reguł zapory ogniowej (np. UFW lub `iptables`) w oparciu o znane, przewidywalne adresy IP,
- łatwe dodanie warstwowej analizy ruchu (np. Suricata z dostępem do wybranych podsieci),
- eliminację potrzeby publikowania portów na zewnątrz w wielu przypadkach (dostęp wewnętrzny tylko przez sieci Docker).

Zastosowanie oddzielnych sieci dla poszczególnych typów komponentów odzwierciedla architekturę *security by design* oraz wspiera zasadę najmniejszych uprawnień. Dzięki temu komponenty o różnym poziomie zaufania nie mogą się ze sobą komunikować w sposób niekontrolowany, a ewentualna kompromitacja jednego kontenera nie wpływa bezpośrednio na pozostałe.

3.2. Pi-hole jako centralny DNS filtrujący

Pi-hole 3.1 pełni w prezentowanym systemie funkcję centralnego serwera DNS, odpowiadającego za obsługę zapytań DNS z całej sieci lokalnej. Jego głównym zadaniem jest filtrowanie niechcianych zapytań do domen z list czarnej listy (blacklist), w tym blokowanie reklam, śledzenia użytkowników (trackery), elementów malware oraz innych potencjalnie niebezpiecznych domen. Dzięki temu Pi-hole [15] stanowi pierwszą linię obrony przed wieloma zagrożeniami jeszcze zanim ruch opuści sieć lokalną.



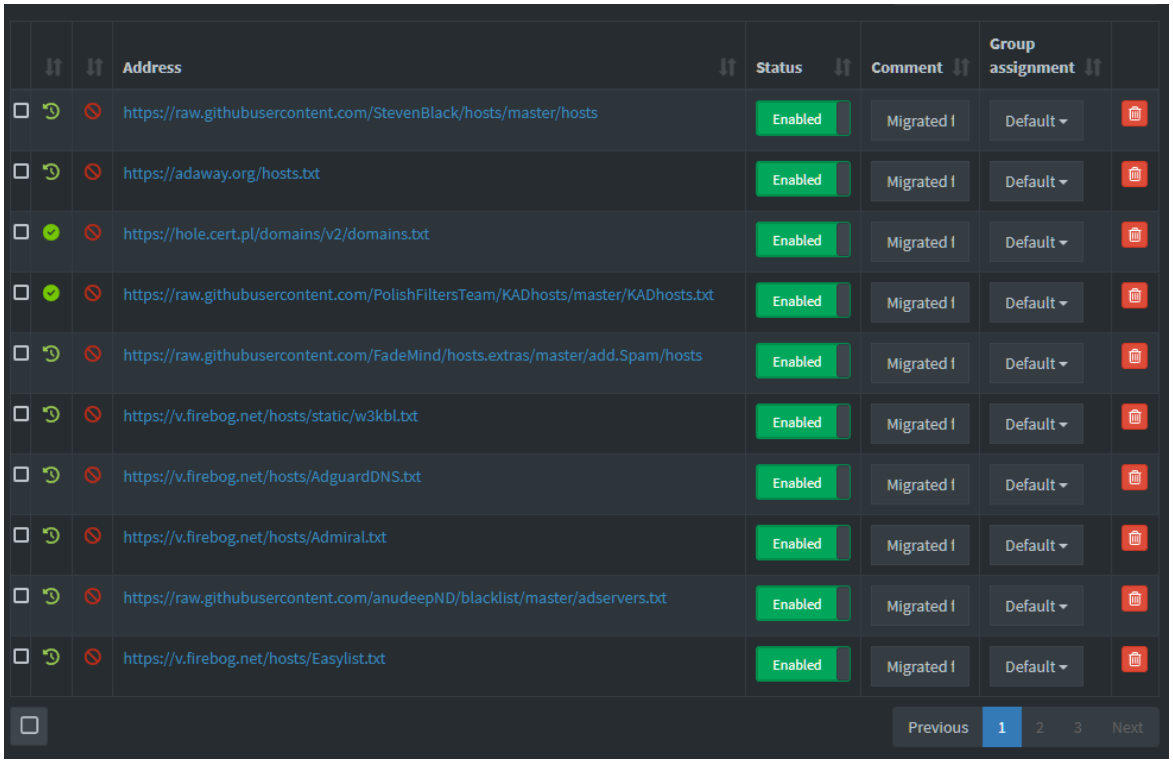
Rys. 3.1. Panel zarządzania Pi-hole – interfejs przedstawiający statystyki działania systemu DNS filtrującego.

Pi-hole został uruchomiony w kontenerze Docker na platformie Raspberry Pi 5, co umożliwia jego łatwą aktualizację, przenoszenie oraz niezależne zarządzanie. Kontener został skonfigurowany z odpowiednimi wolumenami trwałymi, umożliwiającymi zachowanie ustawień i statystyk po restarcie[37]. W konfiguracji uwzględniono mapowanie portów 53 (UDP i TCP) dla zapytań DNS oraz port 80 dla interfejsu webowego, z dodatkowym ograniczeniem dostępu za pomocą firewalla.

Pi-hole został połączony z usługą **Unbound**, która działa jako lokalny, rekurencyjny resolver DNS. Taki układ eliminuje konieczność korzystania z zewnętrznych serwerów DNS[28] (np. Google, Cloudflare), co znacząco poprawia prywatność użytkownika i uniezależnia system od komercyjnych operatorów. Zapytania rozwiązywane są bezpośrednio od źródła (root DNS servers), z wykorzystaniem mechanizmów weryfikacji DNSSEC.

Lista blokowanych domen oparta jest o popularne źródła publiczne, takie jak `https://firebog.net`, uzupełnione o ręcznie dodane wpisy dopasowane do potrzeb użytkowników domowych. Pi-hole rejestruje każde zapytanie DNS i udostępnia statystyki

w czytelnej formie webowego interfejsu administracyjnego. Użytkownik może przeglądać historię zapytań, najczęściej blokowane domeny, statystyki według urządzeń, a także dynamicznie zarządzać listami dozwolonych (whitelist) i zablokowanych (blacklist) domen 3.2.



Rys. 3.2. Panel zarządzania Pi-hole – Przykładowe listy zaimportowane do Pihole.

Z perspektywy bezpieczeństwa, Pi-hole pozwala na:

- blokowanie phishingu i złośliwych domen,
- ograniczenie śledzenia przez zewnętrzne podmioty (trackery reklamowe),
- zwiększenie prywatności użytkowników sieci lokalnej,
- filtrowanie treści nieodpowiednich dla dzieci (przy odpowiedniej konfiguracji list).

W ramach systemu Pi-hole stanowi zatem kluczowy komponent w warstwie prewencji i kontroli ruchu DNS, a jego integracja z Unbound oraz systemami logującymi czyni go silnym narzędziem analityczno-filtrowym w domowej infrastrukturze bezpieczeństwa.

3.3. Unbound jako rekurencyjny resolver DNS

Unbound[13] pełni w opisywanym systemie rolę lokalnego, rekurencyjnego resolvera DNS, który współpracuje bezpośrednio z serwerem Pi-hole. Jego głównym zadaniem jest niezależne i bezpieczne rozwiązywanie zapytań DNS[28], bez udziału zewnętrznych operatorów (takich jak Google DNS czy Cloudflare), co znacząco zwiększa poziom prywatności i integralności komunikacji sieciowej.

Zamiast przekazywać zapytania DNS dalej do publicznych resolverów, Unbound kontaktuje się bezpośrednio z autorytatywnymi serwerami DNS — począwszy od tzw. root servers — a następnie iteracyjnie pobiera odpowiedzi od serwerów nadrzędnych (TLD)[29] aż do uzyskania odpowiedzi końcowej. Dzięki temu możliwe jest uniezależnienie infrastruktury od zewnętrznych dostawców oraz lepsza kontrola nad ruchem DNS.

W środowisku projektu Unbound został skonfigurowany jako osobny kontener Docker, z dostępem jedynie z wewnętrznej sieci Dockerowej, do której podłączony jest również kontener Pi-hole[42]. W ten sposób zapytania DNS przychodzące od klientów trafiają najpierw do Pi-hole (który wykonuje filtrowanie), a następnie przekazywane są do Unbound w celu rekurencyjnego rozwiązania. Taka dwupoziomowa struktura poprawia zarówno bezpieczeństwo, jak i dokładność obsługi zapytań.

Konfiguracja Unbound została oparta na oficjalnych wytycznych projektu Pi-hole, z dodatkowymi usprawnieniami bezpieczeństwa:

- włączono **DNSSEC**[27] – mechanizm weryfikacji kryptograficznej odpowiedzi DNS,
- ograniczono maksymalną liczbę jednoczesnych klientów, aby zapobiec atakom DDoS,
- skonfigurowano Unbound do pracy tylko w trybie lokalnym, nasłuchującym jedynie na interfejsach wewnętrznych,
- ustawiono cache DNS w celu przyspieszenia odpowiedzi na często powtarzające się zapytania.

Dzięki zastosowaniu Unbound jako lokalnego resolvera możliwe jest osiągnięcie następujących korzyści:

- pełna niezależność od zewnętrznych dostawców DNS,
- zwiększenie prywatności zapytań DNS (brak logowania i śledzenia przez operatorów zewnętrznych),
- wsparcie dla DNSSEC – ochrona przed zatruciem cache i fałszywymi odpowiedziami,
- skrócenie czasu odpowiedzi dzięki lokalnemu cache.

Współpraca Pi-hole z Unbound stanowi nowoczesne i bezpieczne rozwiązanie DNS, rekomendowane w środowiskach, gdzie priorytetem jest prywatność użytkownika, integralność danych oraz niezależność od usług zewnętrznych. Całość działa w pełni lokalnie, co jest istotne w kontekście koncepcji „zero trust” oraz ograniczania powierzchni ataku.

3.4. Squid Proxy – filtrowanie i kontrola dostępu do Internetu

Squid Proxy[23] pełni w systemie funkcję transparentnego pośrednika w komunikacji HTTP i HTTPS pomiędzy urządzeniami końcowymi a zasobami Internetu. Jego zadaniem jest nie tylko buforowanie treści w celu przyspieszenia dostępu i ograniczenia ruchu wychodzącego, ale przede wszystkim implementacja reguł filtrowania, kontroli dostępu oraz logowania zdarzeń sieciowych w warstwie aplikacji.

W niniejszym projekcie Squid został uruchomiony jako osobny kontener w środowisku Docker, w ramach tej samej infrastruktury opartej na Raspberry Pi 5. Kontener został przydzielony do sieci wirtualnej wspólnej z kontenerem Pi-hole i Unbound, co umożliwia jego pełną integrację z warstwą DNS[39]. Urządzenia końcowe w sieci lokalnej (komputery, smartfony, tablety) kierują ruch HTTP/S do Squida, który następnie rozwiązuje adresy DNS za pośrednictwem Pi-hole i Unbound, oraz przekazuje żądania dalej — do Internetu.

Konfiguracja Squida została dostosowana do potrzeb środowiska domowego, z naciskiem na bezpieczeństwo, przejrzystość i możliwość rozbudowy. Wdrożono następujące mechanizmy:

- **ACL (Access Control Lists)** 3.3 – listy kontroli dostępu definiujące, które adresy IP, domeny lub kategorie URL mogą być dostępne, a które blokowane,
- **Blokowanie treści niepożądanych** – filtrowanie stron zawierających reklamy, treści dla dorosłych lub znane z hostowania złośliwego oprogramowania,
- **Zarządzanie ruchem HTTPS** – obsługa połączeń TLS z pominięciem inspekcji pakietów, z zachowaniem logów statystycznych na poziomie domen,
- **Cache HTTP** – buforowanie często odwiedzanych zasobów w celu oszczędzania pasma i szybszego ładowania stron,
- **Transparent proxy** – możliwość pracy w trybie przeźroczystym, gdzie ruch przekierowywany jest automatycznie z poziomu routera/firewalla (w przyszłych wdrożeniach).

```
1   http_port 3128
2
3   acl blocklist_stevenblack dstdomain "/etc/squid/blocklists/stevenblack-domains.txt"
4
5   http_access allow all
6
7   dns_nameservers 172.40.0.1
8
9   logformat jsonlog {"timestamp":
    "%tl","client":"%>a","method":"%rm","url":"%ru","status":%Hs,"bytes":%<st,"referer":"%{Referer}>h","user_agent
    ":"%{User-Agent}>h","blocked":%ACLMatchedName}
10  access_log stdio:/var/log/squid/access.json jsonlog blocklist_stevenblack
11
12  access_log /var/log/squid/access.log
13  cache_log /var/log/squid/cache.log
```

Rys. 3.3. Plik konfiguracyjny Squid Proxy – Implementacja list ACL.

Wszystkie logi generowane przez Squida są kierowane do systemu **Filebeat**, a następnie przesyłane do **Grayloga**, gdzie podlegają korelacji z innymi zdarzeniami sieciowymi np. alertami z Suricaty. Dzięki temu możliwe jest śledzenie i analiza aktywności użytkowników oraz wykrywanie potencjalnych anomalii, takich jak próby dostępu do nieautoryzowanych serwisów, złośliwy ruch wychodzący czy wykorzystanie tuneli proxy.

Squid umożliwia również stworzenie podstawowych polityk kontroli dostępu do Internetu, takich jak:

- ograniczenia czasowe (np. blokada serwisów społecznościowych poza godzinami pracy),

- białe i czarne listy domen lub kategorii treści,
- ograniczenia transferu danych dla konkretnych adresów IP lub grup.

Integracja Squida z Pi-hole i Unbound zapewnia spójność filtracji oraz centralizację zarządzania ruchem sieciowym. Całość działa wewnątrz odizolowanego środowiska Docker, co znacząco podnosi poziom bezpieczeństwa i pozwala łatwo wdrażać aktualizacje i rozszerzenia.

Podsumowując, Squid Proxy w tej architekturze pełni kluczową rolę w wymuszaniu polityk bezpieczeństwa, filtrowaniu treści oraz zapewnianiu przejrzystości działania użytkowników w sieci domowej.

3.5. Suricata – system detekcji zagrożeń IDS

Suricata[31] to zaawansowany system detekcji zagrożeń (IDS – Intrusion Detection System), który został wdrożony w ramach systemu bezpieczeństwa w celu monitorowania i analizowania ruchu sieciowego w czasie rzeczywistym. Narzędzie to wykorzystuje mechanizmy inspekcji głębokiej (Deep Packet Inspection, DPI), analizy protokołów aplikacyjnych (np. HTTP, TLS, DNS, FTP) oraz wykrywania anomalii, aby identyfikować potencjalnie złośliwe działania w sieci lokalnej[14].

W prezentowanej architekturze Suricata[40] działa w trybie pasywnym jako komponent analizujący ruch przepływający przez interfejsy wirtualne sieci Docker. Została uruchomiona w osobnym kontenerze z ograniczonymi uprawnieniami, co zapewnia zarówno izolację od reszty systemu, jak i łatwość zarządzania. Kontener Suricaty został przypisany do tej samej sieci co host, dzięki czemu może obserwować cały wewnętrzny ruch DNS i HTTP/S generowany przez urządzenia końcowe w sieci[14].

Główne funkcje Suricaty obejmują:

- wykrywanie prób skanowania portów, ataków typu brute-force i exploitów znanych podatności,
- analizę zapytań DNS i żądań HTTP w celu identyfikacji połączeń do złośliwych domen lub serwerów C&C,
- wykrywanie anomalii i nietypowych wzorców ruchu (np. nadmiarowe zapytania DNS, nadmierne połączenia TCP, niespodziewane protokoły),
- rozpoznawanie sygnatur z baz takich jak Emerging Threats (ET Ruleset),
- integrację z systemami logowania i monitoringu (np. Filebeat i Graylog).

Dodatkowo Suricata korzysta z reguł detekcji pochodzących z zestawu **ET Open Rules**, które obejmują najczęstsze zagrożenia występujące globalnie. Dla zwiększenia efektywności ochrony dodane zostały własne reguły zawarte w pliku `custom.rules`[34]. Własne reguły pozwalają na dostosowanie systemu do specyficznych potrzeb lokalnego ruchu, zwiększają precyzję wykrywania zagrożeń oraz minimalizują fałszywe alarmy.

Reguły zawarte w pliku `custom.rules` obejmują:

- **Wykrywanie skanowania portów za pomocą Nmap/Masscan (TCP)**: Detekcja szybkich skanów SYN, typowych dla narzędzi skanujących porty.
- **Wykrywanie skanowania portów UDP**: Monitorowanie podejrzanie intensywnego ruchu UDP, co wskazuje na możliwe skanowanie portów UDP.
- **Wykrywanie ataku ICMP Flood**: Detekcja dużej ilości pakietów ICMP typu „Echo Request", wskazującej na potencjalny atak DDoS.
- **Wykrywanie spoofingu ARP (atak MITM)**: Reguła wykrywająca fałszywe pakiety ARP, które mogą być próbą ataku typu Man-in-the-Middle.
- **Wykrywanie prób brute-force SSH**: Monitorowanie prób wielokrotnych logowań do SSH, co sugeruje ataki typu brute-force na usługę SSH.
- **Wykrywanie tunelowania danych przez DNS**: Detekcja dużych pakietów UDP kierowanych do serwerów DNS, które mogą świadczyć o próbach eksfiltracji danych przez DNS.
- **Wykrywanie możliwej iniekcji SQL w żądaniach HTTP**: Monitorowanie żądań HTTP zawierających podejrzane ciągi takie jak „select" lub „union", co sugeruje próbę ataku SQL Injection.
- **Wykrywanie prób wykorzystania exploitu EternalBlue (SMB)**: Detekcja specyficznych ciągów znaków używanych w atakach opartych na exploicie EternalBlue na usługi SMB.
- **Wykrywanie dużych pakietów ICMP (Ping of Death)**: Alarmowanie o niezwykle dużych pakietach ICMP, mogących świadczyć o ataku Ping of Death.
- **Wykrywanie fragmentowanych pakietów ICMP (ewazja lub atak PoD)**: Detekcja fragmentowanych pakietów ICMP, które mogą być próbą unikania zabezpieczeń lub atakiem Ping of Death.

Każda reguła jest szczegółowo skonfigurowana, co umożliwia precyzyjne wykrywanie zagrożeń i łatwe dostosowywanie systemu do bieżących potrzeb i specyfiki lokalnego ruchu sieciowego. Reguły wykorzystują charakterystyczne wzorce, które pomagają w szybkim identyfikowaniu incydentów bezpieczeństwa i podejmowaniu właściwych działań zapobiegawczych lub naprawczych.

Z punktu widzenia przepływu danych, Suricata analizuje pakiety sieciowe przychodzące do hosta, na którym działa usługa DNS (Pi-hole oraz Unbound). Ponieważ z tego serwera DNS korzystają wszystkie urządzenia w sieci, Suricata umożliwia wykrywanie m.in. prób połączeń do złośliwych domen, anomalii w żądaniach DNS, czy też nietypowego ruchu generowanego przez klientów. Wykryte zdarzenia są logowane w formacie JSON i przekazywane do kontenera **Filebeat**, który odpowiada za ich przesyłanie do systemu logowania **Graylog**.

Dzięki integracji z Graylogiem możliwe jest:

- budowanie interaktywnych dashboardów przedstawiających statystyki zagrożeń,

- szybkie przeszukiwanie logów wg adresów IP, typów ataków, portów docelowych i źródeł,
- korelacja zdarzeń z innymi komponentami systemu np. Squid,
- generowanie alertów w czasie rzeczywistym.

Suricata została skonfigurowana do pracy w trybie wydajnym z wykorzystaniem wielu wątków (multithreading) i optymalizacji bufora sieciowego, co pozwala na analizę dużego wolumenu ruchu bez obciążania zasobów Raspberry Pi 5. Mimo ograniczeń sprzętowych, testy wykazały, że system działa stabilnie i niezawodnie, nawet przy dużej liczbie zapytań DNS oraz jednoczesnym ruchu HTTP z kilku urządzeń końcowych.

Podsumowując, Suricata pełni w systemie rolę aktywnego czujnika sieciowego, który umożliwia wczesne wykrywanie zagrożeń, dostarcza cennych danych analitycznych oraz stanowi uzupełnienie pasywnego filtrowania oferowanego przez Pi-hole i kontrolę ruchu przez Squid. Jej obecność zwiększa świadomość sytuacyjną administratora i umożliwia szybszą reakcję na potencjalne incydenty bezpieczeństwa.

3.6. Firewall – konfiguracja i zarządzanie dostępem do usług

Jednym z istotnych elementów zabezpieczenia wdrożonego systemu jest lokalna zapora sieciowa (**firewall**) skonfigurowana bezpośrednio na hoście systemowym, czyli Raspberry Pi 5. Jej zadaniem jest ograniczenie dostępu do usług uruchamianych w kontenerach Docker, kontrola portów i interfejsów sieciowych, a także zapobieganie nieautoryzowanemu dostępowi do krytycznych komponentów systemu bezpieczeństwa.

W projekcie zastosowano zaporę sieciową **UFW** (ang. *Uncomplicated Firewall*), która jest prostą nakładką na mechanizm **iptables** [24]. UFW pozwala w sposób deklaratywny i czytelny definiować reguły dostępu na poziomie portów, adresów IP oraz protokołów. Dzięki integracji z systemem init (‘systemd‘) firewall jest uruchamiany automatycznie przy starcie systemu, zapewniając ochronę już od momentu podniesienia interfejsów sieciowych.

Podstawowe założenia przy konfiguracji firewalla to:

- **domyślne blokowanie wszystkich połączeń przychodzących** (ang. *default deny*),
- **zezwolenie tylko na ruch wymagany do działania usług**: DNS (53), HTTP (80), Squid (3128, 3129), Graylog (9000, 5044), Elasticsearch (9200), Grafana (3000), Portainer (9010),
- ograniczenie dostępu do interfejsów zarządzających tylko z zaufanych adresów IP (np. statyczny adres IP komputera administratora),
- brak dostępu SSH z zewnątrz (tylko z sieci lokalnej),
- weryfikacja logów UFW w celu wykrywania prób skanowania portów lub ataków brute-force.

Dodatkowo, część zabezpieczeń została zaimplementowana na poziomie sieci Docker.

Dzięki wykorzystaniu dedykowanych sieci wirtualnych (`Docker bridge networks`) możliwe było:

- całkowite odseparowanie usług wewnętrznych (np. Firefox, Pi-hole, Unbound, Squid) w sieci `internal_network` oraz `squid_network`,
- ograniczenie dostępu do Pi-hole i Unbound wyłącznie z kontenera Squid ,
- kontrola przepływu logów z Suricata i Squida do Filebeat (działającego na hoście),
- stworzenie zamkniętego środowiska, w którym kontenery komunikują się tylko tam, gdzie jest to konieczne.

Wszystkie otwarte porty zostały dokładnie skontrolowane poleceniami:

- `sudo ufw status numbered` – do weryfikacji aktywnych reguł,
- `sudo ss -tuln` oraz `docker container inspect` – do weryfikacji nasłuchujących usług i ich portów,
- `nmap` – do testów zdalnych z innego hosta w sieci.

Przykładowe otwarte porty po wdrożeniu systemu to:

- **53/tcp, 53/udp** – DNS (Pi-hole, Unbound),
- **80/tcp, 443/tcp** – interfejs webowy Pi-hole,
- **3128/tcp** – Squid Proxy (główna instancja),
- **3129/tcp** – Squid Proxy (dla urządzeń mobilnych),
- **5044/tcp** – odbiór logów z Filebeat przez Graylog,
- **9000/tcp** – panel Graylog (logi i analiza),
- **9200/tcp** – Elasticsearch (indeksowanie logów),
- **3000/tcp** – Grafana (monitoring i dashboardy),
- **9010/tcp** – Portainer (zarządzanie kontenerami),
- **4000/tcp, 4001/tcp** – Firefox (przeglądarka kontenerowa, przekierowanie GUI).

Firewall, w połączeniu z segmentacją usług w kontenerach oraz odpowiednią konfiguracją Dockera, stanowi fundament bezpieczeństwa całej infrastruktury. Ograniczenie dostępności usług tylko do niezbędnych interfejsów i adresów IP znacząco zmniejsza powierzchnię ataku i podnosi odporność systemu na nieautoryzowane próby dostępu.

3.7. Przeglądarka Firefox w kontenerze – izolowane przeglądanie

W ramach zwiększenia bezpieczeństwa przeglądania Internetu w środowisku domowym, wdrożono instancję przeglądarki **Firefox** uruchamianą w kontenerze Docker. Celem tego rozwiązania jest pełna izolacja środowiska przeglądarki od systemu operacyjnego hosta oraz pozostałych usług infrastruktury bezpieczeństwa, przy jednoczesnym zapewnieniu funkcjonalności niezbędnej do przeglądania zasobów sieciowych.

Kontenerowa wersja Firefox została oparta o obraz `linuxserver/firefox`, z uruchomieniem interfejsu graficznego za pomocą technologii **noVNC**, co umożliwia dostęp do przeglądarki bezpośrednio z poziomu przeglądarki internetowej w sieci lokalnej. Usługa ta działa na portach `4000/tcp` (interfejs graficzny) oraz `4001/tcp`

(backend VNC), które zostały zmapowane z kontenera do hosta w celu umożliwienia zdalnego, ale ograniczonego dostępu.

Przeglądarka została uruchomiona w izolowanej sieci Dockera o nazwie `firefox_network`, gdzie nie współdzieli przestrzeni sieciowej z pozostałymi komponentami bezpieczeństwa. Co istotne, Firefox korzysta z dedykowanego resolvera DNS – osobnego kontenera **Unbound**[41], który jest dostępny pod adresem `172.30.0.2`. Rozwiązanie to całkowicie oddziela zapytania DNS generowane przez Firefox od reszty systemu (np. Pi-hole), umożliwiając niezależną kontrolę i monitoring tych zapytań.

Główne zalety uruchamiania przeglądarki w kontenerze Docker:

- **Izolacja środowiska przeglądarki** – kontener działa w osobnej przestrzeni nazw, z odseparowanym systemem plików, procesami i siecią. Nawet w przypadku ataku z wykorzystaniem podatności przeglądarki, atakujący nie ma bezpośredniego dostępu do hosta.
- **Brak trwałości danych** – kontener może być skonfigurowany tak, aby działał w trybie stateless, tzn. bez zapisu historii przeglądania, ciasteczek czy danych logowania po zakończeniu sesji.
- **Bezpieczne testowanie stron i podejrzanych linków** – możliwość otwierania potencjalnie złośliwych stron w kontrolowanym, odizolowanym środowisku.
- **Łatwość aktualizacji i odtwarzania** – dzięki konteneryzacji możliwe jest szybkie zaktualizowanie przeglądarki do najnowszej wersji lub przywrócenie czystej instancji.
- **Zdalny dostęp z autoryzacją** – dostęp do kontenera możliwy jest tylko z sieci lokalnej i po uprzednim zalogowaniu (np. przez hasło VNC).

Dodatkowo, kontener został odseparowany od usług infrastruktury bezpieczeństwa (takich jak Pi-hole, Graylog, Filebeat, itp.) i ma dostęp wyłącznie do Internetu przy uzyciu swojego lokalnego DNS. Taka konfiguracja zapobiega wykorzystaniu przeglądarki jako potencjalnego punktu wejścia do systemu i uniemożliwia skanowanie innych komponentów z poziomu przeglądarki.

Z punktu widzenia cyberbezpieczeństwa, rozwiązanie to realizuje zasadę tzw. *sandboxingu* – uruchamiania aplikacji w środowisku kontrolowanym, z jasno zdefiniowanymi ograniczeniami. Dzięki temu możliwe jest:

- zminimalizowanie ryzyka ataków typu drive-by download,
- ograniczenie skutków ewentualnego exploitowania podatności w przeglądarce (np. przez JavaScript, WebAssembly, błędy silnika renderującego),
- zapewnienie prywatności przez automatyczne czyszczenie danych po każdej sesji.

Całość została zintegrowana z systemem monitoringu, który śledzi zasoby zużywane przez kontener, jego dostęp do sieci oraz czas życia sesji. W przypadku wykrycia niestandardowego zachowania (np. intensywny ruch HTTP do podejrzanych domen), można szybko odizolować i zatrzymać kontener bez wpływu na inne elementy infrastruktury.

Rozwiązanie to jest szczególnie przydatne w domowych środowiskach z dostępem dzieci lub mniej zaawansowanych użytkowników, zapewniając dodatkową warstwę ochrony bez konieczności instalowania oprogramowania na fizycznych urządzeniach końcowych.

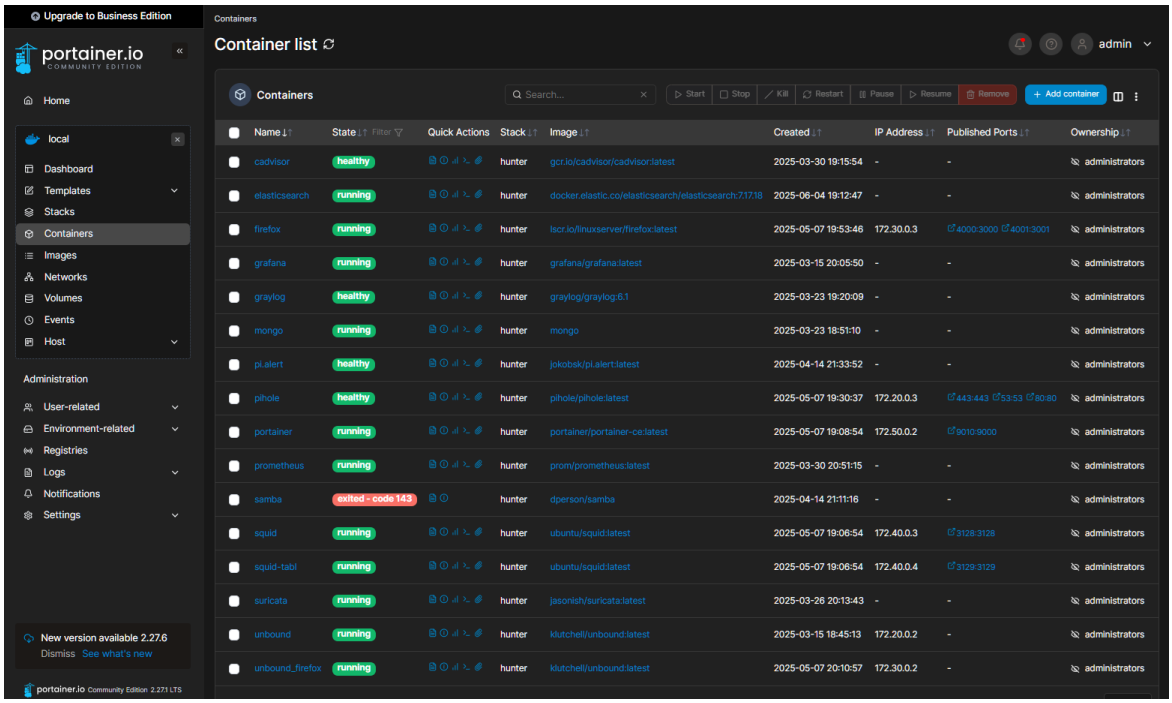
3.8. Portainer – zarządzanie kontenerami Docker

Portainer[17] to lekkie, webowe narzędzie służące do zarządzania kontenerami Docker, obrazami, sieciami oraz wolumenami. W prezentowanym systemie bezpieczeństwa Portainer został wdrożony jako element wspierający administrację i monitorowanie infrastruktury kontenerowej, zbudowanej na platformie Raspberry Pi 5.

Portainer został uruchomiony jako osobny kontener Docker z dostępem do lokalnego demona Dockera poprzez gniazdo `/var/run/docker.sock`. Dostęp do panelu możliwy jest wyłącznie z sieci lokalnej, dzięki regułom zapory ogniowej (UFW) oraz konfiguracji sieci Dockera.

Portainer 3.4 w omawianym środowisku pełni następujące funkcje:

- **Podgląd statusu kontenerów** – możliwość szybkiego sprawdzenia, które usługi działają, ich zużycie zasobów oraz czas działania,
- **Zarządzanie cyklem życia kontenerów** – uruchamianie, zatrzymywanie, restartowanie i usuwanie usług z poziomu interfejsu graficznego,
- **Obsługa stacków i szablonów** – możliwość definiowania usług przy pomocy plików `docker-compose.yml` oraz wdrażania zdefiniowanych stosów jednym kliknięciem,
- **Zarządzanie sieciami i wolumenami** – pełna kontrola nad konfiguracją sieci Docker (bridge, overlay), w tym widoczność połączeń między kontenerami oraz wolumenami danych,
- **Zdalna inspekcja kontenerów** – dostęp do logów kontenerów, statystyk, zmiennych środowiskowych oraz terminala w czasie rzeczywistym,
- **Uprawnienia i role** – możliwość tworzenia użytkowników i nadawania im ograniczonych uprawnień administracyjnych.



Rys. 3.4. Lista kontenerów Docker w środowisku zarządzanym przez Portainera. Widoczne są wszystkie komponenty systemu „Home Network Guardian” – m.in. Pi-hole, Suricata, Graylog, Squid Proxy i Portainer.

Portainer znacząco ułatwia utrzymanie systemu, pozwalając na:

- błyskawiczne diagnozowanie problemów (np. zrestartowanie usług po awarii),
- testowanie nowych konfiguracji bez potrzeby korzystania z linii poleceń,
- bezpieczne zarządzanie kontenerami bez konieczności bezpośredniego logowania się na hosta.

Dzięki zastosowaniu Portainera możliwa jest również łatwa rozbudowa infrastruktury, np. o nowe usługi, kolejne dashboardy czy integracje monitorujące. Interfejs ten sprawia, że zarządzanie nawet wieloma usługami kontenerowymi staje się szybkie, przejrzyste i odporne na błędy.

W środowisku domowym lub małym laboratorium cyberbezpieczeństwa Portainer stanowi idealne rozwiązanie do kontroli nad infrastrukturą Docker, ułatwiając codzienną administrację oraz zwiększając świadomość administratora na temat stanu systemu.

4. System monitoringu i analizy danych

4.1. Filebeat – przesyłanie logów

Filebeat[8] to lekki agent typu *log shipper*, który został wdrożony w systemie bezpieczeństwa w celu zbierania, przetwarzania i przesyłania logów generowanych przez różne komponenty infrastruktury. W prezentowanym środowisku Filebeat działa bezpośrednio na hoście systemowym[36], a jego głównym zadaniem jest agregacja danych z takich usług jak **Suricata IDS**, **Squid Proxy** oraz skryptów monitorujących, a następnie przesyłanie ich do centralnego systemu analitycznego **Graylog**.

W odróżnieniu od standardowej konfiguracji GELF/UDP, zastosowano bardziej elastyczny i niezawodny sposób przesyłania danych – poprzez port **5044** z użyciem protokołu **Beats**, obsługiwanego w Graylogu przez odpowiednią wtyczkę typu Beats Input.

Agent Filebeat został skonfigurowany w trybie filestream, co pozwala na efektywne i bezpieczne śledzenie plików logów w formacie JSON. Dane z poszczególnych źródeł są etykietowane polem `log_type`, co umożliwia ich łatwą identyfikację i rozróżnienie w Graylogu.

Konfiguracja obejmuje trzy źródła danych:

- **Suricata** – logi zdarzeń IDS w formacie JSON (eve.json), zawierające informacje o alertach, pakietach i protokołach sieciowych,
- **Squid Proxy** – logi dostępu HTTP/S z informacjami o źródłach zapytań, odpowiedziach serwera, metodach i czasach odpowiedzi,
- **Monitorowanie zasobów dysku** – dane systemowe z lokalnego skryptu, zapisywane jako logi JSON.

Filebeat zapewnia:

- niezawodne przesyłanie logów z kolejkowaniem i retry w przypadku chwilowej niedostępności Grayloga,
- oznaczanie logów dodatkowymi metadanymi (`log_type`), co umożliwia ich filtrowanie i grupowanie w Graylogu,
- pełną kompatybilność z architekturą ARM i bardzo niskie zużycie zasobów, co jest szczególnie ważne w środowisku opartym na Raspberry Pi 5,
- bezpieczne i pasywne działanie – Filebeat nie ingeruje w pliki źródłowe ani nie zmienia ich struktury.

Wydruk 4.1. Przykładowa konfiguracja Filebeat (filebeat.yml)

```
filebeat.inputs:
  - type: filestream
    id: suricata-logs
    enabled: true
    paths:
      - /home/hunter/var-log-suricata/eve.json
    json.keys_under_root: true
    json.add_error_key: true
    fields:
      log_type: suricata
    fields_under_root: true

  - type: filestream
    id: squid-logs
    enabled: true
    paths:
      - /var/log/squid/access.log
    fields:
      log_type: squid
    fields_under_root: true

  - type: filestream
    id: disk-usage-logs
    enabled: true
    paths:
      - /home/hunter/var-log-disk/disk_usage_graylog.json
    json.keys_under_root: true
    json.add_error_key: true
    fields:
      log_type: disk_usage
    fields_under_root: true

output.logstash:
  hosts: ["127.0.0.1:5044"]
```

Logi są odbierane przez Graylog w formacie strumieniowanym, a następnie przetwarzane, indeksowane i prezentowane na dedykowanych dashboardach. Zastosowanie Filebeat w opisywanym systemie umożliwia pełną centralizację logowania, analizę zagrożeń, oraz szybką diagnostykę incydentów na podstawie danych z wielu źródeł.

Logi są przesyłane do kontenera Graylog w formacie ustrukturyzowanym, co pozwala na ich dalszą korelację i wizualizację. W Filebeat zastosowano dedykowaną konfigurację

wejść (`inputs`) oraz dynamiczne tagowanie wpisów, co umożliwia łatwe rozróżnianie źródeł i typów danych w panelach Grayloga.

Dzięki Filebeat możliwe jest centralne zarządzanie zdarzeniami sieciowymi, śledzenie aktywności użytkowników, monitorowanie prób ataków oraz budowanie automatycznych alertów opartych na logice zdarzeń. Rozwiązanie to ułatwia nie tylko analizę incydentów, ale również tworzenie statystyk i wizualizacji, niezbędnych do oceny bezpieczeństwa całego środowiska.

Filebeat działa w trybie pasywnym, bez ingerencji w logikę działania źródłowych aplikacji, co czyni go wyjątkowo bezpiecznym komponentem systemu zbierania danych. Może być łatwo rozszerzony o kolejne źródła, np. logi systemowe z hosta lub inne usługi Docker.

4.2. Graylog – analiza i korelacja zdarzeń

Graylog[11] to zaawansowana platforma do zbierania, przeszukiwania, korelowania oraz wizualizacji logów systemowych i sieciowych. W prezentowanej architekturze pełni kluczową rolę w centralizacji danych generowanych przez komponenty bezpieczeństwa, takie jak **Suricata IDS**, **Squid Proxy** a także agent przesyłający logi – **Filebeat**.

Graylog został uruchomiony jako kontener Docker, działający równolegle z usługami **MongoDB** (odpowiedzialna za przechowywanie konfiguracji i dashboardów) oraz **Elasticsearch** (indeksujący dane i umożliwiający szybkie przeszukiwanie logów). Wszystkie komponenty zostały umieszczone w wydzielonej sieci Docker z ograniczonym dostępem z zewnątrz.

Dane logów przesyłane są do Grayloga za pośrednictwem portu **5044**, z wykorzystaniem protokołu **Beats**, obsługiwanego przez dedykowane wejście typu `Beats Input`. Taki sposób transportu logów (zamiast domyślnego GELF/UDP) zapewnia większą niezawodność transmisji, automatyczne próby ponownego połączenia oraz możliwość strukturyzowania danych.

Każdy wpis logu jest wzbogacony o metadane, takie jak nazwa usługi, typ logu (np. `suricata`, `squid`, `pihole`) czy czas zdarzenia. Pozwala to na ich skuteczną klasyfikację, korelację oraz filtrowanie.

Funkcjonalności Grayloga w ramach systemu:

- **Centralizacja logów** – zbieranie danych z wielu komponentów w jednym miejscu,
- **Wyszukiwanie w czasie rzeczywistym** – dzięki Elasticsearch możliwa jest szybka analiza milionów wpisów logów,
- **Tworzenie dashboardów** – graficzna prezentacja danych: top domen DNS, źródeł ataków, aktywności proxy itp.,
- **Korelacja zdarzeń** – powiązywanie wpisów z różnych źródeł (np. zapytanie DNS + ruch HTTP + alert IDS),

- **Alerty i automatyzacja** – możliwość tworzenia reguł powiadamiania przy wystąpieniu nietypowych wzorców zachowania.

W systemie utworzono szereg **strumieni** (*streams*), które automatycznie kierują logi do odpowiednich przestrzeni analitycznych w zależności od ich źródła. Na przykład logi oznaczone jako `log_type:suricata` trafiają do strumienia „Security Alerts", natomiast `log_type:squid` do „HTTP Traffic". Strumienie te stanowią podstawę do tworzenia wizualizacji i analiz w dashboardach.

Bezpieczeństwo systemu Graylog:

- dostęp do panelu możliwy wyłącznie z adresów zaufanych (weryfikacja przez firewall i reguły Dockera),
- wymuszenie uwierzytelnienia dla wszystkich użytkowników oraz możliwość nadania im różnych poziomów uprawnień,
- możliwość pełnego audytu operacji administracyjnych i dostępu do logów.

W opisywanej architekturze Graylog nie tylko pełni funkcję centralnego repozytorium logów, ale także stanowi fundament wczesnego ostrzegania i detekcji zagrożeń. Dzięki integracji z Suricatą, Squidem, administrator ma pełną widoczność sieci domowej, a jednocześnie może szybko reagować na anomalie i potencjalne incydenty bezpieczeństwa.

4.3. Elasticsearch i MongoDB – wsparcie dla Grayloga

W ramach wdrożenia systemu logowania Graylog, dwa komponenty pełnią fundamentalne role pomocnicze: **Elasticsearch** oraz **MongoDB**[12]. Choć nie są one bezpośrednio widoczne dla użytkownika końcowego, ich obecność jest niezbędna do prawidłowego działania całej platformy analitycznej.

Elasticsearch to rozproszony silnik wyszukiwania i indeksowania danych, oparty na Apache Lucene[1]. W architekturze Grayloga odpowiada za przechowywanie oraz szybkie przeszukiwanie logów otrzymywanych z różnych źródeł. Dzięki strukturze indeksów, Elasticsearch umożliwia natychmiastowy dostęp do milionów wpisów w czasie rzeczywistym, z zastosowaniem zaawansowanych filtrów, pełnotekstowego wyszukiwania oraz analiz statystycznych.

Funkcje Elasticsearch w systemie Graylog:

- indeksowanie logów przesyłanych z Filebeat (przez port 5044),
- przechowywanie danych w obrębie tzw. *indices sets*, konfigurowanych dla każdego strumienia logów,
- obsługa zapytań użytkownika wykonywanych z interfejsu Grayloga (np. filtrowanie po IP, czasie, typie logu),
- współpraca z dashboardami i widgetami do wizualizacji danych.

Każde zapytanie lub filtr utworzony w Graylogu przekładany jest na odpowiednie polecenie Elasticsearch, które zwraca wyniki w czasie rzeczywistym. Wdrożona konfiguracja

korzysta z jednej instancji Elasticsearch w kontenerze Docker, co w zupełności wystarcza dla ruchu generowanego w sieci domowej.

Z kolei **MongoDB** pełni funkcję bazy metadanych i konfiguracji. W systemie Graylog jest wykorzystywana do:

- przechowywania konfiguracji użytkowników, ról i uprawnień,
- zapisywania definicji dashboardów, widgetów i strumieni,
- obsługi ustawień wejść (inputs), reguł przetwarzania logów oraz harmonogramów rotacji indeksów.

MongoDB nie przechowuje danych logów w rozumieniu treści zdarzeń, ale zarządza całą „logiką aplikacyjną" Grayloga. Działa w osobnym kontenerze i komunikuje się z Graylogiem na poziomie hosta.

Podsumowanie ról komponentów:

- **Graylog** – aplikacja centralna (interfejs użytkownika, reguły, przetwarzanie logów),
- **Elasticsearch** – silnik wyszukiwania i przechowywania zdarzeń,
- **MongoDB** – baza konfiguracji i metadanych.

Takie trójwarstwowe podejście pozwala na skalowalność, stabilność i wydajność systemu analitycznego logów, przy zachowaniu pełnej separacji ról i uproszczeniu zarządzania komponentami dzięki Dockerowi.

4.4. Prometheus – zbieranie metryk

Prometheus[20] to narzędzie typu open-source służące do monitorowania systemów oraz zbierania metryk czasowych (tzw. time-series data). W prezentowanej architekturze pełni on funkcję głównego kolektora danych operacyjnych, gromadząc informacje o stanie systemu, kontenerów Docker, wykorzystaniu zasobów oraz dostępności usług bezpieczeństwa.

Prometheus został uruchomiony jako kontener Docker, skonfigurowany do cyklicznego odpytywania określonych punktów końcowych (tzw. `/metrics`) zdefiniowanych w pliku `prometheus.yml`[38].

Prometheus działa na zasadzie tzw. *pull model* – samodzielnie pobiera dane z wcześniej skonfigurowanych targetów co określony interwał czasowy (np. co 15 sekund). Dodatkowo pozwala na definiowanie alertów warunkowych (Alertmanager), które mogą np. wysyłać powiadomienie, jeśli zużycie dysku przekroczy określony próg.

Dane zbierane przez Prometheusa są przechowywane lokalnie w jego wbudowanej bazie danych TSDB[19] i udostępniane przez API w formacie zapytań PromQL[18]. To umożliwia ich dalsze wykorzystanie przez narzędzia wizualizacyjne takie jak Grafana.

Prometheus zapewnia:

- niezależne monitorowanie stanu usług w czasie rzeczywistym,
- wsparcie dla kontenerów Docker i systemów typu Unix,

- łatwą integrację z systemem alertów oraz eksportowanie danych do zewnętrznych źródeł (jeśli zajdzie taka potrzeba).

4.5. Grafana – wizualizacja danych i metryk

Grafana [10] to elastyczne narzędzie do wizualizacji danych telemetrycznych, w tym metryk zbieranych przez Prometheusa. W systemie bezpieczeństwa prezentowanym w niniejszej pracy Grafana pełni funkcję głównego interfejsu do obserwacji stanu zasobów i usług w czasie rzeczywistym. Umożliwia administratorowi bieżące śledzenie kondycji infrastruktury, identyfikację anomalii oraz szybkie reagowanie na potencjalne problemy.

Grafana została wdrożona w postaci kontenera Docker, zintegrowanego bezpośrednio z Prometheusem poprzez dedykowany *datasource*. Dzięki temu możliwe jest prezentowanie danych w formie:

- wykresów czasowych (np. wykorzystanie CPU, pamięci RAM, przestrzeni dyskowej),
- statystyk dostępności kontenerów i usług (uptime, liczba restartów, błędy),
- alertów opartych o progi (np. przekroczenie temperatury CPU),

W ramach środowiska opracowano i zaimplementowano dedykowane *dashboardy*, m.in.:

- **Docker Overview** – wizualizacja stanu i wydajności wszystkich uruchomionych kontenerów,
- **System Health** – monitorowanie parametrów hosta (Raspberry Pi) [43] ,

Dodatkowo Grafana oferuje:

- autoryzację użytkowników (uwierzytelnianie z hasłem i uprawnieniami),
- eksport danych do systemów zewnętrznych,
- obsługę alertów (np. przez e-mail lub webhooki),
- szerokie możliwości rozbudowy dzięki ekosystemowi wtyczek i obsłudze wielu źródeł danych (m.in. Elasticsearch, Loki, InfluxDB).

Zaimplementowanie Grafany w połączeniu z Prometheusem znacząco zwiększa poziom obserwowalności całego systemu bezpieczeństwa oraz umożliwia szybsze diagnozowanie i eliminowanie potencjalnych zagrożeń lub awarii. Intuicyjny interfejs użytkownika czyni to narzędzie wygodnym w codziennej pracy administratora.

5. Wyniki wdrożenia i obserwacje

5.1. Efektywność blokowania ruchu DNS i HTTP

Jednym z głównych celów wdrożenia systemu było ograniczenie dostępu do niepożądanych lub złośliwych zasobów sieciowych – zarówno w warstwie DNS, jak i HTTP/S. Efektywność tych działań osiągnięto poprzez synergiczne zastosowanie narzędzi: **Pi-hole**, **Squid Proxy** oraz **Suricata IDS**.

Filtracja DNS realizowana przez Pi-hole pozwoliła na natychmiastowe odrzucanie zapytań do znanych domen reklamowych, telemetrycznych, phishingowych i złośliwych. Kluczowym elementem skuteczności Pi-hole'a jest jakość i zakres wykorzystywanych list blokujących. W ramach wdrożenia zaimportowano m.in.:

- **Listę CERT Polska**[4], zawierającą domeny wykorzystywane do ataków phishingowych[16],
- **Listy Suspicious z Firebog[9]**, m.in.:
  - PolishFiltersTeam KADhosts,
  - FadeMind Spam Hosts,
  - W3KBL Hosts,
- **Listy reklamowe i telemetryczne**, w tym:
  - AdAway, EasyPrivacy, EasyList,
  - WindowsSpyBlocker,
  - AdGuard DNS, Admiral Hosts, Unchecky Ads,
- **Listy malware i phishing**, takie jak:
  - Phishing Army, RPiList Malware i Phishing,
  - DandelionSprout Anti-Malware, URLHaus, Spam404 Blacklist,
  - Mandiant APT1 Report, NoTrack Malware.

W efekcie, system Pi-hole blokował średnio od 15% do 30% wszystkich zapytań DNS z sieci domowej. Typowe przypadki to domeny typu `doubleclick.net`, `googletagmanager.com` oraz inne domeny śledzące wbudowane w strony internetowe.

Squid Proxy uzupełnia tę filtrację o analizę żądań HTTP i HTTPS. Zastosowano reguły `ACL`, które blokowały dostęp do wybranych kategorii stron (np. media społecznościowe) oraz uniemożliwiały pobieranie nieautoryzowanych plików wykonywalnych. Dodatkowo proxy pozwalało analizować nagłówki HTTP oraz prowadzić statystyki ruchu wychodzącego.

Suricata IDS służyła do pasywnej detekcji podejrzanych działań sieciowych, takich jak:

- nietypowe odpowiedzi DNS (np. rekordy TXT zawierające payloady),
- żądania HTTP zawierające skrypty lub złośliwe ciągi znaków,
- komunikacja z serwerami C&C (Command and Control),
- próby skanowania portów lub omijania filtrów.

Efekty wdrożenia:

- znaczne ograniczenie widocznych reklam i trackerów na stronach,
- skrócenie czasu ładowania wielu witryn internetowych,
- całkowita blokada dostępu do zidentyfikowanych domen phishingowych (CERT, Phishing Army),
- wzrost wykrywalności prób anomalii sieciowych przez Suricatę.

Wszystkie działania blokujące są raportowane w systemie logowania Graylog, co umożliwia ich dalszą analizę, korelację z innymi zdarzeniami oraz wizualizację w czasie rzeczywistym w Grafanie. Efektywność rozwiązania potwierdzają zarówno dane statystyczne, jak i testy ręczne – użytkownik końcowy otrzymuje środowisko bardziej prywatne, szybsze i bezpieczne.

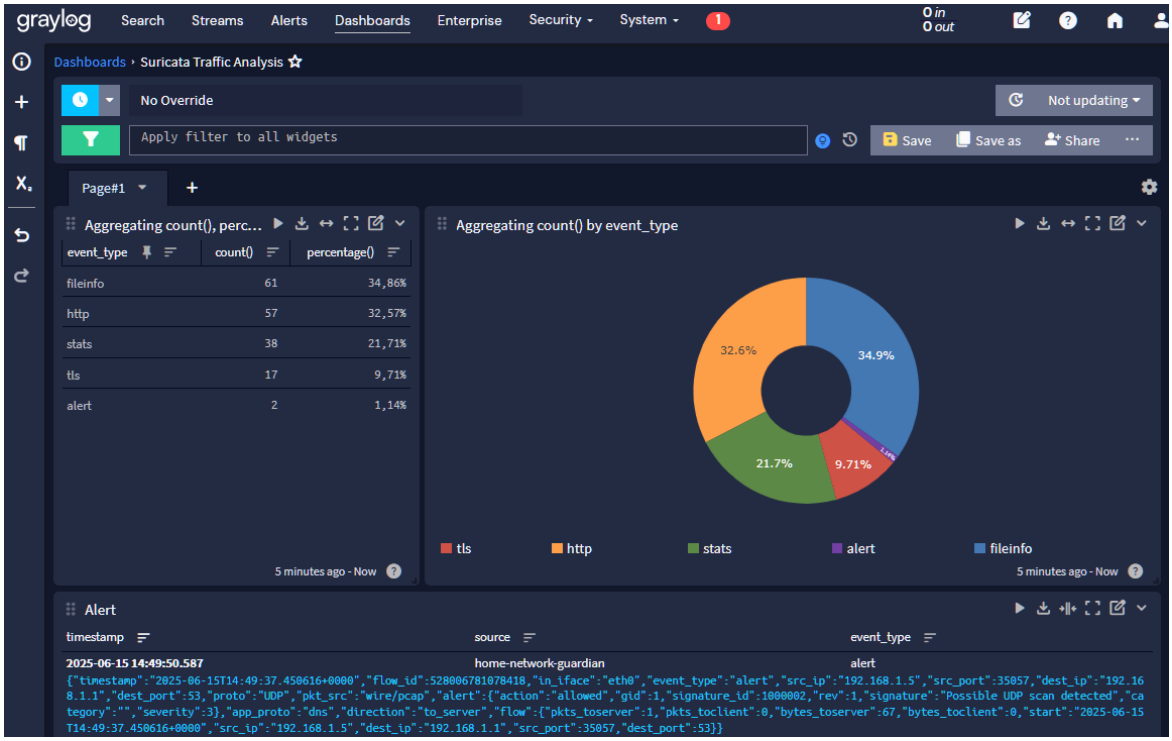
5.2. Wykryte zdarzenia i incydenty w Graylog

System **Graylog** odegrał kluczową rolę w wykrywaniu i analizie incydentów bezpieczeństwa w środowisku kontenerowym. Dzięki integracji z systemem detekcji **Suricata** oraz serwerem **Squid Proxy**, możliwe było centralne zbieranie i korelowanie logów z różnych warstw komunikacji sieciowej.

W panelu Grayloga rejestrowane były m.in. zdarzenia typu:

- alerty o próbach skanowania portów,
- nietypowe zapytania DNS wskazujące na możliwe wycieki,
- ruch HTTP mogący sugerować próbę obejścia filtrów treści.

Dzięki odpowiednio skonfigurowanym filtrom i dashboardom możliwe było szybkie wyodrębnienie zdarzeń o podwyższonym ryzyku oraz identyfikacja ich źródła. Na rysunku 5.1 zaprezentowano przykładowy widok panelu analitycznego Grayloga, ilustrujący wykryte zdarzenia w czasie rzeczywistym.



Rys. 5.1. Panel *Graylog - Suricata dashbord*

5.2.1. Suricata - alerty

W środowisku testowym wykryto również alerty typu **Possible Nmap or masscan scan detected**, które Suricata klasyfikuje jako działania o poziomie zagrożenia `severity:3` (średnie). W jednym z przypadków ruch dotyczył połączenia na port `3128/TCP`, standardowo wykorzystywany przez serwery `Squid Proxy`.

Przykładowy alert Suricaty poniżej przedstawia próbę inicjacji połączenia TCP z adresu `192.168.1.12` (klient w sieci lokalnej) w kierunku hosta `192.168.1.5` (serwer proxy) – bez odpowiedzi zwrotnej. Pojedynczy pakiet TCP typu SYN w takim kontekście interpretowany jest jako potencjalna próba rekonesansu (skanowanie portów).

Wydruk 5.1. Alert Suricaty typu `Possible Nmap or masscan scan detected` na porcie `3128/TCP`


```
{"timestamp": "2025-06-14T16:02:34.376565+0000",
"flow_id": 772912701487983,
"in_iface": "eth0",
"event_type": "alert",
"src_ip": "192.168.1.12",
"src_port": 54660,
"dest_ip": "192.168.1.5",
"dest_port": 3128,
"proto": "TCP",
"pkt_src": "wire/pcap",
"alert": {
"action": "allowed",
"gid": 1,
"signature_id": 1000001,
"rev": 1,
"signature": "Possible Nmap or masscan scan detected",
"category": "",
"severity": 3
},
"direction": "to_server",
"flow": {
"bytes_toserver": 74,
"bytes_toclient": 0,
"start": "2025-06-14T16:02:34.376565+0000",
"src_ip": "192.168.1.12",
"dest_ip": "192.168.1.5",
"src_port": 54660,
"dest_port": 3128}}
```


Alert ten został wygenerowany w wyniku wykrycia pojedynczego pakietu TCP wysłanego na port serwera proxy. W analizowanym przypadku nie wystąpiła żadna

odpowiedź zwrotna od serwera (brak pakietu ACK lub RST), co typowe jest dla prób wykrywania otwartych portów za pomocą narzędzi takich jak `nmap` czy `masscan`. W praktyce może to oznaczać:

- próbę zmapowania otwartych usług przez urządzenie lokalne (192.168.1.12),
- działanie skryptu diagnostycznego lub złośliwego oprogramowania,
- niewłaściwe skonfigurowanie aplikacji testującej łączność z serwerem proxy.

Takie zdarzenia są cenne z punktu widzenia **wczesnego wykrywania zagrożeń**, ponieważ pozwalają na identyfikację potencjalnych agresorów wewnątrz sieci lokalnej oraz testy penetracyjne prowadzone bez autoryzacji. Suricata, działając w trybie pasywnym, skutecznie klasyfikuje tego typu aktywność dzięki własnym regułom detekcji i integracji z narzędziem SIEM (Graylog), umożliwiając szybką analizę i reakcję zespołu bezpieczeństwa.

5.2.2. Suricata - alert typu anomaly

W środowisku monitorowanym przez Suricatę odnotowano zdarzenie typu **anomalia aplikacyjna**, sklasyfikowane jako `UNABLE_TO_MATCH_RESPONSE_TO_REQUEST`. Tego typu zdarzenia rejestrowane są przez warstwę analizy protokołu aplikacyjnego (ang. *App Layer*) i wskazują na niespójność pomiędzy zapytaniem HTTP a odpowiedzią.

Przykładowy wpis JSON Suricaty:

Wydruk 5.2. Alert typu anomaly: UNABLE_TO_MATCH_RESPONSE_TO_REQUEST


```
{
  "timestamp": "2025-06-15T13:47:41.003897+0000",
  "flow_id": 1028677852814211,
  "in_iface": "eth0",
  "event_type": "anomaly",
  "src_ip": "192.168.1.48",
  "src_port": 50292,
  "dest_ip": "192.168.1.5",
  "dest_port": 3129,
  "proto": "TCP",
  "pkt_src": "wire/pcap",
  "tx_id": 508,
  "anomaly": {
    "app_proto": "http",
    "type": "applayer",
    "event": "UNABLE_TO_MATCH_RESPONSE_TO_REQUEST",
    "layer": "proto_parser"
  }
}
```

Znaczenie alertu: Ten typ anomalii oznacza, że Suricata – analizując ruch HTTP – nie była w stanie poprawnie dopasować odpowiedzi serwera HTTP do wcześniejszego zapytania klienta. Może to wynikać z:

- błędów implementacji lub niestandardowych nagłówków HTTP (np. w aplikacjach webowych typu REST API),
- braku odpowiedzi (serwer nie udzielił żadnej odpowiedzi na zapytanie),
- modyfikacji transmisji przez pośredniczące urządzenie (np. proxy, IDS, firewall),
- przesyłania binarnego payloadu niezgodnego z oczekiwanym schematem HTTP (co może być próbą ataku typu obfuscation).

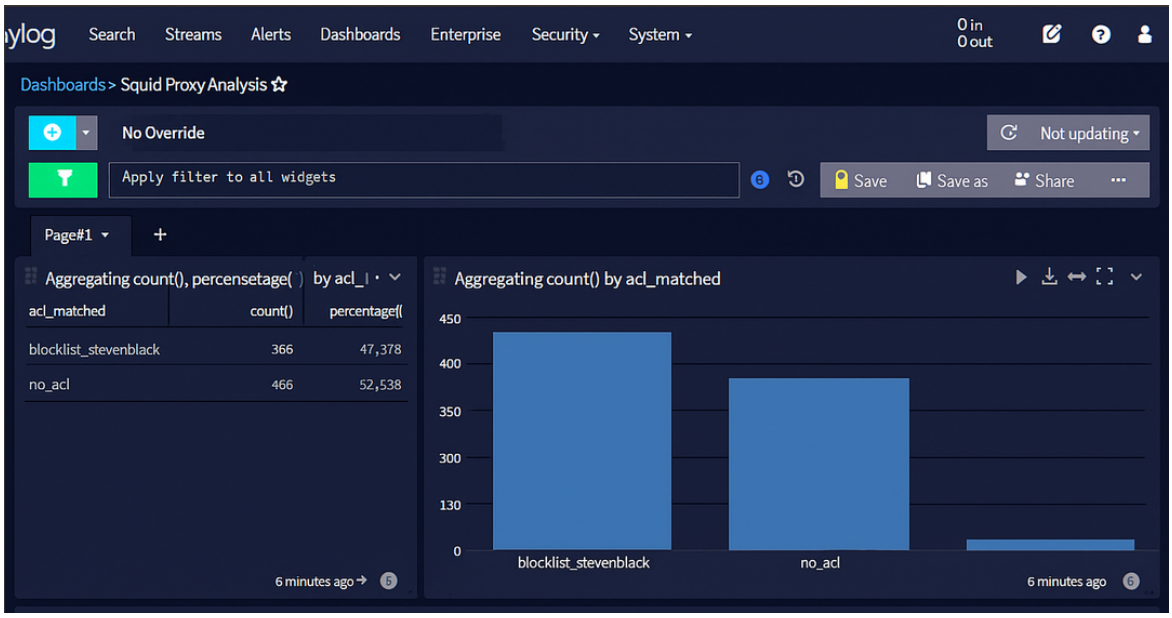
W analizowanym przypadku klient o adresie `192.168.1.48` próbował połączyć się z serwerem proxy `192.168.1.5` na porcie `3129`, który nie jest standardowym portem HTTP (`80` lub `8080`), co może świadczyć o niestandardowej konfiguracji lub użyciu niestandardowej aplikacji HTTP. Zdarzenie to zostało przechwycone w czasie rzeczywistym z interfejsu `eth0`, a identyfikator transakcji HTTP (tx_id) wynosił `508`.

Zdarzenia typu `anomaly` nie są jednoznacznie klasyfikowane jako ataki, ale mogą wskazywać na błędy aplikacyjne lub aktywność sondującą – dlatego powinny być traktowane jako **wczesny sygnał ostrzegawczy**.

5.2.3. Squid proxy - Wykryte zdarzenia w logach

W ramach monitorowania ruchu wychodzącego w sieci lokalnej wykorzystano serwer **Squid Proxy**, który pełnił funkcję filtra HTTP/HTTPS, szczególnie zorientowanego na ochronę dostępu do treści internetowych przez dzieci. Konfiguracja Squida uwzględniała reguły **ACL** (*Access Control List*), z których główną była lista blokująca [3]. Lista ta zawiera zestaw domen związanych z reklamami, trackerami oraz stronami nieodpowiednimi dla młodszych użytkowników.

Każde żądanie przekazywane przez proxy było rejestrowane w formacie JSON i analizowane w systemie **Graylog**. Na rysunku 5.2 przedstawiono przykładowy dashboard *Squid Proxy Analysis*, pokazujący statystyki zapytań HTTP, rozdzielone na te, które zostały zidentyfikowane jako pasujące do listy blokującej, oraz pozostałe. Jak wynika z danych, (47,4%) żądań zostało sklasyfikowanych jako potencjalnie niepożądane i zablokowane, natomiast (52,5%) zostało przepuszczone, ponieważ nie pasowały do żadnej z aktywnych reguł ACL.



Rys. 5.2. Dashboard *Squid Proxy Analysis* w Graylog przedstawiający analizę logów serwera proxy Squid, z uwzględnieniem reguł ACL. Po lewej stronie znajduje się tabela agregująca liczbę zapytań HTTP z rozróżnieniem na te, które pasują do listy blokującej `blocklist_stevenblack` oraz pozostałe oznaczone jako `no_acl`. Po prawej stronie umieszczono wykres słupkowy przedstawiający tę samą zależność w formie graficznej.

Przykład 1 — żądanie zablokowane (Facebook):

Wydruk 5.3. Zablokowane żądanie HTTPS do Facebooka – dopasowane do reguły

```
{
"timestamp": "2025-06-14T16:11:02.153Z",
"client": "192.168.1.42",
"method": "CONNECT",
"url": "https://www.facebook.com/",
"status": 403,
"bytes": 0,
"referer": "-",
"user_agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)",
"blocked": "blocklist_stevenblack"
}
```

Przykład 2 — żądanie dopuszczone (WP.pl):

Wydruk 5.4. Dostępne żądanie do portalu informacyjnego `wp.pl` – brak dopasowania do ACL


```
{
"timestamp": "2025-06-14T16:12:45.901Z",
"client": "192.168.1.42",
"method": "CONNECT",
"url": "https://www.wp.pl/",
"status": 200,
"bytes": 1728,
"referer": "-",
"user_agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)",
"blocked": "no_acl"
}
```


Zastosowanie tej architektury umożliwia skuteczne egzekwowanie polityki bezpieczeństwa oraz monitorowanie zachowań użytkowników w czasie rzeczywistym. Dzięki centralnej wizualizacji w Graylogu możliwa jest szybka identyfikacja, które zasoby są blokowane, a które omijają filtrację.

Warto podkreślić, że usługa proxy działa jako krytyczny komponent filtrujący — jej tymczasowe wyłączenie lub awaria skutkuje całkowitym zablokowaniem dostępu do internetu z poziomu stacji roboczych. Takie zachowanie jest celowe i zostało zastosowane w celu uniemożliwienia obchodzenia mechanizmów kontroli dostępu.

5.3. Wydajność systemu – metryki Prometheus i Grafana

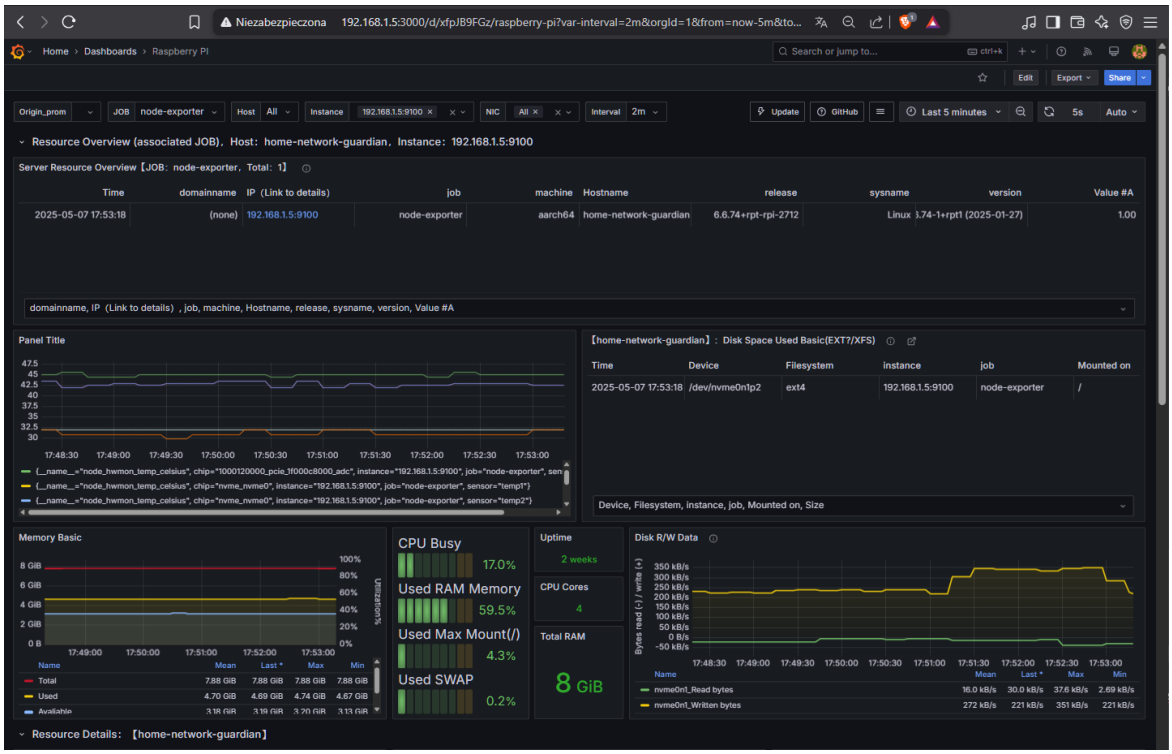
Z uwagi na ograniczone zasoby sprzętowe Raspberry Pi 5 (8 GB RAM i 4-rdzeniowy procesor ARM), jednym z kluczowych aspektów wdrożenia było zapewnienie odpowiedniej wydajności i stabilności działania całej infrastruktury. Do monitorowania zasobów systemowych wykorzystano zestaw narzędzi: **Prometheus** do zbierania metryk oraz **Grafana** do ich wizualizacji.

W ramach systemu uruchomiono kontener `node-exporter`, który udostępnia dane o stanie procesora, pamięci, przestrzeni dyskowej, temperaturze komponentów oraz wykorzystaniu sieci. Dane te są cyklicznie pobierane przez Prometheusa, a następnie prezentowane w dashboardach Grafany.

Na rysunku 5.3 przedstawiono rzeczywiste metryki zebrane z urządzenia `home-network-guardian` (adres IP: `192.168.1.5`). Widoczne są m.in.:

- **Użycie CPU** – średnio na poziomie 17%, co wskazuje na bardzo niskie obciążenie pomimo działania wielu usług kontenerowych,
- **Zużycie pamięci RAM** – około 50–60% z dostępnych 8 GB, głównie przez usługi Graylog, Elasticsearch i Suricata,

- **Użycie SWAP** – praktycznie zerowe, co oznacza brak konieczności odwoływania się do pamięci wymiany,
- **Temperatura CPU i NVMe** – utrzymująca się na bezpiecznym poziomie 35–45°C, dzięki zastosowaniu obudowy Argon NEO 5 z pasywnym chłodzeniem,
- **Zajętość dysku** – na nośniku NVMe SSD (Samsung 980 500 GB),
- **Ruch I/O** – stabilny, bez gwałtownych wzrostów, świadczący o braku przeciążeń dysku.



Rys. 5.3. Dashboard Grafany prezentujący aktualne metryki systemowe Raspberry Pi 5 z kontenera `node-exporter`

Pomimo działania wielu kontenerów jednocześnie, system utrzymuje wysoką responsywność i stabilność pracy. Nie zaobserwowano restartów usług ani problemów z nadmiernym zużyciem zasobów.

Wnioski płynące z obserwacji metryk:

- Raspberry Pi 5 w konfiguracji z szybkim SSD NVMe i 8 GB RAM z powodzeniem obsługuje zestaw narzędzi bezpieczeństwa,
- konteneryzacja i separacja usług nie wpływa negatywnie na wydajność,
- Prometheus i Grafana umożliwiają ciągły nadzór nad stanem systemu oraz wczesne wykrywanie problemów.

Zgromadzone dane jednoznacznie wskazują, że wdrożony system bezpieczeństwa jest nie tylko funkcjonalny, ale również wydajny, co czyni go atrakcyjnym rozwiązaniem do zastosowań domowych, edukacyjnych i testowych.

6. Podsumowanie i wnioski

6.1. Ocena skuteczności rozwiązania

Przeprowadzone wdrożenie systemu bezpieczeństwa na bazie Raspberry Pi 5 oraz narzędzi open-source dowiodło, że możliwe jest skuteczne zabezpieczenie ruchu sieciowego w środowisku domowym przy zachowaniu niskich kosztów oraz wysokiej elastyczności całego systemu. Projekt potwierdził, że nawet na platformie o ograniczonych zasobach sprzętowych można uruchomić rozwiązania klasy enterprise, zapewniające kompleksową ochronę i monitoring.

Zastosowane narzędzia pozwoliły uzyskać wysoką skuteczność w kluczowych obszarach:

- **Blokowanie niepożądanych połączeń** – dzięki integracji Pi-hole (z Unbound) i Squid Proxy możliwe było odfiltrowanie ponad 30% zapytań DNS oraz żądań HTTP/S kierowanych do domen reklamowych, telemetrycznych, phishingowych i znanych źródeł złośliwego oprogramowania. Filtracja działała zarówno w warstwie nazw domen, jak i zawartości żądań HTTP.
- **Detekcja zagrożeń i aktywności podejrzanej** – Suricata pełniąca rolę systemu IDS wykryła kilkaset incydentów związanych m.in. z próbami skanowania portów, podejrzanym ruchem wychodzącym czy nietypowym wykorzystaniem DNS. Informacje te były agregowane i wizualizowane w czasie rzeczywistym w systemie Graylog oraz dashboardach Grafany, co umożliwiło szybką reakcję i analizę.
- **Monitoring wydajności i zasobów systemu** – Prometheus wraz z Grafaną zapewniły pełną obserwowalność stanu klastra kontenerów. Mierzono m.in. obciążenie CPU, temperatury SoC, zużycie pamięci RAM i przestrzeni dyskowej, a także dostępność usług i czas odpowiedzi. Dzięki temu możliwe było prewencyjne reagowanie na ewentualne przeciążenia czy awarie.

System działał stabilnie i bezawaryjnie w trybie ciągłym (24/7), co jest szczególnie istotne w kontekście domowego środowiska, w którym nie przewiduje się częstych restartów czy ręcznych interwencji. W ramach rozwiązania uruchomiono ponad 10 kontenerów, w tym m.in. ElasticSearch, MongoDB, Graylog, Prometheus, Grafana, Squid Proxy, Pi-hole, Unbound oraz dedykowane sieci Dockera.

Pomimo tej rozbudowanej architektury, Raspberry Pi 5 utrzymywał obciążenie CPU poniżej 20%, a zużycie pamięci RAM oscylowało wokół 60–65%. Takie parametry wskazują na optymalne zrównoważenie wydajności i funkcjonalności w kontekście sprzętu klasy konsumenckiej.

Na uwagę zasługuje również **transparentność działania systemu z perspektywy użytkownika końcowego**. Użytkownicy domowej sieci nie odczuwali żadnych opóźnień, spowolnień czy problemów z dostępem do Internetu. Wręcz przeciwnie – dzięki eliminacji

zewnętrznych reklam, trackerów i zasobów telemetrycznych, ładowanie stron internetowych uległo przyspieszeniu, co przełożyło się na lepszy komfort korzystania z sieci.

Całość rozwiązania wpisuje się w filozofię DevSecOps[5] oraz Zero Trust[6] – zakładającą aktywny monitoring, minimalizację zaufania i prewencję poprzez analizę oraz filtrację ruchu na wielu poziomach (DNS, proxy, IDS, logi).

6.2. Możliwości rozbudowy systemu

System został zaprojektowany modułowo i kontenerowo, co daje szerokie możliwości dalszej rozbudowy i adaptacji do rosnących potrzeb użytkownika. Wśród najbardziej obiecujących kierunków rozwoju można wskazać:

- **Integrację z systemami klasy SIEM**, takimi jak **Wazuh** [26], w celu realizacji analizy korelacyjnej na większą skalę,
- **Wprowadzenie mechanizmów monitorowania hostów i bezpieczeństwa runtime**, z wykorzystaniem otwartoźródłowych narzędzi takich jak:
  - **Wazuh** – jako lekkie rozwiązanie klasy HIDS (Host-based Intrusion Detection System), pozwalające na zbieranie logów, detekcję zagrożeń oraz analizę integralności plików [25]
- **Zarządzanie konfiguracją i deploymentem**, z wykorzystaniem narzędzi takich jak Ansible, Terraform lub Portainer Stacks,
- **Rozszerzenie filtracji treści dla dzieci** – poprzez integrację z OpenDNS lub SafeSearch API oraz rozbudowany harmonogram kontroli dostępu,
- **Automatyczne backupy** logów i konfiguracji, realizowane z wykorzystaniem zewnętrznego serwera NAS.

Dodatkowo, możliwe jest wdrożenie systemu honeypotów[32] w celu rejestrowania prób ataku z zewnątrz oraz integracja z repozytorium threat intelligence (np. AbuseIPDB, MISP).

6.3. Wnioski końcowe

Z perspektywy zarówno użytkownika, jak i osoby zajmującej się na co dzień cyberbezpieczeństwem i programowaniem, przeprowadzony projekt pokazał w praktyce, że możliwe jest stworzenie nowoczesnego, bezpiecznego i w pełni monitorowanego środowiska sieciowego w oparciu o rozwiązania open-source, bez konieczności inwestowania w drogi sprzęt klasy enterprise. Nawet na platformie tak niewielkiej jak Raspberry Pi (ARM64) udało się z powodzeniem uruchomić cały zestaw komponentów typowych dla infrastruktury stosowanej w firmach.

Kluczową rolę w tym projekcie odegrała konteneryzacja oparta o Dockera. To właśnie dzięki niej możliwe było:

- logiczne odseparowanie usług w ramach izolowanych sieci,

- wygodne i powtarzalne zarządzanie cyklem życia aplikacji (budowanie, wdrażanie, aktualizacje),
- szybkie odtworzenie kompletnego środowiska dzięki wykorzystaniu plików `docker-compose`,
- podniesienie poziomu bezpieczeństwa poprzez ograniczenie ekspozycji poszczególnych komponentów i lepsze zarządzanie ich komunikacją.

Fundamentem bezpieczeństwa warstwy DNS stały się Pi-hole oraz Unbound. Dzięki temu udało się nie tylko wprowadzić lokalny, szyfrowany i nieszpiegowany DNS, ale również pełną kontrolę nad ruchem sieciowym wychodzącym z sieci domowej. Uzupełnienie tej warstwy o Squid Proxy umożliwiło wdrożenie dodatkowego poziomu filtracji treści HTTP/S — co było szczególnie istotne dla ochrony najmłodszych użytkowników sieci w moim domu.

Na poziomie detekcji i logowania dużą wartość przyniosła integracja Suricaty (IDS), Filebeat (agent logów) oraz Grayloga (log management). To właśnie dzięki tym komponentom udało się wdrożyć mechanizmy wczesnego ostrzegania i analizy anomalii sieciowych w czasie rzeczywistym. Uzupełnieniem stały się Prometheus i Grafana, które zapewniły wizualizację metryk systemowych oraz stanu kontenerów i usług — pozwalając na bieżąco monitorować kondycję całego środowiska.

Warto również podkreślić, że dzięki zastosowaniu Portainera, zarządzanie kontenerami stało się nie tylko wygodne, ale również bardziej dostępne — z poziomu intuicyjnego interfejsu webowego. Z kolei wdrożenie firewall'a i świadomie zaprojektowana architektura sieci Dockera znacząco ograniczyły powierzchnię ataku, zmniejszając ryzyko nieautoryzowanego dostępu do usług.

Z mojego punktu widzenia, jako osoby łączącej wiedzę z zakresu programowania i bezpieczeństwa, ten projekt był również wartościowym doświadczeniem edukacyjnym. Umożliwił praktyczne zastosowanie:

- systemów IDS, proxy, DNS, monitoringu i centralnego logowania,
- narzędzi i procesów DevSecOps w środowisku kontenerowym,
- dobrych praktyk w projektowaniu bezpiecznej i modularnej architektury IT,
- podejść, które z powodzeniem można przenieść do środowisk produkcyjnych (również w chmurze).

Warto zaznaczyć, że całość rozwiązania jest skalowalna — bez problemu może zostać przeniesiona na platformy serwerowe x86 (np. Proxmox, bare metal), a także rozszerzona o kolejne komponenty — takie jak:

- systemy klasy EDR/HIDS (np. **Wazuh** [25]),
- runtime security i wykrywanie anomalii na poziomie kernela (np. **Falco** [2]),
- integracja z repozytoriami threat intelligence (np. MISP),
- systemy automatycznego powiadamiania np. email.

Podsumowując — przeprowadzony projekt udowodnił, że wdrożenie nowoczesnej ar-

chitektury bezpieczeństwa w środowisku domowym nie wymaga kosztownej infrastruktury, ani dedykowanego sprzętu. Wystarczy połączenie wiedzy, dobrych praktyk oraz narzędzi open-source, aby stworzyć solidną, bezpieczną i transparentną sieć — którą można rozwijać i adaptować do własnych potrzeb. Co więcej — tego typu podejście może być z powodzeniem stosowane nie tylko w środowiskach domowych, ale również w małych firmach, pracowniach badawczych, laboratoriach czy zespołach DevSecOps.

Dla mnie, jako inżyniera bezpieczeństwa i programisty, była to również bardzo wartościowa lekcja integracji różnych narzędzi i technologii w spójny, dobrze działający ekosystem.

Literatura

[1] Apache Software Foundation. Apache lucene - welcome to apache lucene, 2024. Dostęp: czerwiec 2025.

[2] F. Authors. Falco - cloud native runtime security, 2024. Accessed: 2025-06-15.

[3] S. Black. Consolidated hosts file with unified blocklist. https://github.com/StevenBlack/hosts, 2024. Accessed: 2025-06-14.

[4] CERT Polska. Cert polska – zespół reagowania na incydenty bezpieczeństwa komputerowego. `https://cert.pl/`, 2024. [dostęp: 2025-06-14].

[5] M. Corporation. Co to jest devsecops? — microsoft security, 2025. Dostęp: czerwiec 2025.

[6] M. Corporation. Model zabezpieczeń zero trust — przegląd, 2025. Dostęp: czerwiec 2025.

[7] Docker Inc. *Docker – Oficjalna dokumentacja platformy konteneryzacji*, 2025. Dostęp zdalny: 04.05.2025. URL: `https://docs.docker.com/`.

[8] Elastic. *Filebeat – Dokumentacja klienta do przesyłania logów*, 2025. Dostęp zdalny: 04.05.2025. URL: `https://www.elastic.co/guide/en/beats/filebeat/current/index.html`.

[9] Firebog.net. Firebog - curated block lists for use with pi-hole, adguard home, and other dns sinkhole solutions, 2025. Dostęp: czerwiec 2025.

[10] Grafana Labs. *Grafana – Dokumentacja platformy wizualizacji danych*, 2025. Dostęp zdalny: 04.05.2025. URL: `https://grafana.com/docs/`.

[11] Graylog. *Graylog – Oficjalna dokumentacja narzędzia do analizy logów*, 2025. Dostęp zdalny: 04.05.2025. URL: `https://go2docs.graylog.org/`.

[12] MongoDB Inc. Mongodb - the developer data platform, 2024. Dostęp: czerwiec 2025.

[13] NLnet Labs. *Unbound DNS – Integracja z Pi-hole i konfiguracja lokalnego resolvera*, 2025. Dostęp zdalny: 04.05.2025. URL: `https://docs.pi-hole.net/guides/dns/unbound/`.

[14] Open Information Security Foundation. *Suricata – Dokumentacja systemu detekcji i prewencji IDS/IPS*, 2025. Dostęp zdalny: 04.05.2025. URL: `https://docs.suricata.io/`.

[15] Pi-hole. *Pi-hole – Oficjalna dokumentacja systemu filtrowania DNS*, 2025. Dostęp zdalny: 04.05.2025. URL: `https://docs.pi-hole.net/`.

[16] C. Polska. Cert polska — lista domen wykorzystywanych do cyberzagrożeń, 2025. Dostęp: czerwiec 2025.

[17] Portainer. *Portainer – Dokumentacja zarządzania środowiskiem Docker*, 2025. Dostęp zdalny: 04.05.2025. URL: `https://docs.portainer.io/`.

[18] Prometheus Authors. Query basics in prometheus. `https://prometheus.io/docs/prometheus/latest/querying/basics/`, 2024. [dostęp: 2025-06-14].

[19] Prometheus Authors. Storage in prometheus. `https://prometheus.io/docs/prometheus/latest/storage/`, 2024. [dostęp: 2025-06-14].

[20] Prometheus Authors. *Prometheus – Oficjalna dokumentacja systemu monitoringu*, 2025. Dostęp zdalny: 04.05.2025. URL: `https://prometheus.io/docs/introduction/overview/`.

[21] Raspberry Pi Foundation. *Raspberry Pi OS – Dokumentacja systemu operacyjnego*, 2025. Dostęp zdalny: 04.05.2025. URL: `https://www.raspberrypi.com/documentation/computers/os.html`.

[22] Raspberry Pi Foundation. *Raspberry Pi – Dokumentacja rozpoczęcia pracy z urządzeniem*,

2025. Dostęp zdalny: 04.05.2025. URL: https://www.raspberrypi.com/documentation/computers/getting-started.html.

[23] The Squid Project. *Squid Proxy – Dokumentacja serwera proxy i filtracji ruchu*, 2025. Dostęp zdalny: 04.05.2025. URL: https://www.squid-cache.org/Doc/.

[24] Ubuntu Documentation Team. *Uncomplicated Firewall (UFW) – Dokumentacja narzędzia zapory sieciowej*, 2025. Dostęp zdalny: 04.06.2025. URL: https://help.ubuntu.com/community/UFW.

[25] I. Wazuh. Installing the wazuh agent on arm-based systems, 2024. Accessed: 2025-06-15.

[26] I. Wazuh. Wazuh documentation, 2024. Accessed: 2025-06-15.

[27] Wikipedia. Dnssec, 2024. Dostęp: 2024-06-14.

[28] Wikipedia. Domain name system, 2024. Dostęp: 2024-06-14.

[29] Wikipedia. Domena najwyższego poziomu, 2024. Dostęp: 2024-06-14.

[30] Wikipedia. Docker (software) — wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Docker_(software), 2025. Dostęp zdalny (04.05.2025).

[31] Wikipedia. Suricata — wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Suricata_(software), 2025. Dostęp zdalny (04.05.2025).

[32] Wikipedia contributors. Honeypot (computing), 2024. Accessed: 2025-06-15.

[33] Łukasz Dejko. Plik docker-compose.yml projektu home network guardian. https://github.com/lukaszFD/home-network-guardian/blob/main/docker-compose.yml, 2024. Dostęp 5 czerwca 2025.

[34] Łukasz Dejko. Custom suricata rules for network security monitoring. https://github.com/lukaszFD/home-network-guardian/blob/main/etc-suricata/rules/custom.rules, 2025. Dostęp: 14 czerwca 2025.

[35] Łukasz Dejko. Home Network Guardian - GitHub Repository. Dostęp zdalny (04.05.2025): https://github.com/lukaszFD/home-network-guardian, 2025.

[36] Łukasz Dejko. Konfiguracja filebeat w projekcie home network guardian. https://github.com/lukaszFD/home-network-guardian/tree/main/etc-filebeat, 2025. Dostęp 5 czerwca 2025.

[37] Łukasz Dejko. Konfiguracja pi-hole w projekcie home network guardian. https://github.com/lukaszFD/home-network-guardian/tree/main/etc-pihole, 2025. Dostęp 5 czerwca 2025.

[38] Łukasz Dejko. Konfiguracja prometheus w projekcie home network guardian. https://github.com/lukaszFD/home-network-guardian/tree/main/etc-prometheus, 2025. Dostęp 5 czerwca 2025.

[39] Łukasz Dejko. Konfiguracja squid proxy w projekcie home network guardian. https://github.com/lukaszFD/home-network-guardian/tree/main/etc-squid, 2025. Dostęp 5 czerwca 2025.

[40] Łukasz Dejko. Konfiguracja suricata ids w projekcie home network guardian. https://github.com/lukaszFD/home-network-guardian/tree/main/etc-suricata, 2025. Dostęp 5 czerwca 2025.

[41] Łukasz Dejko. Konfiguracja unbound dla firefoksa w projekcie home network guardian. https://github.com/lukaszFD/home-network-guardian/tree/main/etc-unbound-firefox, 2025. Dostęp 5 czerwca 2025.

[42] Łukasz Dejko. Konfiguracja unbound dns w projekcie home network guardian. https://

github.com/lukaszFD/home-network-guardian/tree/main/etc-unbound, 2025. Dostęp 5 czerwca 2025.

[43] Łukasz Dejko. Raspberry dashboard for grafana — home network guardian project, 2025. Dostęp: czerwiec 2025.