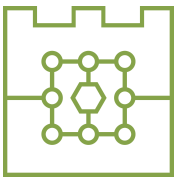
Politechnika Krakowska
im. Tadeusza Kościuszki
Wydział Informatyki i Telekomunikacji



Bartosz Gołas

numer albumu: 129122

Analiza bezpieczeństwa systemów gromadzących i przetwarzających duże zbiory danych na przykładzie aplikacji opartych na Apache Spark, Scala i Apache Hadoop

Security analysis of systems collecting and processing large data sets on the example of applications based on Apache Spark, Scala and Apache Hadoop

Praca magisterska
na kierunku Informatyka
specjalność Cyberbezpieczeństwo

Praca przygotowana pod kierunkiem:
dr Radosław Kycia

Kraków, 2024

Streszczenie

Praca zawiera analizę bezpieczeństwa rozporoszonych systemów opartych na Apache Spark, Scala i Apache Hadoop. Celem było opisanie zagrożeń i używanych zabezpieczeń. Przeanalizowane zostały ataki na dane w czasie ich przechowywania oraz przetwarzania. Zaproponowano sposoby zwiększania bezpieczeństwa.

Abstract

The paper contains a security analysis of distributed systems based on Apache Spark, Scala and Apache Hadoop. The goal was to describe the threats and the security measures used. Attacks on data during storage and processing were analyzed. Methods to enhance security were suggested.

Spis treści

1. Wprowadzenie

W ramach tej pracy przeprowadzono analizę bezpieczeństwa systemu gromadzącego i przetwarzającego duże zbiory danych. Systemy komputerowe stosowane do takich zadań są zwykle projektowane i budowane pod konkretne, z góry ustalone zastosowanie wynikające ze specyfiki działalności podmiotu oraz danych, jakie on wykorzystuje i generuje. Z tego powodu ciężko jednoznacznie określić ich architekturę i wybrać technologie do ich zbudowania. Jednocześnie niewątpliwie narzędzia takie jak Apache Spark i Apache Hadoop są wykorzystywane w wielu takich systemach, co potwierdza ich popularność w ogłoszeniach na portalach specjalizujących się w zatrudnianiu programistów, jak i w wielu studium przypadków prezentowanych w materiałach marketingowych firm z obszaru przetwarzania i gromadzenia danych.

Mimo szerokiego zastosowania tych technologii niezwykle ciężko jest znaleźć kompleksowe badania zabezpieczeń stosowanych w takich systemach. Znane w dziedzinie cyberbezpieczeństwa fundacje jak m.in. OWASP[1] nie opracowały standardów zabezpieczania takich systemów. Wykorzystywane przez profesjonalistów „Web Security Testing Guide"[20], „OWASP Top 10"[19], czy „Mobile Application Security"[18], choć pomocne i mogące służyć za odwołanie w czasie opracowania metod analizy bezpieczeństwa systemów Big Data, nie mogą być stosowane bezpośrednio przy pracy z systemami gromadzącymi i przetwarzającymi duże zbiory danych. Ponadto w dokumentacji technicznej Apache Spark[4] i Apache Hadoop[2] autorzy wprost przyznają, że przy domyślnych ustawieniach zakładają brak możliwości nieuprawnionego dostępu do zasobów systemu. Mimo faktu, że systemy te zwykle znajdują się w prywatnych sieciach wewnętrznych, a dostęp do nich faktycznie jest mocno ograniczony, takie założenie zdaje się przeczyć powszechnie przyjętej praktyce „Multiple-layer security". Ponadto zupełnie ignoruje istnienie zagrożeń wewnętrznych. Takie systemy składają się często z setek lub tysięcy komputerów i ich efektywnie wykorzystanie wymaga udostępnienie tych zasobów dziesiątkom różnych zespołów, które wykonują różne prace i składają się z setek pracowników. Założenie braku intencjonalnych lub przypadkowych nadużyć może być uznane za naiwne i nieżyciowe.

[1] Open Worldwide Application Security Project

Temat tej pracy został wybrany, aby uzupełnić lukę badawczą w tematyce bezpieczeństwa, możliwości zabezpieczenia oraz podatności systemów opartych na Apache Spark i Apache Hadoop.

1.1 Cel badań

W ramach pracy zostanie przeprowadzona analiza bezpieczeństwa systemu gromadzącego i przetwarzającego duże zbiory danych. Oceniony zostanie wpływ domyślnych ustawień na poziom bezpieczeństwa, jaki oferuje klaster wyposażony w wymienione wyżej oprogramowanie. Rozpoznane zostaną możliwe zmiany, których celem będzie zwiększenie bezpieczeństwa systemu. Obiektem badań będzie tylko oprogramowanie związane z domeną systemów Big Data z pominięciem ryzyk związanych z systemem operacyjnym czy innymi elementami sieci komputerowej.

1.2 Zakres pracy

Zakres obejmował:

- przybliżenie dziedziny problemu przez opis systemów Big Data i ich zastosowania,
- omówienie budowy takich systemów i stosowanych technologii,
- przegląd zagrożeń, na jakie podatne są dane w trakcie ich przechowywania, przesyłania oraz wykorzystania,
- przegląd zagrożeń, na jakie są podatne klastry komputerowe służące do gromadzenia i przetwarzania dużych zbiorów danych,
- analiza bezpieczeństwa przykładowego systemu opartego o Apache Spark i Apache Hadoop,
- przedstawienie wyników badań nad bezpieczeństwem analizowanego systemu,
- opracowanie zaleceń mających na celu minimalizację ryzyk.

1.3 Przedmiot badań

Systemy gromadzące i przetwarzające duże zbiory danych budowane są z użyciem wielu różnych oprogramowań i architektur. Dostępne są modele chmurowe oraz zbudowane na podstawie prywatnych klastrów. Mnogość rozwiązań utrudnia zapewnienie bezpieczeństwa takich systemów i wymaga również poczynienia szeregu założeń w ramach

przeprowadzonej analizy. W tej pracy, w oparciu o doświadczenia jej autora, skupiono się na popularnym rozwiązaniu stosowanym przez duże korporacje. Rozwiązaniem tym jest prywatny klaster obliczeniowy oparty o najpopularniejsze rozwiązania open-source, które utrzymywane są w ramach działalności Apache Software Foundation. Analizowany system oparto o technologie takie jak Apache Spark, Apache Hadoop oraz język programowania Scala.

1.4 Problemy badawcze

W celu wykonania właściwej analizy bezpieczeństwa badanego systemu konieczne jest postawienie problemów badawczych, na podstawie których przeprowadzone zostaną badania. W związku z faktem, że zainstalowane oprogramowanie umożliwia zmianę wielu szczegółów konfiguracyjnych, w czasie analizy należy zwrócić szczególną uwagę na wpływ tych ustawień na bezpieczeństwo. Zgodnie z rankingiem OWASP Top 10 z 2021 roku kategoria zagrożeń "Security Misconfiguration" zajmuje wysokie piąte miejsce.[19] Zapewnienie właściwej konfiguracji to zadanie dla administratorów platformy, ale niestety często nie jest ono wykonane prawidłowo i wystarczająco dokładnie. Z powodu wyżej wymienionych czynników analiza bezpieczeństwa systemu będzie przeprowadzona tak, aby odpowiedzieć na poniższe pytania:

Q1 Jaki poziom zabezpieczeń zapewniają zainstalowane komponenty przy domyślnych ustawieniach?

Q2 Jakie kroki należy podjąć, aby zapewnić możliwie wysoki poziom bezpieczeństwa w systemie gromadzącym i przetwarzającym duże zbiory danych?

- w jaki sposób zabezpieczyć przechowywane dane?
- w jaki sposób zabezpieczyć komunikację w czasie przetwarzania danych?

1.5 Hipotezy badawcze

W odpowiedzi na postawione problemy badawcze w tej pracy postawiono następujące hipotezy badawcze:

H1 Domyślna konfiguracja zapewnia niski poziom bezpieczeństwa i podatna jest na wiele ataków.

H2 Aby zapewnić możliwie wysoki poziom bezpieczeństwa należy zapewnić właściwą

konfigurację zainstalowanych komponentów oraz bezpieczne narzędzia uwierzytelniania i autoryzacji.

- Zapewnienie bezpieczeństwa przechowywanych danych wymaga wykorzystania algorytmów szyfrujących oraz właściwie skonfigurowanego dostępu.
- Komunikacja w obrębie klastra powinna zostać zabezpieczona przy pomocy algorytmów kryptograficznych.

1.6 Struktura pracy

W pierwszym rozdziale przedstawiono cel badań oraz zakres pracy. Wymieniono problemy i hipotezy badawcze.

W drugim rozdziale przedstawiono charakterystykę systemów gromadzących i przetwarzających duże zbiory danych. W drugiej kolejności wymieniono branże, które najczęściej korzystają z takich systemów oraz opisano cechy, jakimi charakteryzują się przetwarzane dane. Następnie omówiono najczęściej wykorzystywane technologie i architektury. Dodatkowo opisano zagrożenia, na jakie podatne są dane oraz klastry komputerowe i przedstawiono Apache Spark i Apache Hadoop.

W trzecim rozdziale omówiono zabezpieczenia dostępne w Apache Hadoop oraz Apache Spark. Zidentyfikowano dostępne metody uwierzytelniania i autoryzacji, oraz wymieniono dostępne metody szyfrowania danych w transporcie i w spoczynku.

W czwartym rozdziale przeprowadzono analizę udostępnionych interfejsu. Na testowym klastrze, na którym zainstalowano Apache Hadoop oraz Apache Spark, zbadano, jakie są otwarte porty. W drugiej kolejności zidentyfikowano usługi i protokoły, które są wykorzystywane, oraz opisano uruchomione interfejsy WWW i udostępnione przez nie ścieżki. Następnie przeprowadzono atak na protokół RPC oraz na HDFS.[2]

W piątym rozdziale opisano przeprowadzoną analizę ruchu sieciowego. Opisano, jak wygląda komunikacja między usługami, które działają w obrębie klastra oraz przedstawiono jakie dane mogą zostać przechwycone w czasie podsłuchiwania ruchu sieciowego.

W rozdziale szóstym przedstawiono metody zabezpieczania systemu. Następnie przeprowadzono ponownie ataki z poprzednich rozdziałów i udowodniono skuteczność zastosowanych metod zabezpieczenia dostępu i szyfrowania.

[2] Hadoop Distributed File System

Część teoretyczna

2. Systemy gromadzące i przetwarzające duże zbiory danych

Systemy gromadzące i przetwarzające duże zbiory danych są znane też pod nazwą: **system Big Data**. Pierwszy raz termin Big Data został użyty przez Michael Coxa i Davida Ellswortha w 1997 r. w pracy pod tytułem "Managing big data for scientific visualization".[8][5] Zdefiniowali oni kolekcje Big Data i obiekty Big Data.

Kolekcje opisywali jako agregaty wielu zbiorów danych, które dotyczą wielu dziedzin i przechowywane są w kilku lokalizacjach za pomocą licznych baz danych. Obiekty charakteryzowali jako pojedyncze rekordy lub zbiory, które są zbyt duże, aby mogły być przetwarzane za pomocą powszechnie dostępnego sprzętu komputerowego.[8]

W 2004 r. gdy Jeffrey Dean i Sanjay Ghemawat zaprezentowali prace pod tytułem "Mapreduce: simplified data processing on large clusters"[9] postrzeganie problemu przetwarzania Big Data drastycznie się zmieniło, a znaczenie terminu uproszczono i współcześnie używany jest w odniesieniu do danych, które nie mogą być przechowywane oraz przetwarzane przez pojedynczą jednostkę komputerową.[5]

Systemy Big Data powstały w odpowiedzi na szybko rosnące tempo powstawania nowych danych i w wyniku zapotrzebowania na ich przetwarzanie, przechowywanie oraz analizowanie.

W literaturze takie systemy charakteryzuje pięć cech. Znane są pod nazwą “5 Vs”, która odnosi się do ich angielskich nazw.[15] Są to:

- objętość (ang. volume),
- różnorodność danych (ang. variety),
- prędkość przetwarzania (ang. velocity),
- weryfikowalność (ang. veracity),
- wartość (ang. value).

Objętość wynika z rosnącej prędkości generowania nowych danych. Szacuje się, że do 2025 roku każdego dnia będzie powstawać 463 nowych eksabajtów danych (1 eksabajt to 1000000 terabajtów).[10] Generowane są one przez miliardy internautów oraz urządzeń i komputerów podłączonych do sieci. Dane te często niosą ze sobą wartość naukową,

finansową i analityczną. Tysiące podmiotów na całym świecie gromadzi je i analizuje. Obecnie systemy gromadzące i przetwarzające duże zbiory danych powszechnie stosowane są w:[24][1]

- sektorze ochrony zdrowia,
- edukacji,
- transporcie,
- sektorze finansowym,
- mediach społecznościowych,
- przemyśle.

Różnorodność oznacza, że gromadzone dane mogą mieć różne formy. Mogą być to dane wygenerowane przez człowieka lub maszynę. Mogą one mieć uporządkowaną strukturę, jak np. dane przechowywane w tradycyjnych tabelarycznych bazach danych, lub mogą nie posiadać żadnej struktury jak np. obrazy czy filmy. Istnieją też formaty częściowo ustrukturyzowane jak np. XML.[15]

Prędkość przetwarzania musi rosnąc wraz z rosnącą prędkością generowania nowych danych. Systemy wyspecjalizowane w przetwarzaniu danych pracują często całą dobę. Możliwe jest przetwarzanie danych w czasie rzeczywistym lub praca na podzielonych zbiorach danych tzw. batchach np. z zadanego przedziału czasowego. Bez względu na sposób przetwarzania systemy Big Data charakteryzują się wysoką wydajnością dostarczania wyniku tzw. throughputem.[16]

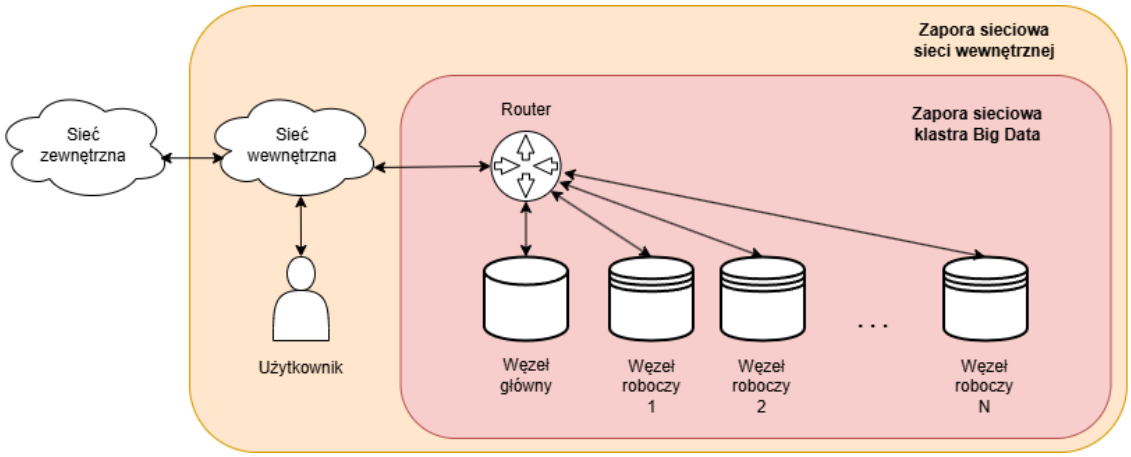
Weryfikowalność to cecha związana z jakością i prawdziwością danych. Gdy przetwarzane są duże zbiory danych, systemy komputerowe muszą mierzyć się z wieloma błędami oraz brakami. Gromadzone informacje pochodzą z wielu źródeł, które charakteryzują się różnymi poziomami dokładności i wiarygodności.[15] Dodatkowo forma prezentacji często jest wybierana indywidualnie przez każde ze źródeł. Dlatego powszechną praktyką jest przeprowadzanie wstępnego oczyszczania i normalizowania gromadzonych danych.[5] W ten sposób zapewnia się efektywny sposób przechowywania i wysoką jakość ostatecznego wyniku przetwarzania.

Wartość danych wynika z wiedzy, jaką można z nich uzyskać za pomocą różnych metod eksploracji danych. Duże zbiory danych i zastosowanie wielu różnorodnych źródeł umożliwia odkrywanie nowych wzorców, korelacji i wniosków, które nie były dostrzegalne w czasie analizy źródłowych zbiorów danych. Pozwala to na dostrzeżenie nowych szans

i umożliwia m.in. wspomaganie decyzji biznesowych, redukcje kosztów, optymalizacje procesów, oraz tworzenie nowych produktów.[15][16] To wszystko przekłada się na realne korzyści finansowe, które odnoszone są przez organizacje wykorzystujące systemy Big Data.

2.1 Budowa systemów gromadzących i przetwarzających duże zbiory danych

Rozmiar informacji przetwarzanych przez system Big Data przekracza możliwości, jakie zapewniają pojedyncze stacje robocze.[5] Często konieczne jest zastosowanie redundancji, co dodatkowo zwiększa zapotrzebowanie tych systemów na zasoby.[16][5] Z tego powodu obecnie najczęściej stosowanym rozwiązaniem są klastry komputerowe zbudowane z wielu jednostek pełniących rolę węzłów.[16] Połączone są ze sobą przy pomocy sieci Ethernet. Na każdej jednostce instaluje się oprogramowanie umożliwiające rozproszone przechowywanie i przetwarzanie danych. Na rysnuku 2.1 zaprezentowano przykład takiej architektury. Węzły takiego klastra dzielimy ze względu na wykonywane przez nie zadania. Możemy wyróżnić dwie główne kategorie: węzły główne (ang. Master nodes) oraz węzły robocze (ang. Worker nodes).[5][16]



Rysunek 2.1 Przykład architektury systemu Big Data

Opracowanie własne

Węzły główne odpowiedzialne są za zarządzanie zasobami systemu Big Data. W obrębie tych węzłów zainstalowane jest oprogramowanie rozdzielające prace i nadzo-

rujące działanie rozproszonego systemu plików lub innych rozwiązań przechowywania danych w sposób rozproszony.[5][16]

Węzły robocze to jednostki, które odpowiedzialne są za wykonywanie zadań obliczeniowych oraz przechowywanie danych.[5][16]

Dane w systemach Big Data przetwarzane są na dwa różne sposoby:

- Tryb wsadowy (ang. Batch processing). Polega on na przetwarzaniu dużych zbiorów danych, które dzielone są na mniejsze fragmenty.[16] Części można przygotować w dowolny sposób, ale często są podzielone chronologiczne. Przetwarzanie takich porcji danych może odbywać się regularnie lub doraźnie. W wielu organizacjach tryb wsadowy wykorzystywany jest do regularnego wykonywania analiz jak np. generowanie dziennych raportów lub częściowego przygotowywania bieżących danych do dalszych obliczeń. Spośród wkorzystywanych technologii warto wymienić MapReduce oraz Apache Spark.[16]
- Przetwarzanie strumieniowe (ang. Stream processing). Przetwarzanie strumieniowe często jest łączone z przetwarzaniem czasu rzeczywistego.[16] Wykorzystywane jest w sytuacjach gdy dane nie mają jasno określonego końca i dostarczane są w sposób ciągły. Jest to forma przetwarzania równoległego, która polega na tworzeniu strumieni składających się z operacji, którym musi być poddany każdy element strumienia. Takie przetwarzanie upraszcza zrównoleglenie całego procesu, maksymalnie wykorzystuje moc infrastruktury oraz umożliwia zaimplementowanie kolejek. Spośród wykorzystywanych technologii warte odnotowania są Apache Kafka oraz Apache Flink.

Obecnie powstaje coraz więcej rozwiązań zapewniających system Big Data w postaci SaaS[1], PaaS[2] lub IaaS[3][27][13]. Rozwiązania te oparte są o platformy chmurowe. Upraszczają one proces przechowywania i przetwarzania danych w jeszcze większym stopniu, skupiając się na ostatecznej analizie danych, często samemu zajmując się problematyką techniczną, taką jak szczegóły przechowywania i przetwarzania. Rozwiązania te adresują problemy znane z klastrów opartych na ekosystemie Hadoop, jak np. kwestie optymalizacyjne czy integrację danych między elementami ekosystemu. Jako przykład takich rozwiązań można podać Snowflake czy Databricks.

[1] Software as a Service - Oprogramowanie jako usługa

[2] Platform as a Service - Platforma jako usługa

[3] Infrastructure as a Service - Infrastruktura jako usługa

2.2 Zagrożenia dla danych

2.2.1 Dane w transporcie

Dane w transporcie to dane przesyłane przez sieć komputerową. W systemach Big Data pojedyncza informacja może być przesyłana wiele razy w ciągu swojego cyklu życia. Wynika to z faktu, że może być wielokrotnie używana do różnych obliczeń i analiz. Ze względu na wykorzystywaną replikacje każda informacja musi być przesłana na wiele węzłów.[5] Z powodu działania frameworków obliczeniowych ta sama część obliczeń może być wykonywana wielokrotnie na wielu różnych węzłach obliczeniowych.[6] Pozostawanie danych w ruchu jest zagrożeniem, ponieważ każda forma transportu może być przechwycona przez innych uczestników sieci.

2.2.2 Dane w spoczynku

Dane w spoczynku to dane, które zostały zapisane w trwałej pamięci komputerowej. Takie dane w wyniku ataku mogą zostać odczytane, zmodyfikowane lub usunięte przez nieuprawnionego użytkownika. W tej formie często przechowywane są dane wrażliwe, prawnie chronione lub będące częścią systemu informatycznego.[5] Atakujący może przechwycić klucze kryptograficzne lub pliki konfiguracyjne, co może umożliwić kolejne ataki. Nieodpowiednie zabezpieczenie danych może prowadzić do problemów prawnych, strat wizerunkowych i finansowych. W określonych w aktach prawnych sytuacjach konieczne jest ich szyfrowanie.[5]

2.2.3 Dane w użyciu

Nawet w sytuacji gdy dane są właściwie chronione w czasie ich przechowywania i przesyłania, to w przypadku wielu operacji i analiz konieczne jest ich odszyfrowanie i przetwarzanie w postaci jawnej. Mimo wielu trwających obecnie prac nad metodami przetwarzania danych zaszyfrowanych nie jest jeszcze możliwe ich wykorzystanie we wszystkich współczesnych procesach z uwagi na wysoką złożoność obliczeniową.[26] Kluczowe staje się zabezpieczenie stacji roboczych i systemów operacyjnych w taki sposób, aby zminimalizować szanse ataków na pamięć operacyjną lub pamięć cache i rejestry procesorów.[5]

2.3 Zagrożenia dla klastrów

Klastry obliczeniowe, oprócz zagrożeń, na które narażone są pojedyncze jednostki komputerowe, zmagają się z zagrożeniami wynikającymi z ich budowy opartej na sieciach komputerowych oraz wielu komputerach.

Podstawowym wyzwaniem jest zapewnienie bezpieczeństwa pojedynczych jednostek komputerowych i utrzymanie kompatybilnych ustawień oraz wersji oprogramowania na wszystkich węzłach. Wysoka liczba maszyn wpływa na ilość czasu potrzebnego do osiągnięcie tego celu.[22]

Aby zapewnić bezpieczeństwo systemu komputerowego konieczne jest jego monitorowanie i zapis działań uruchamianych aplikacji, czynności wykonywanych przez użytkowników i procesy, a także statystyk związanych z ruchem sieciowym. Przy dużej skali systemu należy zapewnić rozwiązanie, które będzie gromadzić te informacje, aby zapewnić możliwość ich analizowania nawet w przypadku utraty pojedynczych węzłów.[22]

W Systemach BigData zapewnienie dostępu użytkownikowi wymaga utworzenia jego konta na wielu węzłach. Alternatywnym rozwiązaniem jest zainstalowanie oprogramowania, które zajmie się uwierzytelnianiem. Proces uwierzytelniania jest fundamentalny dla bezpieczeństwa systemu komputerowego.[7] Z systemem uwierzytelniania często powiązany jest system autoryzacji oraz monitorowania systemu.

Kolejnym wyzwaniem w systemach BigData jest autoryzacja. Występuje wiele wyzwań związanych z propagowaniem właściwych zasad dostępu w obrębie systemu. Użytkownicy i administratorzy muszą się liczyć z problemami związanymi ze zmieniającymi się danymi, rozbudową klastra oraz używanego oprogramowania. Błędy we właściwym skonstruowaniu i aktualizowaniu zasad autoryzacji prowadzą do wewnętrznych nadużyć np. dostępu do danych, które powinny być niedostępne dla danej grupy użytkowników.[17]

Kryptografia jest podstawowym środkiem zabezpieczeń. Klucze kryptograficzne używane są w procesach uwierzytelniania i autoryzacji.

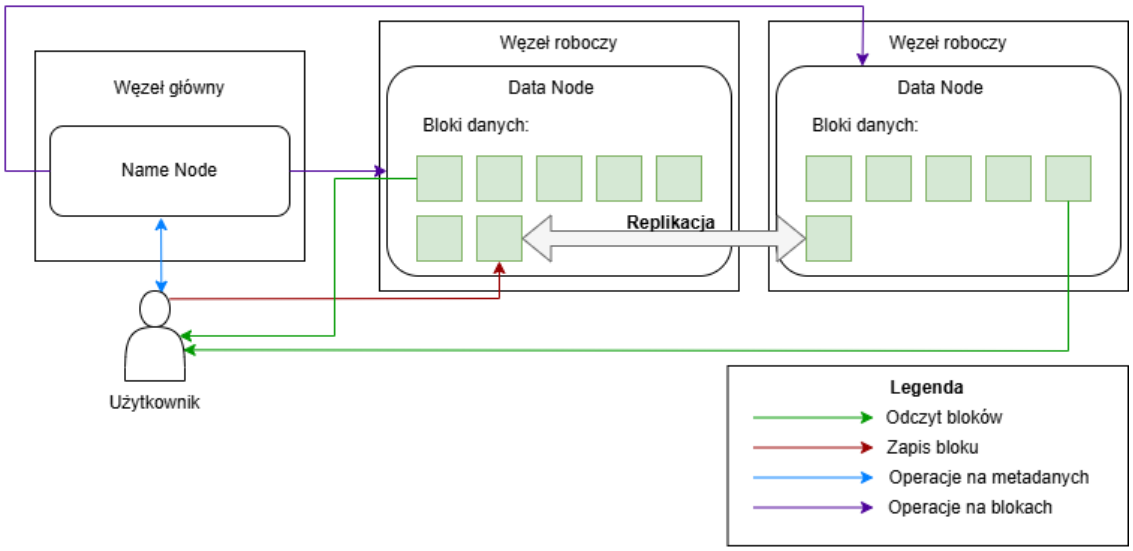
W przypadku klastrów komputerowych zachodzi potrzeba przesyłania kluczy publicznych oraz prywatnych pomiędzy węzłami. Przechwycenie klucza użytkownika może dać atakującemu możliwość podszycia się pod niego oraz używania zasobów, do których ma on dostęp. Właściwe zarządzanie kluczami jest jednym z najpoważniejszych zagrożeń,

z jakimi mierzą się użytkownicy systemów komputerowych. Właściwe zabezpieczenie kluczy ma większe znaczenie niż siła wybranego algorytmu kryptograficznego.[7]

2.4 Architektura Apache Hadoop

Apache Hadoop to platforma programistyczna, której celem jest umożliwienie przechowywania i przetwarzania dużych zbiorów danych w oparciu o klastry komputerowe. Jej kod źródłowy powstał głównie w Javie. Składa się z 4 głównych modułów: Hadoop Commons, Hadoop Distributed File System, Hadoop Yarn i Hadoop MapReduce.[29] Wokół tej platformy powstało wiele projektów, które z niej korzystają i rozszerzają jej możliwości. Pierwsze stabilne wydanie powstało w 2006 roku. Projekt pozostaje aktywnie rozwijany i jest szeroko używany przez firmy i instytucje pracujące z dużymi zbiorami danych. Na podstronie oficjalnej strony projektu poświęconej jej użytkownikom znajdziemy takie znane firmy jak m.in. Adobe, Ebay, Facebook, Google, IBM, LinkedIn, Spotify i Yahoo![23]

Hadoop Distributed File System



Rysunek 2.2 Architektura Hadoop Distributed File System

Opracowanie własne na podstawie:

https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs

/images/hdfsarchitecture.png, dostęp: 29.09.2024

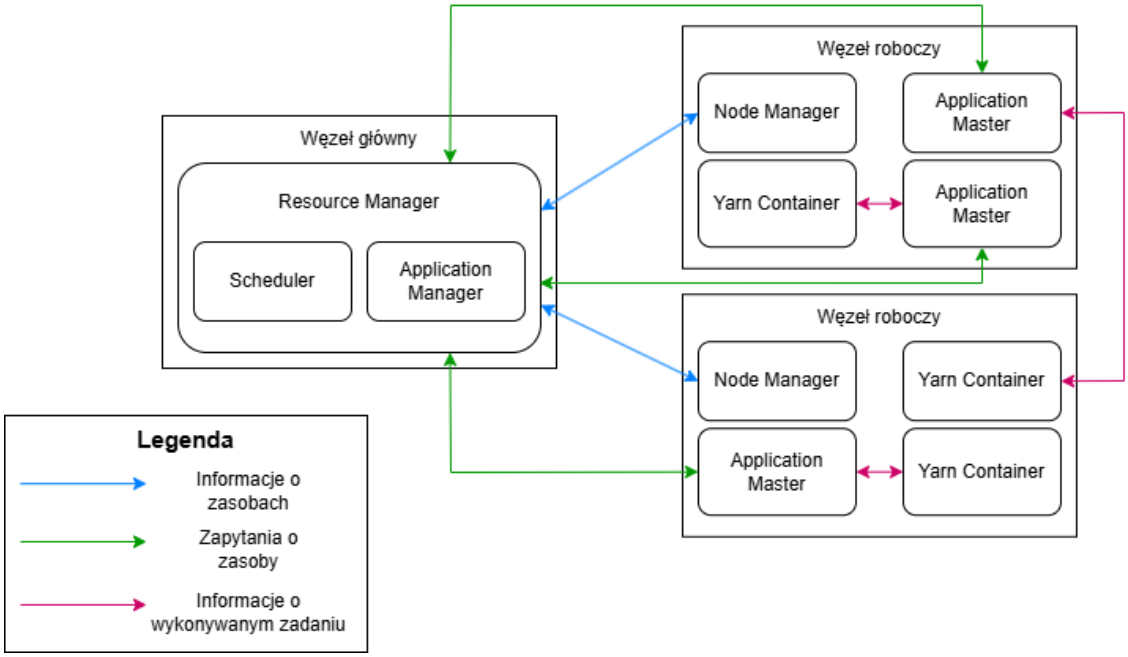
Hadoop Distributed File System to rozproszony system plików. Zastosowano w nim

schemat master/slave. Sercem HDFS jest NameNode, który zarządza całym systemem. Pozostałe węzły nazywane są Data Node'ami i to one przechowują pliki. HDFS zaprojektowano w taki sposób, aby był odporny na różnego rodzaju błędy związane z hardwarem, softwarem i siecią. System plików może być uruchomiony na dowolnej platformie, która obsługuje JVM[4]. Zapewnia on replikacje plików.[29] Każdy zapisany plik podzielony jest na bloki, które następnie zapisane są na węzłach klastra. To NameNode decyduje o miejscu, gdzie powinny zostać zapisane poszczególne bloki i jego zadaniem jest śledzenie ich lokalizacji, a także decydowanie o utworzeniu większej ilości kopii lub usunięciu istniejących. Klienci klastra łączą się z wieloma węzłami jednocześnie w czasie czytania danych. Ten mechanizm pozwala na zwiększenie prędkości odczytu. W czasie zapisu klient łączy się z pierwszym wskazanym przez NameNode węzłem i to na nim zapisuje dane. Ten węzeł otrzymuje od NameNode listę kopii, jaka powinna być utworzona w obrębie klastra, więc w czasie zapisywania danych od klienta przesyła je dalej do kolejnego węzła na tej liście. Proces jest powtarzany, do momentu utworzenia zakładanej ilości kopii. Schemat działania zapisywania, czytania oraz replikacji danych w HDFS przedstawiono na rysunku 2.2. Ten system plików został zaprojektowany z myślą o dużych plikach, które są zapisywane raz, ale odczytywane wielokrotnie. Jednym z założeń systemu jest wysoka wydajność całkowita odczytu, nawet z poświęceniem czasu dostępu.[29]

Hadoop Yarn

Hadoop Yarn to część środowiska Hadoop odpowiedzialna za przydział zasobów, planowanie działań i monitorowanie postępu działających aplikacji.[29] W obrębie Yarna istnieje globalny ResourceManager, który uruchomiony jest na głównym węźle oraz Node Managery uruchamiane na każdym węźle roboczym. Node Managery monitorują zużycie zasobów i przesyłają te informacje do Resource Managera. Składa się on z Schedulera i Application Managera. Scheduler odpowiada za przydział wszystkich zasobów w obrębie klastra. Zasoby przydzielane są na podstawie zdefiniowanych kolejek. Application Manager przyjmuje zgłoszenia programów do wykonania i odpowiedzialny jest za uruchomienie pierwszego kontenera dla każdego z nich. Ten pierwszy kontener to Application Master, który odpowiada za zgłoszenie zapotrzebowania na kolejne kontenery potrzebne do wyko-

[4] Java Virtual Machine



Rysunek 2.3 Architektura Hadoop Yarn

Opracowanie własne na podstawie:

https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site

/yarn_architecture.gif, dostęp: 29.09.2024

nania zadania i ich nadzorowanie.[29] Schemat zależności aplikacji działających w ramach działania Yarna przedstawiono na rysunku 2.3.

Hadoop MapReduce

Hadoop MapReduce to część środowiska Hadoop umożliwiająca tworzenie oprogramowania do przetwarzania równoległego danych zgromadzonych w obrębie klastra. Zaprojektowany jest tak, aby wspierać wiele języków programowania. Podstawowym założeniem tego oprogramowania jest podział wykonywanych zadań na zadania typu Map oraz zadania typu Reduce.[29] Po wczytaniu dane zostają podzielone na części i przekazane do przetwarzania równoległego przez zadania typu Map. Ich celem jest przetworzenie danych wejściowych na częściowe wyniki. Kolejnym etapem jest segregacja danych. Następnie są one dostarczane do zadań typu Reduce, które dokonują ich agregacji. Wszystkie operacje są wykonywane na indywidualnych parach kluczy i wartości. MapReduce korzysta z Yarna, więc zapewnia skalowalność i odporność na błędy.[29]

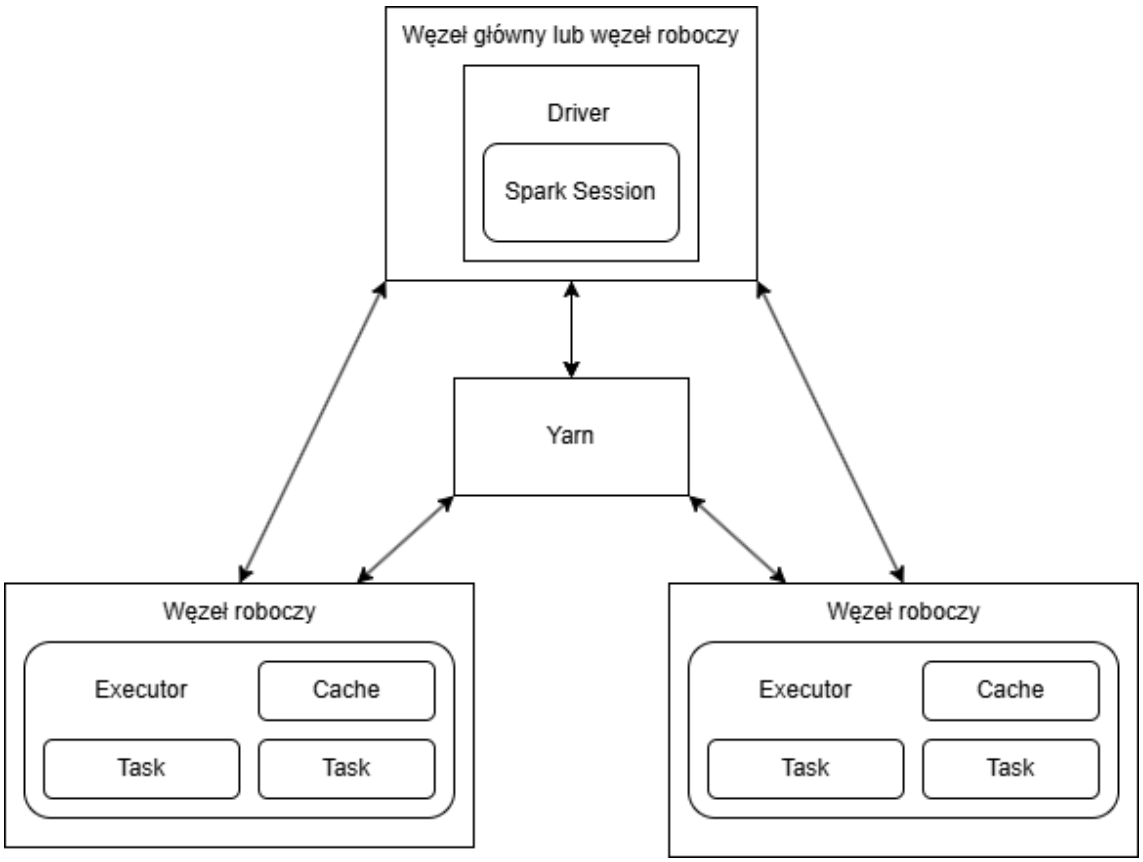
2.5 Architektura Apache Spark

Apache Spark to narzędzie do przetwarzania dużych zbiorów danych. Stworzony został w 2009 r. na Uniwersytecie Kalifornijskim w Berkeley. Można z niego korzystać, używając takich języków programowania jak Scala, Java, Python, R i SQL. Sam kod źródłowy napisany jest głównie w Scali. Do głównych zastosowań Sparka należy Data Engineering, Data Science i Machine Learning. To jedno z najpopularniejszych narzędzi służących do przetwarzania danych. W jego rozwoju wzięło udział już ponad 2000 programistów i nadal pozostaje aktywnie rozwijany. Wykorzystywany jest przez 80% firm z listy Fortune 500.[3]

Każda aplikacja korzystająca ze Spark składa się z dwóch rodzajów procesów. Pierwszym z nich jest proces nazywany Driver. Drugim typem procesu jest tzw. Executor.

Driver składa się z dwóch komponentów, Spark Context oraz kodu napisanego przez użytkownika. To on odpowiedzialny jest za działanie wszystkich procesów typu Executor. Zarządzanie zadaniami może odbywać się przez scheduler wbudowany w Spark lub za pomocą kontaktu z zewnętrznymi rozwiązaniami takimi jak YARN, Mesos lub Kerberos. Każda uruchomiona aplikacja Spark zawiera dokładnie jeden proces typu Driver. Uruchomiony on może być lokalnie lub w obrębie klastra. W tym samym czasie w systemie BigData może być uruchomionych wiele takich procesów obsługujących wiele aplikacji. Driver działa od początku do końca działania aplikacji i może być z nią utożsamiany. Executory wykonują zadania zlecone przez proces Driver. W obrębie jednej aplikacji może istnieć ich wiele. Są to procesy wykonujące kod Sparka w celu osiągnięcia zleconego zadania. Budowę aplikacji korzystającej ze Sparka przedstawiono na rysunku 2.4.

Typowa aplikacja korzystająca ze Sparka zawiera utworzenie Spark Session i instrukcje wykonywane na tym obiekcie. Na ich podstawie Spark dzieli zadanie na tzw. Taski, a dane na mniejsze części, czyli partycje. Każdy task to wykonanie jednej instrukcji na jednej partycji. Taski wykonywane są przez pojedyńcze Executory. Są one grupowane w zbiory zwane Stage'ami. Każdy Stage wymaga przemieszczenia się danych w obrębie klastra, ponieważ niektórych operacji nie można wykonać na pojedynczej partycji. Stage składają się na tzw. Joby, czyli ciągi transformacji wykonywanych na zbiorze danych. Każda aplikacja może składać się z wielu jobów.[6]



Rysunek 2.4 Architektura Apache Spark w systemie Big Data opartym o Yarn

Opracowanie własne na podstawie:

https://spark.apache.org/docs/latest/img/cluster-overview.png, dostęp: 29.09.2024

3. Dostępne zabezpieczenia

3.1 Zabezpieczenia w Apache Hadoop

Na podstawie oficjalnej dokumentacji[2][12][14][28][30], dostępnej konfiguracji oraz kodu źródłowego, zostaną przedstawione zabezpieczenia, jakie dostępne są w obrębie platformy Apache Hadoop.

3.1.1 Uwierzytelnianie

Przy domyślnych ustawieniach Apache Hadoop przeprowadza prosty proces uwierzytelniania. Przesyłane są tylko dane wskazujące na nazwę użytkownika, bez żadnego dowodu tożsamości jak hasło czy token. Wszystkie procesy są uruchamiane jako tacy sami użytkownicy, jacy zostali użyci do uruchomienia demonów platformy.

Interfejsy WWW

Apache Hadoop udostępnia wiele różnych typów interfejsów, które oferują odmienne sposoby uwierzytelniania. Jednym z nich są interfejsy WWW udostępniane przez platformę Hadoop. Domyślnie uwierzytelnianie jest dla nich wyłączone, ale istnieje możliwość jego uruchomienia. Dostępnymi metodami są Kerberos za pomocą protokołu HTTP SPNEGO lub tzw. prosta autoryzacja. W przypadku prostej autoryzacji konieczne jest dodanie parametru zawierającego nazwę użytkownika np. http://localhost:8088/cluster?-user.name=nazwa_użytkownika. Możliwe jest też stworzenie własnej metody uwierzytelniania przez rozszerzenie platformy o własny kod. Hadoop pozwala również na konfigurację CORS na wszystkich uruchomionych interfejsach HTTP.[11]

Protokół RPC i inne metody dostępu

Poszczególne węzły klastra komunikują się ze sobą m.in. przy pomocy protokołu RPC. W tym przypadku jedyną dostępną metodą bezpiecznego uwierzytelniania jest wykorzystanie protokołu Kerberos. Podobnie przebiega uwierzytelnianie w przypadku innych metod dostępu, jak np. uruchamianie komend na poziomie powłoki systemu ope-

Tabela 3.1 Zalecany podział na użytkowników i grupy

Użytownik	Grupa	Usługa
hdfs	hadoop	NameNode, Secondary NameNode, JournalNode, DataNode
yarn	hadoop	ResourceManager, NodeManager
mapred	hadoop	MapReduce JobHistory Server

Opracowanie własne na podstawie: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SecureMode.html, dostęp: 03.11.2024 r.

racyjnego przez użytkowników klastra. W dokumentacji[12] autorzy zalecają utworzenie osobnych użytkowników dla poszczególnych usług i zastosowanie dla nich wszystkich jednej grupy.[12] Zalecaną konfiguracje przedstawiono w tabeli 3.1.

Po włączenia uwierzytelniania, Hadoop domyślnie mapuje Kerberos Principal na użytkownika zdefiniowanego na poziomie systemu operacyjnego. Sposoby tego mapowania mogą zostać określone w konfiguracji Hadoopa.[12] Aby utrzymać prawidłowe działania całego systemu, ważne jest utrzymanie synchronizacji między ustawieniami Hadoopa, Kerberosa i listą użytkowników zdefiniowanych na poziomie systemu operacyjnego poszczególnych węzłów klastra.

3.1.2 Autoryzacja

Usługi

Autoryzacja na poziomie usług w Apache Hadoop odbywa się za pomocą Access Control List oraz Blocked Access Control List. Za pomocą list można skonfigurować prawo poszczególnych użytkowników i grup do korzystania z protokołów wykorzystywanych przez Apache Hadoop. Dodatkowo możliwe jest stworzenie Access Control List oraz Blocked Access Control List na podstawie adresów IP, nazw hostów lub zakresów adresów IP. Domyślnie wszystkie ACL[1] zawierają „*” dając wszystkim użytkownikom prawa korzystania ze wszystkich protokołów. Listę protokołów wspierających zastosowanie ACL przedstawiono w tabeli 3.2.

[1] Access Control List

Tabela 3.2 Protokoły dostępne w Apache Hadoop

Protokół	Opis
ClientProtocol	Protokół używany przez kod użytkownika do komunikacji z rozproszonym systemem plików.
ClientDatanodeProtocol	Protokół używany do komunikacji przez klientów z Data Nodem w trakcie procesu "Block recovery".
DatanodeProtocol	Protokół używany przez Data Node do komunikacji z Name Nodem.
InterDatanodeProtocol	Protokół używany przez Data Node do komunikacji z innymi Data Node'ami.
DatanodeLifelineProtocol	Protokół używany przez Data Node do wysyłania wiadomości typu „lifeline" do Name Node'a.
NamenodeProtocol	Protokół używany przez Secondary Name Node do komunikacji z głównym Name Nodem.
AdminOperationsProtocol	Protokół używany do przesyłania komend administracyjnych.
GetUserMappingProtocol	Protokół używany przez Name Node i JobTracker do mapowania użytkowników na ich grupy.
RefreshUserMappingsProtocol	Protokół używany do odświeżenia instrukcji mapowania użytkowników na grupy.
ReconfigurationProtocol	Protokół używany do odświeżenia konfiguracji Name Node'a lub Date Node'a.
RefreshAutorization PolicyProtocol	Protokół używany do odświeżenia obecnie obowiązujących zasad autoryzacji.
RefreshCallQueueProtcol	Protokół używany do odświeżenia kolejek.
GenericRefreshProtocol	Generyczny protokół do odświeżania konfiguracji.
TaskUmbilicalProtocol	Protokół używany przez MapReduce. Za jego pomocą odbywa się komunikacja między zadaniami typu map oraz typu reduce z Node Managerem.
HAServiceProtocol	Protokół używany przez administratorów do zarządzania stanami Name Node'a.
ZKFailoverProtocol	Protokół do zarządzania systemem „ZK Failover".
QJournalProtocol	Protokół używany przez Namenode do komunikacji z Journal Node'ami gdy używany jest Quorum Journal Manager.
InterQJournalProtocol	Protokół używany przez Journal Node'y do komunikacji z innymi JournalNode'ami.
HSClientProtocol	Protokół używany przez klientów do komunikacji z MapReduce History Server.

Protokół	Opis
ResourceTrackerProtocol	Protokół używany przez Resource Managera i Node Managera do wzajemnej komunikacji.
ResourceManager AdministrationProtocol	Protokół używany do przesyłania komend administracyjnych związanych z Resource Managerem.
ApplicationClientProtocol	Protokół używany przez Resource Managera i aplikacje zlecające zadania do wzajemnej komunikacji.
ApplicationMasterProtocol	Protokół używany przez Resource Managera i Application Mastera do wzajemnej komunikacji.
ContainerManagementProtocol	Protokół używany przez Node Managera i Application Mastera do wzajemnej komunikacji. Służy do uruchamiania i przerywania pracy kontenerów Yarn. Używany jest też do zarządzania przypisanymi zasobami.
ResourceLocalizerProtocol	Protokół używany przez Node Managera i Resource Localizera do wzajemnej komunikacji.
MRClientProtocol	Protokół używany przez MapReduce do komunikacji z Application Masterem.
ApplicationHistoryProtocol	Protokół używany przez Yarn Timeline Server i generyczne aplikacje klienckie do wzajemnej komunikacji.
CollectorNodemanagerProtocol	Protokół używany przez Node Managera, jeśli uruchomiona jest druga wersja Timeline Service. Służy do komunikacji wzajemnej z Timeline Collectorem.
ApplicationMasterProtocol	Protokół używany przez Application Mastera i Node Managera do wzajemnej komunikacji.
DistributedSchedulingAM Protocol	Protokół używany przez Node Managera i Resource Managera do wzajemnej komunikacji.
LocalizationProtocol	Protokół używany przez Resource Localization Service.

Opracowanie własne na podstawie pliku konfiguracyjnego hadoop-policy.xml będącego częścią instalacji Apache Hadoop

Hadoop Distributed File System

HDFS posiada zaimplementowany system uprawnień podobny do tego stosowanego w systemach z rodziny UNIX. Pliki i katalogi mają przypisanego swojego właściciela i jedną grupę. Właściciel może określać prawa do odczytu, pisania i wykonania osobno dla siebie, zdefiniowanej grupy i całej reszty użytkowników. W ramach HDFS nie istnieją pliki wykonywalne, dlatego dla plików uprawnienie wykonywania nie zmienia nic,

Tabela 3.3 Dostępne poziomy zabezpieczenia protokołu RPC

Poziom zabezpieczeń	Opis
authentication	Przeprowadzany jest proces uwierzytelniania.
integrity	Przeprowadzany jest proces uwierzytelniania, a dodatkowo sprawdzana jest integralność przesyłanych danych.
privacy	Przeprowadzany jest proces uwierzytelniania, sprawdzana jest integralność, a dane są zaszyfrowane.

Opracowanie własne na podstawie: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/hdfs-default.xml, dostęp: 07.12.2024 r.

a dla katalogów pozwala na otworzenie podkatalogów. Z tego samego powodu system uprawnień w HDFS nie wspiera setui ani setgid. Możliwe jest wykorzystanie bitu lepkości (ang. sticky bit) dla katalogów, aby uniemożliwić usuwanie lub przemieszczanie plików i podkatalogów innym użytkownikom niż ich właściciele. W momencie utworzenia nowego pliku lub katalogu jego właścicielem zostanie użytkownik, który go utworzył, a grupa zostanie ustawiona na grupę katalogu, w którym został utworzony.[14]

Poza podstawowym systemem uprawnień HDFS wspiera również znane z POSIX Access Control List. W ramach tych list możliwe jest dodatkowe ustawianie uprawnień dla użytkowników i grup. System ten wspiera maskowanie uprawnień oraz w przypadku katalogów umożliwia definiowanie domyślnych uprawnień, jakie odziedziczą nowo utworzone pliki i podkatalogi.[14]

3.1.3 Szyfrowanie danych w transporcie

Przy domyślnych ustawieniach dane w transporcie nie są szyfrowane. W Hadoopie możliwe jest zastosowanie szyfrowania na poziomie czterech kanałów:

- Protokołu RPC używanego do komunikacji między usługami i klientami.
- Protokołu TCP/IP używanego do przesyłania danych o blokach między Data Node'ami i klientami.
- Protokołu HTTP w przypadku interfejsów WWW.
- Protokołu HTTP w czasie wykonywania operacji shuffle przez MapReduce.

W przypadku zastosowania protokołu RPC szyfrowanie możliwe jest tylko, gdy wcześniej uruchomione zostanie uwierzytelnianie za pomocą Kerberosa. Możliwe trzy poziomy ochrony przedstawiono w tabeli 3.3.

Tabela 3.4 Dostępne poziomy zabezpieczenia interfejsów WWW

Poziom zabezpieczeń	Opis
HTTP_ONLY	Domyślny poziom zabezpieczeń to wykorzystanie protokołu HTTP bez szyfrowania.
HTTPS_ONLY	W przypadku zastosowania tego poziomu zabezpieczeń komunikacja możliwa jest tylko za pomocą protokołu HTTPS.
HTTP_AND_HTTPS	Ten poziom pozwala zarówno na komunikację przy pomocy HTTP i HTTPS.

Opracowanie własne na podstawie: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/hdfs-default.xml, dostęp: 07.12.2024 r.

W przypadku użycia poziomu privacy domyślnie używany algorytm zależy od implementacji JVM, ale zwykle jest to 3DES. Dla RPC możliwe jest użycie własnej implementacji dowolnego algorytmu. W przypadku protokołu TCP/IP, używanego do transmisji danych o blokach, możliwe jest użycie 3DES lub RC4. Dodatkowo po wymianie kluczy przy pomocy wybranego algorytmu dalsza transmisja może odbyć się przy pomocy algorytmu AES.

W przypadku interfejsu WWW korzystających z HTTP lub HTTPS możliwe jest skonfigurowanie jednego z trzech poziomów zabezpieczeń. Wymieniono je w tabeli 3.4. Możliwe jest skonfigurowanie innych wartości dla HDFSa, Yarna i MapReduce'a.

3.1.4 Szyfrowanie danych w spoczynku

W ramach platformy Hadoop zaimplementowano tzw. „Przezroczyste szyfrowanie" (ang. Transparent Encryption). To opcjonalne rozwiązanie zapewnia szyfrowanie end-to-end, które nie wymaga zmian w istniejących aplikacjach. Do działania tej funkcji wymagana jest dodatkowa usługa: Hadoop Key Management Server. W ramach istniejącego systemu plików administratorzy klastra tworzą nową ścieżkę nazywaną encryption zone. Każda taka ścieżka posiada własny klucz kryptograficzny. W jej obrębie wszystkie pliki i podkatalogi są szyfrowane w tle przez klientów HDFS. Do działania tego rozwiązania są potrzebne trzy rodzaje kluczy:[28]

- Encryption zone key — klucz związany z pojedynczą encryption zone.
- Data encryption key (DEK) — klucz używany do szyfrowania pojedynczego pliku.
- Encrypted data encryption key (EDEK) — to DEK pojedynczego pliku zaszyfrowany

przy pomocy encryption zone key, związanego z encryption zone, w której plik został zapisany.

Gdy nowy plik lub katalog jest tworzony, zostaje on zaszyfrowany przy pomocy DEK, który został dostarczony przez KMS. Zaszyfrowane dane są przesyłane przez klienta na węzeł HDFS. Namenode w trakcie tworzenia pliku wysyła żądanie o pobranie EDEK do KMS. Otrzymany EDEK zostaje zapisany wraz z metadanymi o tworzonym pliku. Gdy plik jest czytany przez użytkownika, otrzymuje on zaszyfrowane dane, EDEK oraz numer wersji zastosowanego encryption zone key. Klient wysyła żądanie odszyfrowania EDEK do KMS, który weryfikuje uprawnienia użytkownika i odsyła DEK. Dzięki temu rozwiązaniu tylko klient może uzyskać dostęp do zawartości zaszyfrowanego pliku. HDFS i KMS nie posiadają informacji niezbędnych do odczytania danych.[28]

3.1.5 Zabezpieczenie kontenerów Yarn

Yarn pozwala na wybranie implementacji Container Executora. Jest to klasa odpowiedzialna za uruchamianie i zarządzanie procesami zleconymi do wykonania przez aplikacje klienckie. Domyślna implementacja wykorzystuje tego samego użytkownika, który jest właścicielem procesu Node Managera. Istnieją dwie inne, bezpieczniejsze implementacje: Linux Secure Container Executor oraz Windows Secure Container Executor. Te implementacje pozwalają na uruchomienie procesu jako ten sam użytkownik, który zlecił jego wykonanie lub jako inny predefiniowany użytkownik. Umożliwia to zastosowanie ograniczeń w oparciu o systemy uprawnień HDFS oraz lokalnego systemu plików. Dodatkowo obie implementacje umożliwiają stworzenie Access Control List oraz Blocked Access Control List użytkowników.[30]

3.2 Zabezpieczenia w Apache Spark

Na podstawie oficjalnej dokumentacji[25][21], dostępnej konfiguracji oraz kodu źródłowego, zostaną przedstawione zabezpieczenia, jakie dostępne są w Apache Spark. Platforma ta umożliwia samodzielne działanie w obrebie klastra komputerowego lub w oparciu o Yarna, Kerberosa lub Mesos. W zależności od tego wyboru dostępne są inne zabepiezpiecznia. W tej pracy nacisk zostanie położony na możliwości oferowane w klastrach opartych o Yarna.

3.2.1 Uwierzytelnianie

Domyślnie uwierzytelnianie w Sparku jest wyłączone. Po jego uruchomieniu protokół RPC wykorzystuje przesyłany sekret. Gdy używany jest Yarn, unikalny sekret jest generowany dla każdej aplikacji korzystającej ze Sparka. System ten działa w oparciu o protokół Kerberos. W przypadku protokołu HTTP możliwe jest uruchomienie uwierzytelniania przy pomocy filtrów serwletów javax. Spark nie zawiera domyślnej implementacji takiego filtra, więc użytkownik musi stworzyć własną implementację.[25]

3.2.2 Autoryzacja

Przy domyślnych ustawieniach Apache Spark nie przeprowadza autoryzacji, jednak umożliwia jej uruchomienie i skonfigurowanie.

W przypadku interfejsów WWW autoryzacja oparta jest o Access Control List. Istnieją trzy poziomy dostępu: view, modify i admin. View pozwala na wyświetlanie interfejsu. Modify na modyfikowanie i zatrzymywanie aplikacji. Admin to połączenie uprawnień view i modify. Domyślnie użytkownik, który utworzył aplikacje, jest dodawany do tych list. Możliwe jest dodanie zarówno użytkowników jak i grup.

3.2.3 Szyfrowanie danych w transporcie

Szyfrowanie nie jest domyślnie uruchomione, jednak Spark umożliwia zabezpieczenie protokołu RPC, którego używa do komunikacji w czasie wykonywania zleconych działań. Platforma pozwala na oparcie tego procesu na jednej z dwóch technologii: AES lub SASL. Zalecane jest stosowanie szyfrowania opartego o algorytm AES, ponieważ szyfrowanie oparte o SASL oznaczone jest jako przestarzałe i obsługiwane jest przez nowe wersje, aby zapewnić kompatybilność wsteczną. W przypadku wybrania algorytmu szyfrowania AES możliwe jest użycie jednej z dwóch jego wersji: AES/CTR/NoPadding lub AES/GCM/NoPadding. Pierwsza z nich nie wykonuje uwierzytelniania i wspiera starsze wersje Spark. Druga obsługuje uwierzytelnianie. Dla innych protokołów możliwe jest zastosowanie SSL, który może być oparty o dowolny protokół TLS wspierany przez stosowaną JVM.[25]

3.2.4 Szyfrowanie danych w spoczynku

Spark umożliwia użytkownikowi zaszyfrowanie danych przed ich zapisem. Dodatkowo możliwe jest szyfrowanie w czasie zapisu. Ta funkcja wspierana jest dla plików w formacie parquet, które składają się z kolumn. Zaszyfrować można jedną lub wiele z nich.[21] W czasie przetwarzania część danych może zostać tymczasowo zapisana na dysku twardym. Może to nastąpić gdy są one większe niż dostępna pamięć RAM, lub wynikać z zastosowanego cache'owania. Możliwe jest włączenie szyfrowania tymczasowych plików. Domyślnie zastosowanym algorytmem jest HmacSHA1 z 128 bitowym kluczem.[25]

Część badawcza

4. Analiza udostępnionych interfejsów

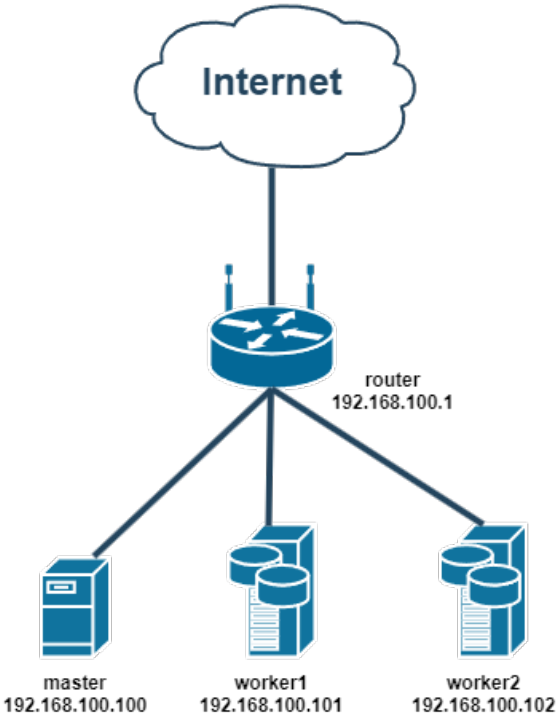
W tym rozdziale przedstawione zostaną wyniki przeprowadzonego badania udostępnionych interfejsów. Przeanalizowane zostały otwarte porty, metody stosowanej komunikacji i udostępnione witryny WWW. Szczególną uwagę poświęcono danym i zagrożeniom, jakie mógłby spowodować nieuprawniony dostęp lub podsłuchanie. W przypadku portów, które umożliwiają dodatkowe metody zabezpieczenia poza uwierzytelnianiem i szyfrowaniem, omówiono te sposoby zwiększania bezpieczeństwa. Dla portów wykorzystujących RPC pominięto możliwość wykorzystania ACL, ponieważ jak wykazano w rozdziale trzecim, jest to możliwe dla wszystkich implementacji tego protokołu.

4.1 Opis analizowanego systemu

Badania przeprowadzone zostały na klastrze składający się z trzech węzłów. Jednego węzła głównego oraz dwóch węzłów roboczych. Wszystkie działały w oparciu o system operacyjny Alpine Linux w wersji 3.20.3 na architekturze x86 w wersji 64-bitowej. Zainstalowano na nich Apache Hadoop w wersji 3.4.0 oraz Apache Spark w wersji 3.5.3, skompilowany do działania z Hadoopem w wersjach 3.3.0 i wyższych. Środowiskiem wykonawczym było OpenJDK w wersji 1.8.0_402. Kod w Scali napisano w oparciu o wersje 2.12. Schemat sieci zaprezentowano na rysunku 4.1.

4.2 Otwarte porty na węzłach klastra

Na rysunku 4.2 zaprezentowano porty nasłuchujące na węźle głównym, a na rysunku 4.3 porty nasłuchujące na węźle roboczym. Nasłuchują one nawet gdy nie jest wykonywane żadne zadanie. Port 22 jest odpowiedzialny za działanie serwera SSH, pozostałe związane są z działaniem Apache Hadoop i zostały omówione poniżej.



Rysunek 4.1 Analizowany system

Opracowanie własne

```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:9868            0.0.0.0:*               LISTEN      3774/java
tcp        0      0 0.0.0.0:9870            0.0.0.0:*               LISTEN      3519/java
tcp        0      0 master:39617            0.0.0.0:*               LISTEN      4068/java
tcp        0      0 master:8088             0.0.0.0:*               LISTEN      3975/java
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0      0 master:9000             0.0.0.0:*               LISTEN      3519/java
tcp        0      0 master:8030             0.0.0.0:*               LISTEN      3975/java
tcp        0      0 master:8031             0.0.0.0:*               LISTEN      3975/java
tcp        0      0 0.0.0.0:13562           0.0.0.0:*               LISTEN      4068/java
tcp        0      0 master:8042             0.0.0.0:*               LISTEN      4068/java
tcp        0      0 master:8040             0.0.0.0:*               LISTEN      4068/java
tcp        0      0 master:8032             0.0.0.0:*               LISTEN      3975/java
tcp        0      0 master:8033             0.0.0.0:*               LISTEN      3975/java
tcp6       0      0 [::]:22                 [::]:*                  LISTEN      -
```

Rysunek 4.2 Porty otwarte na węźle głównym

Opracowanie własne za pomocą komendy netstat -ltnp

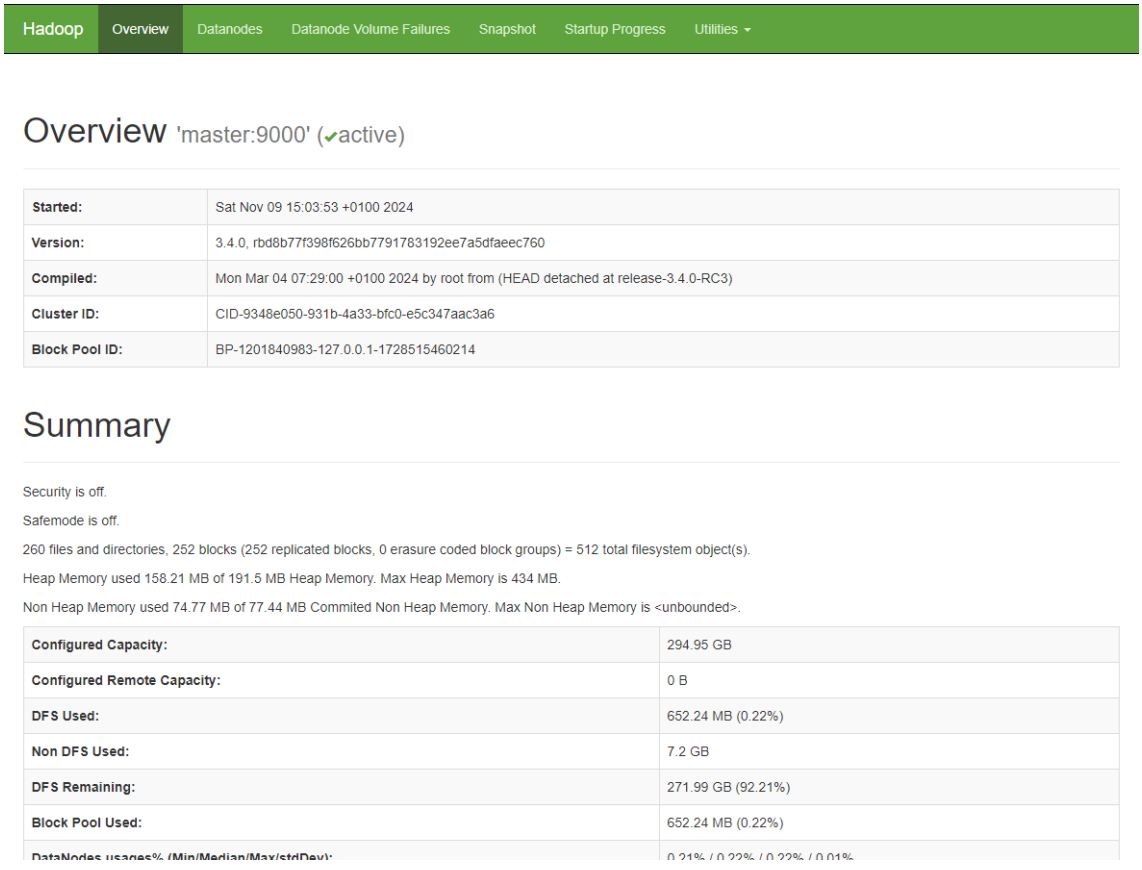
```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 worker1:40539           0.0.0.0:*               LISTEN      4053/java
tcp        0      0 worker1:8042            0.0.0.0:*               LISTEN      4053/java
tcp        0      0 worker1:8040            0.0.0.0:*               LISTEN      4053/java
tcp        0      0 0.0.0.0:9867            0.0.0.0:*               LISTEN      3948/java
tcp        0      0 0.0.0.0:9866            0.0.0.0:*               LISTEN      3948/java
tcp        0      0 0.0.0.0:9864            0.0.0.0:*               LISTEN      3948/java
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:44381         0.0.0.0:*               LISTEN      3948/java
tcp        0      0 0.0.0.0:13562           0.0.0.0:*               LISTEN      4053/java
tcp6       0      0 [::]:22                 [::]:*                  LISTEN      -
```

Rysunek 4.3 Porty otwarte na węźle roboczym

Opracowanie własne za pomocą komendy netstat -ltnp

4.3 Porty nasłuchujące na węźle głównym

4.3.1 Port 9870: Name Node, protokół HTTP



Rysunek 4.4 Interfejs WWW Name Node

Opracowanie własne

Na tym porcie uruchomiony jest interfejs WWW Name Node'a. Witrynę przedstawiono na rysunku 4.4. Dostępne narzędzia umożliwiają atak za pomocą wielu wekto-

rów. Udostępnione informacje mogą posłużyć do przygotowania kolejnych ataków. Wśród krytycznych udostępnionych danych można wymienić: rejestry zdarzeń, adresy węzłów roboczych, pliki konfiguracyjne, zawartość HDFS, wersje stosowanego oprogramowania, szczegóły uruchomionych procesów. Po uruchomieniu szyfrowania interfejs jest dostępny na porcie 9871. Wskazane jest zablokowanie lub ograniczenie ruchu na tym porcie do możliwie małego grona hostów. Należy zwrócić też uwagę na ustawienia webhdfs, czyli interfejsu służącego do przeglądania plików na HDFS za pomocą przeglądarki WWW. Oferowane są ustawienia mające uniemożliwiać ataki typu CSRF, które domyślnie są wyłączone. Ten interfejs może być zabezpieczony za pomocą ACL. Dostępne adresy przedstawiono w tabeli 4.1.

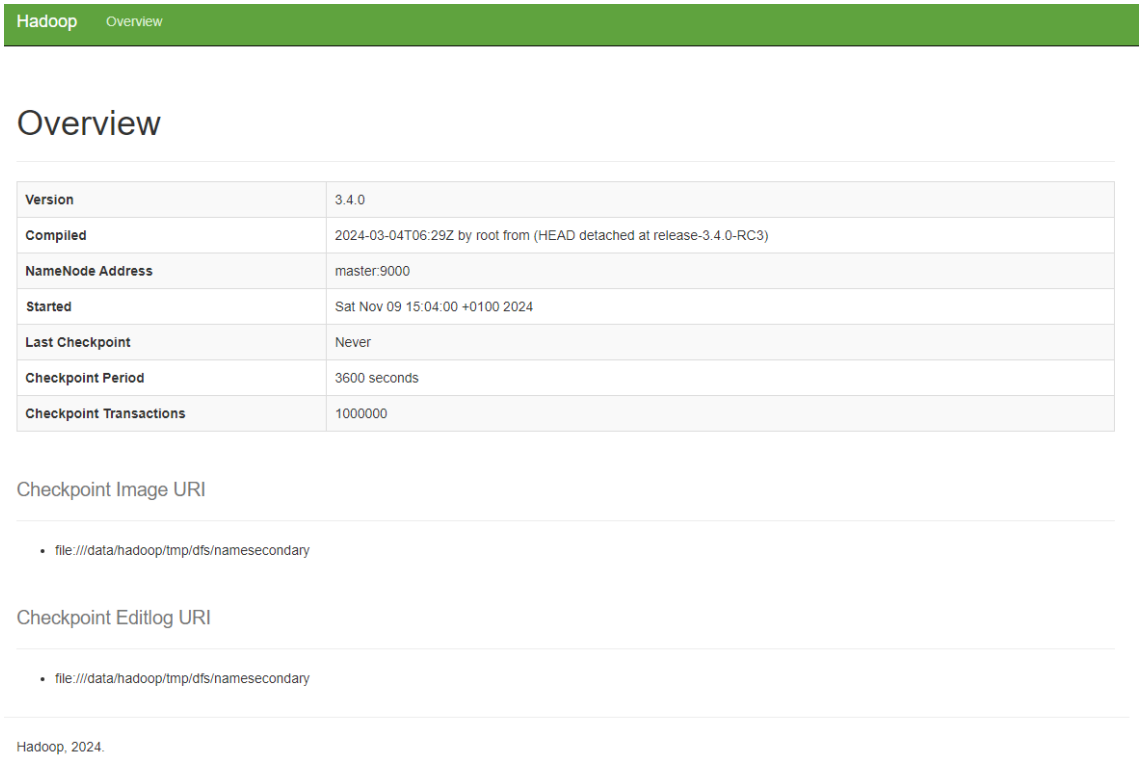
Tabela 4.1 Lista ścieżek HTTP Name Node

Ścieżka	Metoda	Opis
/dfshealth.html	GET	Plik HTML podzielony na sekcje odpowiedzialne za wyświetlanie poszczególnych podstron. Ich lista to: tab-overview, tab-datanode, tab-datanode-volume-failures, tab-snapshot, tab-startup-progress. Każde wczytanie sekcji, powoduje odświeżenie się treści strony, zgodnie z konwencją Single Page Application.
/static/*	GET	Pliki js oraz css związane z warstwą wizualną.
/static/rest-csrf.js	GET	Plik JavaScript zawierający narzędzia mające zapobiegać atakom typu CSRF. Wczytywany jest przez explorer.html.
/dfshealth.js	GET	Plik JavaScript, który pobiera informacje o Name Node oraz renderuje je na pobranym wcześniej HTML.
/startupProgress	GET	Plik JSON zawierający szczegóły o procesie uruchamiania się klastra.
/topology	GET	Plik tekstowy zawierający topologie sieci zastosowaną w klastrze.
/explorer.js	GET	Plik JavaScript zawierający narzędzia mające umożliwić prace z webHDFS.
/explorer.html	GET	Plik HTML dla narzędzia do przeglądania zawartości HDFS.
/webhdfs/v1/*	GET,PUT,DELETE	Scieżka używana do wykonywania operacji za pomocą webHDFS.
/conf	GET	Plik xml zawierający konfigurację węzła.
/jmx?query=	GET	Zapytanie o konkretne elementy konfiguracji, wysyłane przez snn.js, aby pobrać dane.

Ścieżka	Metoda	Opis
/jmx	GET	Plik JSON zawierający konfigurację węzła.
/logs	GET	Wyświetla rejestry zdarzeń dostępne na maszynie, gdzie Name Node jest udostępniony.
/logs/jetty-dir.css	GET	Plik css dla listy rejestrów zdarzeń.
/logLevel	GET	Aplikacja pozwalająca sprawdzić poziom rejestrowania zdarzeń dla każdej klasy Java.
/logLevel?log=	GET	Zapytanie o poziom rejestrowania zdarzeń dla konkretnej klasy Java.
/stacks	GET	Lista uruchomionych procesów.

Opracowanie własne

4.3.2 Port 9868: Secondary Name Node, protokół HTTP



Rysunek 4.5 Interfejs WWW Seconday Name Node

Opracowanie własne

Na tym porcie uruchomiony jest interfejs WWW Secondary Name Node'a, jednej z usług Apache Hadoop. Interfejs przedstawiono na rysunku 4.5. Zawiera on informacje o wersji, z której korzysta klaster, datę jego uruchomienia i szczegóły konfiguracji check-

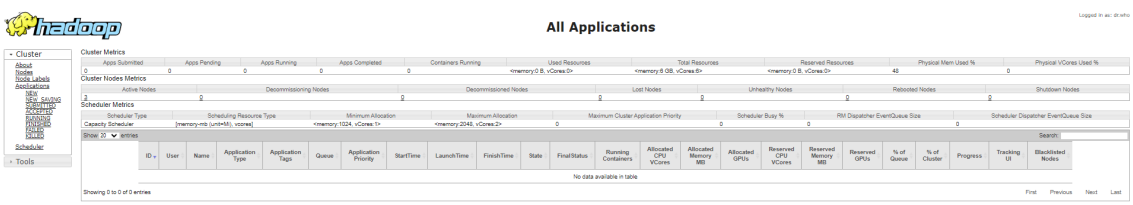
Tabela 4.2 Lista ścieżek HTTP Secondary Name Node

Ścieżka	Metoda	Opis
/status.html	GET	Plik zawierający HTML.
/static/*	GET	Pliki js oraz css związane z warstwą wizualną.
/snn.js	GET	Plik JavaScript używany do pobrania danych o platformie Hadoop wyświetlanych na witrynie.
/jmx?query=	GET	Zapytanie o konkretne elementy konfiguracji, wysyłane przez snn.js, aby pobrać dane.
/jmx	GET	Plik JSON zawierający konfigurację węzła Apache Hadoop.
/logs	GET	Wyświetla rejestry zdarzeń dostępne na maszynie, gdzie Seconaday Name Node jest udostępniony.
/logs/jetty-dir.css	GET	Plik css dla listy rejestrów zdarzeń.
/logLevel	GET	Aplikacja pozwalająca sprawdzić poziom rejestrowania zdarzeń dla każdej klasy Java.
/logLevel?log=	GET	Zapytanie o poziom rejestrowania zdarzeń dla konkretnej klasy Java.
/stacks	GET	Lista uruchomionych procesów.
/conf	GET	Plik xml zawierający konfigurację węzła.

Opracowanie własne

pointu wykonywanego regularnie. Po uruchomienie HTTPS jest dostępny na porcie 9869. Jeśli witryna nie będzie aktywnie używana, zalecane jest ograniczenie ruchu na tym porcie przy użyciu firewalla. Możliwe jest skonfigurowanie ACL dla tego kanału dostępu. Dostępne adresy przedstawiono w tabeli 4.2.

4.3.3 Port 8088: Resource Manager, protokół HTTP



Rysunek 4.6 Interfejs WWW Resource Managera

Opracowanie własne

Interfejs WWW Resource Managera pozwala na wyświetlenie zakończonych i obecnie wykonywanych zadań. Witrynę przedstawiono na rysunku 4.6. Pozwala na przeglądanie zasobów klastra, takich jak m.in. dostępne węzły, zdefiniowane kolejki, dostępne procesory i pamięć RAM. Dodatkowo udostępnia narzędzie, które pozwalają na odczyt konfiguracji i rejestrów zdarzeń. Przy domyślnych ustawieniach możliwe jest zakończenie trwających zadań za pomocą wyświetlanego przycisku. Z przyczyn bezpieczeństwa warto go wyłączyć, tak aby nie był wyświetlany. W przypadku zastosowanie HTTPS interfejs dostępny jest na porcie 8090. Dostępne adresy przedstawiono w tabeli 4.4.

Tabela 4.3 Lista ścieżek HTTP Resource Managera

Ścieżka	Metoda	Opis
/cluster	GET	Plik HTML zawierający treść witryny Resource Managera.
/static/*	GET	Pliki css oraz js oraz obrazy w formacie png zawierające elementy witryny.
/cluster/cluster	GET	Plik HTML wyświetlający szczegółowy opis klastra.
/cluster/nodes	GET	Plik HTML wyświetlający informacje o węzłach klastra.
/cluster/nodelabels	GET	Plik HTML wyświetlający informacje o etykietach węzłów.
/cluster/apps	GET	Plik HTML wyświetlający listę wszystkich aplikacji.
/custer/apps/<identyfikator>	GET	Plik HTML wyświetlający informacje o aplikacji posiadającej podany identyfikator.
/cluster/appattempt/ <identyfikator>	GET	Plik HTML wyświetlający informacje o próbie wykonania aplikacji posiadającej podany identyfikator.
/cluster/apps/<status>	GET	Plik HTML wyświetlający listę wszystkich aplikacji o wybranym statusie. Dostępne statusy to: NEW, NEW_SAVING, SUBMITTED, ACCEPTED, RUNNING, FINISHED, FAILED, KILLED.
/cluster/nodes/<status>	GET	Plik HTML wyświetlający listę wszystkich węzłów o wybranym statusie. Dostępne statusy to: decommissioning, decommissioned, lost, unhealthy, rebooted, shutdown.
/cluster/scheduler	GET	Plik HTML wyświetlający informacje o schedulerze i kolejkach.
/ws/v1/cluster/scheduler/logs	GET	Zapytanie powodujące wygenerowanie rejestru zdarzeń schedulera.
/jmx?query=Hadoop:*	GET	Plik JSON zawierający konfigurację węzła Apache Hadoop.
/jmx	GET	Plik JSON zawierający konfigurację węzła Apache Hadoop.

Ścieżka	Metoda	Opis
/logs	GET	Wyświetla rejestry zdarzeń dostępne na maszynie, gdzie Resource Manager jest uruchomiony.
/logs/jetty-dir.css	GET	Plik css dla listy rejestrów.
/stacks	GET	Lista uruchomionych procesów.
/conf	GET	Plik xml zawierający konfigurację węzła.

Opracowanie własne

4.3.4 Port 9000: Name Node, protokół RPC

Wybór numeru portu następuje w czasie konfiguracji klastra, zazwyczaj wykorzystywany jest numer 9000 lub 8020. Na tym porcie na głównym węźle nasłuchuje Name Node. Ten kanał komunikacji korzysta z takich implementacji protokołów RPC jak: HAServiceProtocol, ReconfigurationProtocol, NamenodeProtocol, DatanodeProtocol, RefreshAuthorizationPolicyProtocol, RefreshUserMappingsProtocol, RefreshCallQueueProtocol, GenericRefreshProtocol, GetUserMappingsProtocol. Przy użyciu tego portu odbywa się większość operacji w obrębie klastra Hadoop. Wybrane metody, które mogłyby zostać wykorzystane w ataku na ten port, przedstawiono w tabeli 4.4.

Tabela 4.4 Wybrane metody, które można wykorzystać do ataku na Name Node za pomocą RPC

Metoda	Protokół	Opis
delete	ClientProtocol	Usunięcie plików lub katalogów.
setPermission	ClientProtocol	Zmiana uprawnień pliku.
setOwner	ClientProtocol	Zmiana właściciela pliku.

Opracowanie własne

Warto za pomocą ACL ograniczyć możliwość korzystania z protokołów: RefreshAuthorizationPolicyProtocol, RefreshUserMappingsProtocol, RefreshCallQueueProtocol, GenericRefreshProtocol i ReconfigurationProtocol.

4.3.5 Port 8030: Yarn Scheduler, protokół RPC

Na tym porcie nasłuchuje Yarn Scheduler, który jest częścią Resource Managera. Komunikacja odbywa się za pomocą RPC przy pomocy protokołu ApplicationMasterProtocol. Application Master używa tego kanału, aby zarejestrować swoją obecność, wyrejestrować się lub zgłosić zapotrzebowanie na zasoby. Wykorzystanie tego protokołu,

daje atakującemu możliwość zarejestrowania własnego Application Mastera. Dzięki jego użyciu może wnioskować o udostępnienie zasobów klastra i uruchamianie procesów, także na innych węzłach roboczych.

4.3.6 Port 8032: Resource Manager, protokół RPC

Na tym porcie przy pomocy RPC nasłuchuje Resource Manage, który wykorzystuje protokół ApplicationClientProtocol. Komunikacja odbywa się między klientami a Resource Managerem. Dostępne metody pozwalają na zlecanie nowych zadań, przerywanie istniejących, a także pobieranie informacji o zadaniach, węzłach, kolejkach i ACL. Wybrane metody, które mogłyby zostać wykorzystane w ataku na ten port, przedstawiono w tabeli 4.5.

Tabela 4.5 Wybrane metody, które można wykorzystać do ataku na Resource Managera za pomocą RPC

Metoda	Protokół	Opis
failApplicationAttempt	ApplicationClientProtocol	Przerwanie próby wykonania aplikacji.
forceKillApplication	ApplicationClientProtocoll	Przerwanie wykonywanej aplikacji.
submitApplication	ApplicationClientProtocol	Zlecenie wykonania nowej aplikacji.

Opracowanie własne

4.3.7 Port 8033: Resource Manager, interfejs administratorski, protokół RPC

Za pomocą tego portu i RPC odbywa się komunikacja między administratorami klastra a ResourceManager. Wykorzystywany protokół to ResourceManagerAdministrationProtocol. Dodatkowo implementacja dziedziczy z GetUserMappingsProtocol. Dostępne metody pozwalają na pobieranie wielu krytycznych zmiennych, odświeżanie konfiguracji i nadpisywanie niektórych ustawień. Metody, takie jak replaceLabelsOnNode, removeFromClusterNodeLabels czy addToClusterNodeLabels mogą być wykorzystane do zmiany etykiet węzłów co może doprowadzić do zakłócenia pracy Resource Managera. Zalecane jest ograniczenie ruchu na tym porcie do zaufanych hostów.

4.4 Porty nasłuchujące na węzłach roboczych

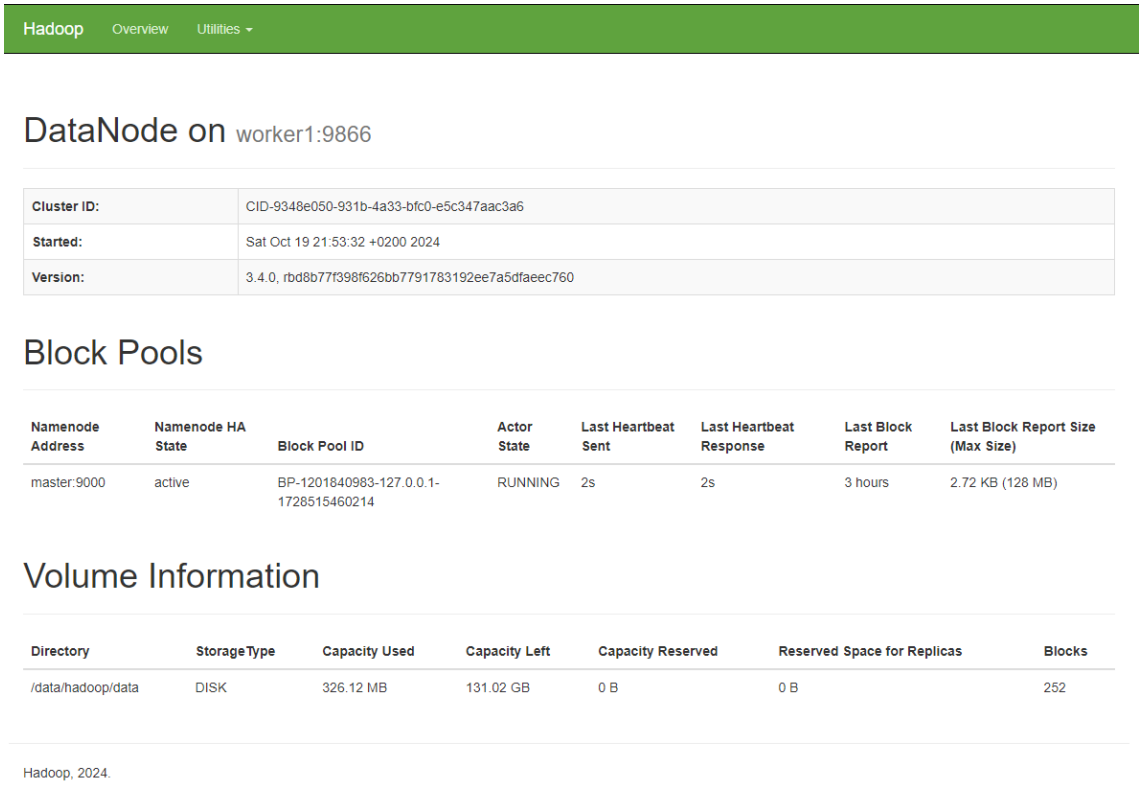
4.4.1 Port 9867: Data Node, protokół RPC

Na tym porcie nasłuchuje Data Node za pomocą RPC. Wykorzystywane protokoły to InterDatanodeProtocol, ClientDatanodeProtocol i ReconfigurationProtocol. Komunikacja na tym porcie odbywa się między Data Node a jego klientami i innymi węzłami roboczymi. Name Node nigdy nie rozpoczyna komunikacji z żadnym z Data Node'ów. Komunikacja między nimi zawsze rozpoczynana jest przez Data Node. Możliwe jest wyłączenie procesu przy pomocy metody shutdownDatanode z protokołu ClientDatanodeProtocol.

4.4.2 Port 9866: Data Node, protokół TCP/IP, Data Transfer

Na tym porcie Data Node nasłuchuje za pomocą java.net.ServerSocket. Port 9866 jest wybierany domyślnie gdy zabezpiecznie nie są uruchomione. Używany jest do przesyłania danych między Data Nodem a innymi węzłami i klientami. Zastosowanie prostej implementacji opartej o ServerSocket jest kluczowe dla wydajności. Domyślnie ruch na tym porcie nie jest szyfrowany, ani w żaden sposób uwierzytelniany. Istnieje możliwość włączenia szyfrowania za pomocą algorytmu AES. W starszych wersjach Apache Hadoop zalecane było uruchomienie tej usługi za pomocą portów uprzywilejowanych (ang. privileged ports), czyli portów poniżej numeru 1024. Używanie tych uprawnień służyło jako proces uwierzytelniania. Obecnie w tym celu używana jest wymiana kluczy. Platforma nadal umożliwia uruchomienie Data Node jako root w celu zajęcia takiego portu i zmianę użytkownika na wybranego dla HDFS. Korzystanie z tej możliwości nie jest już wymagane, ale nadal zalecane przez wielu administratorów klastrów Hadoop.

4.4.3 Port 9864: Data Node, protokół HTTP



Rysunek 4.7 Interfejs WWW Data Node'a

Opracowanie własne

Tabela 4.6 Lista ścieżek HTTP Data Node'a

Ścieżka	Metoda	Opis
/datanode.html	GET	Plik HTML zawierający treść witryny Data Node.
/static/*	GET	Pliki .js oraz .css związane z warstwą wizualną.
/dn.js	GET	Plik JavaScript, który odpowiedzialny jest za pobranie danych wyświetlanych przez datanode.html.
/jmx?query=	GET	Plik JSON zawierający konfigurację węzła.
/jmx	GET	Plik JSON zawierający konfigurację węzła Apache Hadoop.
/logs	GET	Wyświetla rejestry zdarzeń dostępne na maszynie, gdzie Resource Manager jest uruchomiony.
/logs/jetty-dir.css	GET	Plik css dla listy rejestrów.
/stacks	GET	Lista uruchomionych procesów.
/conf	GET	Plik xml zawierający konfigurację węzła.

Opracowanie własne

Tabela 4.7 Wybrane metody, które można wykorzystać do ataku na Node Managera za pomocą RPC

Metoda	Protokół	Opis
startContainers	ContainerManagementProtocol	Uruchomienie kontenerów
stopContainers	ContainerManagementProtocol	Wyłączenie kontenerów
restartContainer	ContainerManagementProtocol	Przerwanie pracy i ponowne uruchomienie kontenerów.

Opracowanie własne

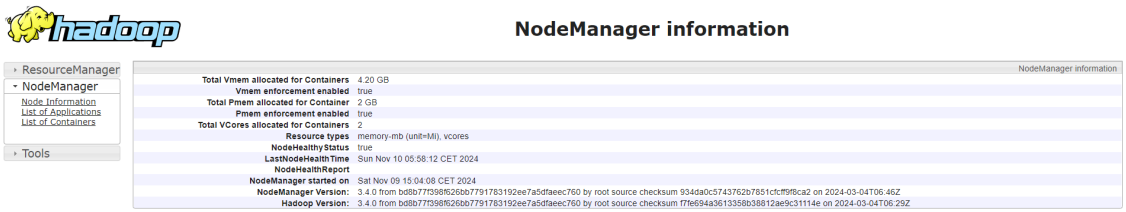
Na tym porcie nasłuchuje interfejs WWW Data Node'a. Za pomocą tej witryny możemy sprawdzić szczegóły związane z obecny zużyciem przestrzeni dyskowej i konfiguracji. Dodatkowo udostępnione są narzędzia, które pozwalają na przeglądanie rejestrów zdarzeń i listy działających procesów. Gdy uruchomiony jest protokół HTTPS, strona nasłuchuje na porcie 9865. Witrynę przedstawiono na rysunku 4.7. Z punktu widzenia bezpieczeństwa wrażliwe są rejestry zdarzeń, pliki konfiguracyjne, wersja stosowanego oprogramowania i szczegóły uruchomionych procesów. Podobnie jak w przypadku Data Transfer zalecane jest skorzystanie z portów uprzywilejowanych. W plikach konfiguracyjnych można zdefiniować ACL dla tego interfejsu. Dostępne adresy przedstawiono w tabeli 4.6.

4.5 Porty nasłuchujące na wszystkich węzłach

4.5.1 Porty 39617 i 40539: Node Manager, protokół RPC

Numer tego portu jest wybierany losowo za każdym razem gdy uruchamiany jest Node Manager. Na rysunku 4.2 możemy zauważyć tę usługę na porcie 39617, a na rysunku 4.3 na porcie 40539. Służy ona do obsługiwania ContainerManagementProtocol. Ten protokół opiera się o RPC i jest wykorzystywany do uruchamiania kontenerów, restartowania i przerywania ich pracy oraz przydzielania zasobów. Komunikacja odbywa się między Node Managerem a Application Masterem. Gdy uwierzytelnianie jest włączone, Node Manager dodatkowo weryfikuje, korzystając z Resource Managera, czy zasoby zostały przypisane do wnioskującego o nie procesu. Wybrane metody, które mogłyby zostać wykorzystane w ataku na ten port, przedstawiono w tabeli 4.7.

4.5.2 Port 8042: Node Manager, protokół HTTP



Rysunek 4.8 Interfejs WWW Node Managera

Opracowanie własne

Tabela 4.8 Lista ścieżek HTTP Node Managera

Ścieżka	Metoda	Opis
/node	GET	Plik HTML zawierający treść witryny Node Managera.
/static/*	GET	Pliki css oraz js oraz obrazy w formacie png zawierające elementy witryny.
/node/allApplications	GET	Lista aplikacji wykonywanych na danym węźle.
/node/allContainers	GET	Lista kontenerów Yarna działających na danym węźle.
/jmx?query=Hadoop:*	GET	Plik JSON zawierający konfigurację węzła.
/jmx	GET	Plik JSON zawierający konfigurację węzła Apache Hadoop.
/logs	GET	Wyświetla rejestry zdarzeń dostępne na maszynie, gdzie Resource Manager jest uruchomiony.
/logs/jetty-dir.css	GET	Plik css dla listy rejestrów.
/stacks	GET	Lista uruchomionych procesów.
/conf	GET	Plik xml zawierający konfigurację węzła.

Opracowanie własne

Na tym porcie udostępniony jest interfejs WWW Node Manager. Interfejs przedstawiono na rysunku 4.8. Za jego pomocą można sprawdzić ustawienia węzła, przechowywane na nim rejestry zdarzeń, uruchomione aplikacje, kontenery Yarn i inne procesy. Te informacje można wykorzystać do przygotowania kolejnych ataków. Gdy uruchomiony jest protokół HTTPS, witryna jest dostępna na porcie 8044. Dostępne adresy przedstawiono w tabeli 4.8.

4.5.3 Port 8040: Resource Localization Service, protokół RPC

Na tym porcie nasłuchuje Resource Localization Service. Jest to usługa, będąca częścią Node Managera, odpowiedzialna za zarządzanie plikami. Zajmuje się kopiowaniem plików z innych węzłów oraz lokalnego systemu plików w taki sposób, aby były one dostępne dla uruchamianych aplikacji.

4.5.4 Port 13562: MapReduce Shuffle Handler, protokół HTTP

Na tym porcie nasłuchuje MapReduce Shuffle Handler. Jest to usługa używana przez MapReduce w czasie wykonywania zadań. Wykorzystywany jest protokół HTTP, jednak jest to REST API, a nie witryna internetowa. Przy domyślnych ustawieniach dane nie są szyfrowane ani w czasie transportu, ani przechowywania, jeśli w trakcie wykonywania zadania nastąpi zapis na dysk. Istnieje możliwość uruchomienia szyfrowania danych w transporcie oraz danych tymczasowo zapisanych na dysku.

4.6 Atak na protokół RPC

W czasie analizy dostępnych interfejsów zauważono, że większość kanałów komunikacji wykorzystuje protokoły HTTP lub RPC. Szczególnie atak na protokół RPC byłby znaczącym zagrożeniem dla bezpieczeństwa klastra opartego o Apache Hadoop. Platforma udostępnia wiele metod, które atakujący mógłby wykorzystać. W ramach tej pracy autor przygotował atak, mogący wykorzystać protokół ClientDataNodeProtocol do wyłączenia jednego z Data Nodów. Aby skutecznie przeprowadzić atak na protokół RPC, atakujący musi zgromadzić następujące informacje:

- adres IP atakowanego węzła,
- numer portu, na którym nasłuchiwany jest protokół RPC,
- sygnatury obsługiwanych metod,
- wersje wykorzystywanego protokołu,
- dane uwierzytelniające użytkownika, który ma odpowiednie uprawnienia.

Użytkownik mający dostęp do systemu zazwyczaj zna jego podstawową konfigurację taką jak wersja Hadoop czy adresy IP węzłów. W przeciwnym wypadku możliwe jest ich ustalenie przez wykorzystanie podstawowych narzędzi sieciowych lub wyszukanie ich w rejestrach zdarzeń. Właściwy port można ustalić, korzystając z komendy telnet, co może

nie być konieczne, ponieważ często stosowany jest domyślny numer. Platforma Apache Hadoop jest otwartym oprogramowaniem, co ułatwia zadanie ustalenia sygnatur metod i wersji wykorzystywanego protokołu. Użycie pliku jar zawierającego kod źródłowy pozwala na wykorzystanie tego samego kodu, który wykorzystywany jest przez uprawnione procesy. Nie jest konieczne zastosowanie dokładnie tej samej wersji oprogramowania. Wymagane jest, tylko aby wersja stosowanego protokołu zgadzała się z tą zastosowaną na klastrze. Konieczne jest ustalenie, którzy użytkownicy posiadają wysoki poziom dostępu. Rozpoznanie tego możliwe jest przez sprawdzenie kto jest właścicielem lokalizacji / na hdfs. Istnieje duża szansa, że ten użytkownik będzie posiadał wysokie uprawnienia. Przy domyślnych ustawieniach zabezpieczeń nie jest konieczna znajomość szczegółów uwierzytelniających, ponieważ Apache Hadoop nie zweryfikuje w żaden sposób czy atakujący faktycznie ma dostęp do konta użytkownika, którego nazwy użyje.

4.6.1 Projekt aplikacji atakującej protokół RPC

Przy projektowaniu narzędzia ataku wykorzystano język Java, tak aby można było skorzystać z kodu źródłowego Apache Hadoop. W tym celu użyto narzędzia Maven i dodano zależności widoczne na rysunku 4.9.

```
<dependencies>
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-common</artifactId>
        <version>3.4.0</version>
    </dependency>
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-client</artifactId>
        <version>3.4.0</version>
    </dependency>
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-hdfs</artifactId>
        <version>3.4.0</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

Rysunek 4.9 Zależności umożliwiające skorzystanie z kodu źródłowego Apache Hadoop

Opracowanie własne

Do ataku można wykorzystać jedną z wielu klas implementujących interfejs Pro-

tocolTransalator. W tym przypadku wykorzystano ClientDataNodeProtocolTranslatorPB co można zobaczyć na rysunku 4.10.

```
public static void main(String[] args) {
    try {
        InetSocketAddress address = new InetSocketAddress( hostname: "worker2",  port: 9867);
        Configuration conf = new Configuration();
        ClientDatanodeProtocolTranslatorPB translatorPB = new ClientDatanodeProtocolTranslatorPB(
                address,
                UserGroupInformation.createRemoteUser("hduser"),
                conf,
                NetUtils.getDefaultSocketFactory(conf));
        translatorPB.shutdownDatanode( forUpgrade: false);
        System.out.println("Datanode shutdown");
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

Rysunek 4.10 Kod źródłowy ataku na protokół RPC

Opracowanie własne

Aby atak zadziałał prawidłowo, wymagane jest wprowadzenia adresu i portu atakowanej usługi w konstruktorze InetSocketAddress, oraz nazwy uprawnionego użytkownika jako argument metody UserGroupInformation.createRemoteUser. Wykonanie translatorPC.shutdownDatanode(false) powoduje wysłanie instrukcji przy pomocy RPC, a po jej odebraniu Data Node zakończy pracę. Podobne ataki można przeprowadzić też na inne wykorzystywane w Apache Hadoop protokoły. Dostępnym zabezpieczeniem jest uruchomienie uwierzytelniania za pomocą Kerberos.

4.7 Atak na HDFS

Komunikacja z HDFS odbywa się za pomocą dwóch kanałów. Polecenia związane z manipulowaniem plikami i pobieraniem metadanych przesyłane są za pomocą protokołu RPC udostępnionego na porcie 9000 węzła głównego. Pisane i czytane dane przesyłane są między klientem a Data Nodem, który wskazał Name Node. Odbywa się to na porcie 9866 przy pomocy protokołu TCP/IP w oparciu o ServerSocket. Z tego powodu część ataków na HDFS jak np. usunięcie pliku, zmiana uprawnień czy zmiana nazwy może być dokonana przy pomocy protokołu RPC, ale ataki mające na celu pobranie danych z klastra lub przesłanie ich na klaster będą wymagać wieloetapowego ataku na więcej niż jeden węzeł i port oraz zastosowanie dwóch form komunikacji. Do przeprowadzenia ataku potrzebne są te same informacje, które wymagane są przy ataku na protokół RPC.

W tym przypadku atakowaną maszyną jest węzeł główny, a informacje o nim są zwykle dostępne dla wszystkich użytkowników klastra. Nie jest wymagana znajomość informacji o żadnym Data Node'dzie, ponieważ te wskaże Name Node. Celem ataku w tej pracy będzie wypisanie zawartości katalogu, utworzenie nowego katalogu i pobranie jednego z plików znajdujących się na HDFS.

4.7.1 Projekt aplikacji atakującej HDFS

Tak jak w przypadku ataku na protokół RPC fakt, że Hadoop jest oprogramowaniem otwartym, daje możliwość użycia gotowego kodu źródłowego. W czasie przygotowania tego ataku wykorzystano te same zależności jak w przypadku ataku na RPC. Nie jest konieczne wykorzystanie klas implementujących interfejs ProtocolTranslator, ponieważ istnieje klasa org.apache.hadoop.FileSystem, która implementuje wszystkie potrzebne metody. Wykorzystanie jej sprawia, że nie jest wymagane ręczne wykonywanie zapytań RPC do węzła głównego, ani implementacja połączenia z Data Nodem. Na rysunku 4.11 przedstawiono kod źródłowy aplikacji, która wypisuje zawartość katalogu /user/hduser, tworzy w nim katalog test_directory, a następnie pobiera na lokalny komputer plik life_expectancy.csv.

```
public static void main(String[] args) {
    try {
        Configuration conf = new Configuration();
        String url = "hdfs://master:9000";
        FileSystem fs = FileSystem.get(URI.create(url), new Configuration(), user: "hduser");

        RemoteIterator<LocatedFileStatus> it = fs.listFiles(new Path( pathString: "/user/hduser"), recursive: false);
        while(it.hasNext()) System.out.println(it.next().toString());
        System.out.println("Done listing");
        fs.mkdirs(new Path( pathString: "test_directory"));
        FileOutputStream out = new FileOutputStream( name: "life_expectancy.csv");
        FSDataInputStream in = fs.open(new Path( pathString: "/user/hduser/life_expectancy.csv"));
        IOUtils.copyBytes(in, out, conf);

        System.out.println("Done downloading");
    } catch (IOException | InterruptedException e) {
        throw new RuntimeException(e);
    }
}
```

Rysunek 4.11 Kod źródłowy ataku na HDFS

Opracowanie własne

W czasie tworzenia obiektu klasy FileSystem należy zdefiniować adres i port węzła głównego, a dodatkowo można wybrać użytkownika, który będzie użyty w czasie procesu

uwierzytelniania. Tak jak w przypadku nieuprawnionego wykorzystania protokołu RPC, jedyną skuteczną metodą zapobieganiu takim atakom jest zastosowanie uwierzytelniania opartego o protokół Kerberos.

5. Analiza ruchu sieciowego

W tym rozdziale przedstawione zostaną wyniki przeprowadzonej analizy ruchu sieciowego. Przechwycono pakiety, które wygenerowane zostały w trakcie wykonywania zadania przez Apache Spark. Wyniki przedstawiono w tabeli nr 5.1. Pominięto w nich pakiety odpowiedzialne za uwierzytelnianie, przesyłanie tzw. heart beatów i odpowiedzi serwerów. Dodatkowo uproszczono zapis komunikacji pomiędzy Executorem a Driverem, który składa się z wielu zapytań i odpowiedzi dotyczących statusu wykonywanych Tasków oraz instrukcji, jakie mają być wykonane. Aby zwiększyć czytelność tabeli, niektóre nazwy plików zostały uproszczone.

Wykonywanym zadaniem było wyliczenie średniej długości życia w krajach rozwijających się i rozwiniętych. Danymi źródłowymi był plik /user/hduser/life_expectancy.csv zawierający dane WHO o średniej długości życia obywateli wszystkich krajów świata oraz podział na kraje rozwijające się i rozwinięte. Wynik zapisano w katalogu na HDFS. Wykorzystana ścieżka to /user/hduser/life_expectancy_result.csv. Aplikacja została skonfigurowana w taki sposób, aby utworzyła proces Driver w obrębie klastra oraz jeden proces typu Executor. Porty poszczególnych usług były skonfigurowane tak samo jak w rozdziale 4, poza Node Managerami, które uruchomione były na portach 39543 na węźle głównym, 35429 na worker1 i 32925 na worker2.

Tabela 5.1 Ruch sieciowy w badanym klastrze

Lp.	Czas	Kierunek	Opis
1	20:42:21	master:40408→worker1:9866	Przesyłanie bibliotek Spark.
2	20:42:21	worker1:38206→worker2:9866	Przesyłanie bibliotek Spark.
3	20:42:22,323	worker2:51112→master: 9000	Wywołanie metody: blockReceivedAndDeleted z DatanodeProtocol.
4	20:42:22,435	worker1:35218→master:9000	Wywołanie metody: blockReceivedAndDeleted z DatanodeProtocol.
5	20:42:26,958	worker2:51112→master: 9000	Wywołanie metody: blockReceivedAndDeleted z DatanodeProtocol.
6	20:42:26,960	worker1:35218→master:9000	Wywołanie metody: blockReceivedAndDeleted z DatanodeProtocol.

Lp.	Czas	Kierunek	Opis
7	20:42:26,967	master:50904→worker2:9866	Przesyłanie plików konfiguracyjnych Spark.
8	20:42:25,989	worker2:3850→worker1:9866	Przesyłanie plików konfiguracyjnych Spark.
9	20:42:28,317	worker2:51112→master: 9000	Wywołanie metody: blockReceivedAndDeleted z DatanodeProtocol.
10	20:42:28,319	worker1:35218→master:9000	Wywołanie metody: blockReceivedAndDeleted z DatanodeProtocol.
11	20:42:30,767	worker1:35218→master:9000	Wywołanie metody: blockReceivedAndDeleted z DatanodeProtocol.
12	20:42:30,771	worker2:51112→master: 9000	Wywołanie metody: blockReceivedAndDeleted z DatanodeProtocol.
13	20:42:30,779	master:50908→worker2:9866	Przesyłanie plików.
14	20:42:30,785	worker2:38354→worker1:9866	Przesyłanie plików.
15	20:42:31,315	worker1:35218→master:9000	Wywołanie metody: blockReceivedAndDeleted z DatanodeProtocol.
16	20:42:31,318	worker2:51112→master:9000	Wywołanie metody: blockReceivedAndDeleted z DatanodeProtocol.
17	20:42:32,597	worker1:35218→master:9000	Wywołanie metody: blockReceivedAndDeleted z DatanodeProtocol.
18	20:42:32,598	worker2:51112→master:9000	Wywołanie metody: blockReceivedAndDeleted z DatanodeProtocol.
19	20:42:32,711	master:40710→worker1:40710	Przesyłanie whodata.jar.
20	20:42:32,719	worker1:59608→worker2:9866	Przesyłanie whodata.jar.
21	20:42:32,723	worker2:51112→master:9000	Wywołanie metody: blockReceivedAndDeleted z DatanodeProtocol.
22	20:42:32,725	worker1:35218→master:9000	Wywołanie metody: blockReceivedAndDeleted z DatanodeProtocol.
23	20:42:32,963	master:40720→worker1:9866	Przesyłanie konfiguracji.
24	20:42:32,977	worker1:59610→worker2:9866	Przesyłanie konfiguracji.
25	20:42:32,987	worker2:51112→master: 9000	Wywołanie metody: blockReceivedAndDeleted z DatanodeProtocol.
26	20:42:32,989	worker1:35218→master:9000	Wywołanie metody: blockReceivedAndDeleted z DatanodeProtocol.
27	20:42:34:103	master:51112→worker1:35429	Wywołanie metody: blockReceivedAndDeleted z DatanodeProtocol.
28	20:42:35,576	worker1:53702→master:9000	Wywołanie metody: getFileInfo spark_libs.zip z ClientProtocol.

Lp.	Czas	Kierunek	Opis
29	20:42:35,609	worker1:53702→master:9000	Wywołanie metody: getFileInfo spark_libs.zip z ClientProtocol.
30	20:42:35,726	worker1:53702→master:9000	Wywołanie metody: getServerDefaults z ClientProtocol.
31	20:42:41,101	worker1:53702→master:9000	Wywołanie metody: getFileInfo spark_conf.zip z ClientProtocol.
32	20:42:41,103	worker1:53702→master:9000	Wywołanie metody: getBlocksLocations spark_conf.zip z ClientProtocol.
33	20:42:41,531	worker1:53702→master:9000	Wywołanie metody: getFileInfo whodata.jar z ClientProtocol.
34	20:42:41,533	worker1:53702→master:9000	Wywołanie metody: getBlocksLocations whodata.jar z ClientProtocol.
35	20:42:46,183	worker1:51332→master:8030	Wywołanie metody: getBlocksLocations whodata.jar z ClientProtocol.
36	20:42:46,283	worker1:40090→master:9000	Wywołanie metody: getBlocksLocations whodata.jar z ClientProtocol.
37	20:42:46,572	worker1:51332→master:8030	Wywołanie metody: allocate z ApplicationMasterProtocol.
38	20:42:46,606	worker1:51332→master:8030	Wywołanie metody: allocate z ApplicationMasterProtocol.
39	20:42:46,811	worker1:51332→master:8030	Wywołanie metody: allocate z ApplicationMasterProtocol.
40	20:42:46,944	worker1:50488→worker2:32925	Wywołanie metody: startContainers z ContainerManagementProtocol.
41	20:42:48,026	worker2:35666→master:9000	Wywołanie metody: getFileInfo spark_libs.zip z ClientProtocol.
42	20:42:48,059	worker2:35666→master:9000	Wywołanie metody: getBlocksLocations spark_libs.zip z ClientProtocol.
43	20:42:48,130	worker2:35666→master:9000	Wywołanie metody: getServerDefaults z ClientProtocol.
44	20:42:49,835	worker1:51332→master:8030	Wywołanie metody: allocate z ApplicationMasterProtocol.
45	20:42:52,841	worker1:51332→master:8030	Wywołanie metody: allocate z ApplicationMasterProtocol.
46	20:42:53,373	worker2:35666→master:9000	Wywołanie metody: getFileInfo spark_conf.zip z ClientProtocol.

Lp.	Czas	Kierunek	Opis
47	20:42:53,380	worker2:35666→master:9000	Wywołanie metody: getBlocksLocations spark_conf.zip z ClientProtocol.
48	20:42:53,784	worker2:35666→master:9000	Wywołanie metody: getFileInfo whodata.jar z ClientProtocol.
49	20:42:53,784	worker2:35666→master:9000	Wywołanie metody: getBlocksLocations whodata.jar z ClientProtocol.
50	20:42:55,848	worker1:51332→master:8030	Wywołanie metody: allocate z ApplicationMasterProtocol.
51	20:42:56,733	worker2:38530→worker1:45845	Wywołanie metody: CoarseGrainedScheduler checkExistence.
52	20:42:56,798	worker2:38530→worker1:45845	Wywołanie metody: CoarseGrainedScheduler RetrieveSpakAppConfig.
53	20:42:56,950	worker2:38542→worker1:45845	Komunikacja Executora z Driverem, rejestracja Executora i wymiana metadanych.
54	20:42:58,567	worker1:41672→master:9000	Wywołanie metody: getFileInfo /user/hduser/ life_expectancy.csv z ClientProtocol.
55	20:42:58,584	worker1:41672→master:9000	Wywołanie metody: getFileInfo /user/hduser/ life_expectancy.csv z ClientProtocol.
56	20:42:58,628	worker1:41672→master:9000	Wywołanie metody: getListing /user/hduser/ life_expectancy.csv z ClientProtocol.
57	20:42:58,866	worker1:51332→master:8030	Wywołanie metody: allocate z ApplicationMasterProtocol.
58	20:43:01,872	worker1:51332→master:8030	Wywołanie metody: allocate z ApplicationMasterProtocol.
59	20:43:02,713	worker2:37734→worker1:36501	Spark komunikacja Executor z Driverem, przesyłanie Tasków do wykonania.
60	20:43:04,789	worker2:47120→master:9000	Wywołanie metody: getBlockLocations /user/ hduser/life_expectancy.csv z ClientProtocol.
61	20:43:04,881	worker2:47120→master:9000	Wywołanie metody: getServerDefaults z ClientProtocol.
62	20:43:04,877	worker1:51332→master:8030	Wywołanie metody: allocate z ApplicationMasterProtocol.
63	20:43:06,413	worker2:47120→master:9000	Wywołanie metody: getBlockLocations /user/ hduser/life_expectancy.csv z ClientProtocol.
64	20:43:07,545	worker1:41672→master:9000	Wywołanie metody: getFileInfo /user/hduser/ life_expectancy_result.csv z ClientProtocol.

Lp.	Czas	Kierunek	Opis
65	20:43:07,548	worker1:41672→master:9000	Wywołanie metody: getFileInfo /user/hduser/ life_expectancy_result.csv z ClientProtocol.
66	20:43:07,554	worker1:41672→master:9000	Wywołanie metody: delete /user/hduser/ life_expectancy_result.csv z ClientProtocol.
67	20:43:07,643	worker1:41672→master:9000	Wywołanie metody: mkdirs /user/hduser/ life_expectancy_result.csv/_temporary/0 z ClientProtocol.
68	20:43:07,886	worker1:51332→master:8030	Wywołanie metody: allocate z ApplicationMasterProtocol.
69	20:43:08,368	worker2:47120→master:9000	Wywołanie metody: create /user/hduser/ life_expectancy_result.csv/_temporary /0/_temporary/attempt_0/part_0
70	20:43:08,461	worker2:47120→master:9000	Wywołanie metody: addBlock /user/hduser/ life_expectancy_result.csv/_temporary /0/_temporary/attempt_0/part_0
71	20:43:08,499	worker2:33942→worker1:9866	Przesyłanie wyniku przetwarzania.
72	20:43:08,507	worker1:35218→master:9000	Wywołanie metody: blockReceivedAndDeleted z DatanodeProtocol.
73	20:43:08,508	worker2:51112→master: 9000	Wywołanie metody: blockReceivedAndDeleted z DatanodeProtocol.
74	20:43:08,510	worker2:47120→master:9000	Wywołanie metody: complete /user/hduser/ life_expectancy_result.csv/_temporary /0/_temporary/attempt_0/part_0 z ClientProtocol.
75	20:43:08,517	worker2:47120→master:9000	Wywołanie metody: getFileInfo /user/hduser/ life_expectancy_result.csv/_temporary /0/_temporary/attempt_0/part_0 z ClientProtocol.
76	20:43:08,523	worker2:47120→master:9000	Wywołanie metody: getFileInfo /user/hduser/ life_expectancy_result.csv/_temporary /0/_temporary/attempt_0 z ClientProtocol.
77	20:43:08,536	worker2:47120→master:9000	Wywołanie metody: getFileInfo /user/hduser/ life_expectancy_result.csv/_temporary /0/_temporary/attempt_0 z ClientProtocol.
78	20:43:08,540	worker2:47120→master:9000	Wywołanie metody: getFileInfo /user/hduser/ life_expectancy_result.csv/_temporary /0/task_0 z ClientProtocol.

Lp.	Czas	Kierunek	Opis
79	20:43:08,544	worker2:47120→master:9000	Wywołanie metody: rename /user/hduser/ life_expectancy_result.csv/_temporary /0/_temporary/attempt_0/user/hduser/ life_expectancy_result.csv/ _temporary/0/task_0 z ClientProtocol.
80	20:43:08,595	worker2:47120→master:9000	Wywołanie metody: getListing /user/hduser/ life_expectancy_result.csv/_temporary/0 z ClientProtocol.
81	20:43:08,597	worker1:41672→master:9000	Wywołanie metody: getFileInfo /user/hduser/ life_expectancy_result.csv z ClientProtocol.
82	20:43:08,599	worker1:41672→master:9000	Wywołanie metody: getListing /user/hduser/ life_expectancy_result.csv/temporary/0/task_0 z ClientProtocol.
83	20:43:08,601	worker1:41672→master:9000	Wywołanie metody: getFileInfo /user/hduser/ life_expectancy_result.csv/part_0 z ClientProtocol.
84	20:43:08,605	worker1:41672→master:9000	Wywołanie metody: rename /user/hduser/ life_expectancy_result.csv/_temporary /0 /user/hduser/life_expectancy_result.csv/ part_0 z ClientProtocol.
85	20:43:08,610	worker1:41672→master:9000	Wywołanie metody: delete /user/hduser/ life_expectancy_result.csv/_temporary z ClientProtocol.
86	20:43:08,621	worker1:41672→master:9000	Wywołanie metody: create /user/hduser/ life_expectancy_result.csv/_SUCCESS z ClientProtocol.
87	20:43:08,624	worker1:41672→master:9000	Wywołanie metody: complete/user/hduser/ life_expectancy_result.csv/_SUCCESS z ClientProtocol.
88	20:43:08,651	worker1:41672→master:9000	Wywołanie metody: delete /user/hduser/ life_expectancy_result.csv/.spark-staging z ClientProtocol.
89	20:43:08,691	worker1:41672→master:9000	Wywołanie metody: stopExecutor przez Drivera.
90	20:43:08,776	worker1:51332→master:8030	Wywołanie metody: finishApplicationMaster z ApplicationMasterProtocol.

Lp.	Czas	Kierunek	Opis
91	20:43:08,890	worker1:51332→master:8030	Wywołanie metody: finishApplicationMaster z ApplicationMasterProtocol.
92	20:43:08,893	worker1:41672→master:9000	Wywołanie metody: delete /user/hduser/ .sparkStaging/application_0 z ClientProtocol.
93	20:43:09,061	master:46454→worker1:35429	Wywołanie metody: stopContainers z ContainerManagementProtocol.

Opracowanie własne

W pakietach o numerach 1 i 2 zauważono działanie mechanizmu replikacji bloków. To rozwiązanie zwiększa szanse na przechwycenie danych przez atakującego, który ma dostęp tylko do ruchu sieciowego części węzłów. Pakiety od 3 do 6 to potwierdzenie otrzymania plików, które wysyłają Data Node'y do Name node'a. Kolejne pakiety do numeru 26 to powtórzenie tego procesu dla kolejnych przesyłanych plików. Między numerem 1 a 26 zaobserwowano przesyłanie bibliotek Spark, plików konfiguracyjnych oraz pliku whodata.jar, który zawiera kod źródłowy wykonywanego zadania. Pakiet 27 to instrukcja uruchomienia kontenera Yarn dla Drivera wykonywanego zadania. Wiersze od 28 do 34 dotyczą pobieranie informacji o blokach potrzebnych do wykonania zadania. Pakiet 35 to zarejestrowanie Application Mastera co oznacza, że Driver został prawidłowo uruchomiony. W wierszu 36 sprawdzane są informacje o lokalizacji pliku z konfiguracją, a między 37 a 39 alokowane są pierwsze potrzebne zasoby. Pakiet 40 to uruchomienie kontenera dla Executora. Pakiety 41-39 z wyjątkiem 44 i 45 to pobieranie informacji o danych wymaganych przez Executor. Pakiety 44, 45 i 50 to zapytania o alokowanie dodatkowych zasobów. Wiersze między 51 a 56 związane są z przygotowaniem Executora do pracy. Pakiety 57, 58 i 62 to puste zapytania allocate, które w tym przypadku wykorzystane są przez Application Mastera jako wiadomości heartbeat. Wiersze między 59 i 67 wygenerowane zostały przez Apache Spark i związane są z przetwarzaniem i przygotowaniem miejsca zapisu tymczasowych danych. Pakiet 68 to kolejne wykorzystanie allocate jako wiadomość heartbeat. Wiersze między 69 a 88 związane są z procesem zapisywania wyniku przez Apache Spark. To narzędzie zapisuje swój wynik w lokalizacji tymczasowej, która dopiero po zakończeniu przetwarzania jest kopiowana do właściwego katalogu. Spark tworzy też plik o nazwie _SUCCESS symbolizujący wykonanie zadania oraz usuwa wszystkie tymczasowe pliki. Wiersz 89 to zakończenie pracy Executora przez Drivera. Pakiet 90 i 91 to wyrejestrowanie wykonywanego zadania z Yarn Scheduler. Wiersz 92 to usuwanie plików tymczasowych

związanych z zadaniem. Pakiet 93 to zatrzymanie kontenera, w którym uruchomiony był Driver.

Z punktu widzenia bezpieczeństwa przechwycenie ruchu sieciowego daje atakującemu wiele możliwości. Sama analiza pakietów i używanych protokołów pozwala określić adresy węzłów, numery portów działających usług oraz wykorzystywane ścieżki HDFS. Możliwe jest określenie, na których węzłach roboczych przechowywane są bloki określonych plików. Z pakietów można wyczytać, jakie nazwy użytkowników są używane w obrębie klastra. Szczególnie cenne dla atakującego są pakiety 19 i 20, w których przesyłany jest plik jar zawierający kod źródłowy wykonywanego zadania. Odczytanie pakietu 71 pozwala poznać wyniki przetwarzania, co pokazano na rysunku 5.1. Można z niego odczytać wartości Developed,79.19785156625 i Developing,67.11146523178812 bezpośrednio po przekonwertowaniu wartości w systemie szesnastkowym na tablice znaków ASCII.

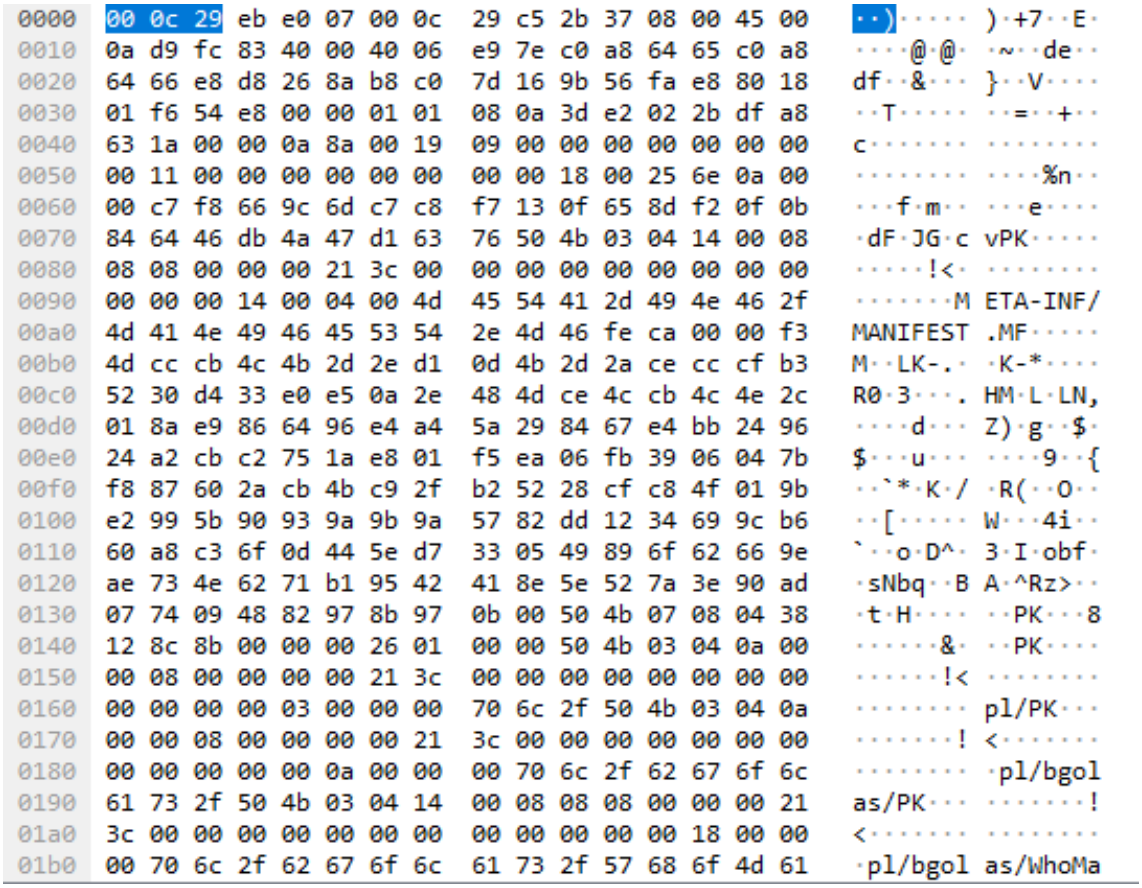
Rysunek 5.1 Pakiet zawierający wynik przedstawiony za pomocą tablicy ASCII

Opracowanie własne

Plik life _expectancy.csv, który zawiera dane wykorzystywane przez wykonywane zadanie, nie był przesyłany przez sieć. Powodem była dostępność bloków pliku na węzłach, gdzie uruchomione były Driver i Executor.

5.1 Wydobycie pliku jar z danych przesyłanych w sieci

Po przechwyceniu pakietów 19 i 20 można podjąć próbę wydobycia pliku jar. Na rysunku 5.2 przedstawiono pierwsze jego bajty w systemie szesnastkowym i przekonwertowane na tablice ASCII.



Rysunek 5.2 Pakiet zawierający przesyłany plik whodata.jar

Opracowanie własne

Każdy pakiet rozpoczyna się nagłówkami związanymi z wykorzystanymi protokołami. Aby wydobyć plik należy znaleźć bajt, który go rozpoczyna i usunąć zbędne nagłówki. Pliki jar posiadają podobną strukturę do plików zip, które rozpoczynają się sekwencją “50 4b”. Prezentacja tych bajtów w ASCII to “PK”. W celu wydobycia przesyłanego pliku jar, dane w formie szesnastkowej zostały skopiowane do edytora HxD. Następnie usunięto wszystkie bajty przed “50 4b” i zapisano otrzymany ciąg jako plik test.jar. Za pomocą programu jd-gui zdekompilowano ten plik, w wyniku czego udało się uzyskać kod źródłowy aplikacji, który pokazano na rysunku 5.3.

Otrzymany kod, zawiera pewne zniekształcenia wynikające z faktu, że użyty de-

kompilator próbował przedstawić kod napisany w Scali przy pomocy języka Java, jednak wynik nadal jest czytelny. Otrzymany plik jar nie jest uszkodzony i możliwe jest jego użycie. W podobny sposób można próbować przechwycić plik zip zawierający biblioteki Apache Spark, przesyłany w pakietach 1 i 2 oraz plik zip zawierający konfigurację z pakietów 7 i 8.

Przeprowadzone ataki potwierdzają pierwszą z postawionych hipotez. Domyślna konfiguracja nie zapewnia satysfakcjonującego poziomu bezpieczeństwa i podatna jest na wiele ataków.

```
package pl.bgolas;

import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.SaveMode;
import org.apache.spark.sql.SparkSession;
import scala.collection.Seq;

public final class WhoMain$ {
   public static WhoMain$ MODULE$;

   private final String appName;

   public String appName() {
      return this.appName;
   }

   public void main(String[] args) {
      SparkSession spark = org.apache.spark.sql.SparkSession$.MODULE$.builder().appName(appName()).getOrCreate();
      Dataset data = spark.read().option("delimiter", ",").option("header", true).csv("/user/hduser/life_expectancy.csv").repartition(4);
      data
         .groupBy("status", (Seq)scala.Predef$.MODULE$.wrapRefArray((Object[])new String[0]))
         .agg(org.apache.spark.sql.functions$.MODULE$.avg("life_expectancy").as("life_expectancy"), (Seq)scala.Predef$.MODULE$.wrapRefArray((Object[])new org.apache.spark.sql

         .mode(SaveMode.Overwrite)
         .csv("/user/hduser/life_expectancy_result.csv");
   }

   private WhoMain$() {
      MODULE$ = this;
      this.appName = "WhoDataApp";
   }
}
```

Rysunek 5.3 Odzyskany kod zródłowy whodata.jar

Opracowanie własne

6. Zabezpieczenie systemu Big Data

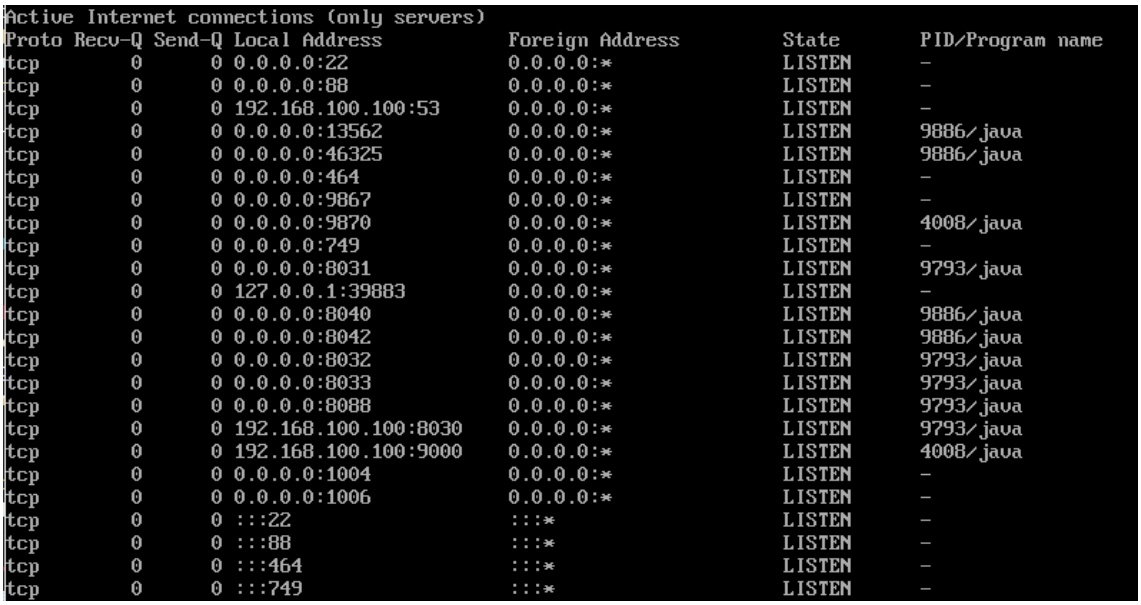
W tym rozdziale przedstawione zostaną efekty zabezpieczenia systemu opartego o Apache Hadoop i Apache Spark za pomocą protokołu uwierzytelniającego Kerberos oraz szyfrowania.

6.1 Uwierzytelnianie za pomocą Kerberos

Na węźle głównym zainstalowano serwer Kerberos. Na pozostałych węzłach zainstalowano aplikacje klienckie. Utworzono 4 tożsamości: hdfs, yarn, mapred i HTTP i na wszystkich węzłach stworzono odpowiadających im użytkowników, oraz uruchomiono uwierzytelniania za pomocą Kerberos. Zgodnie z tabelą 3.1 poszczególne usługi uruchomiono za pomocą zalecanych użytkowników. Dodatkowo Data Node'y skonfigurowano w taki sposób, aby korzystały z portów uprzywilejowanych. W związku z zastosowaniem przez Kerberosa DNSa zmienione zostały nazwy hostów, które umieszczono w domenie bgolas.pl. Po zmianie ich pełne nazwy to: master.bgolas.pl, worker1.bgolas.pl, worker2.bgolas.pl. Aby zapewnić działanie tej lokalnej domeny, na węźle głównym zainstalowano serwer DNS oparty o aplikację NSD. Zawiera on konfigurację domeny i używany jest przez wszystkie węzły. Porty otwarte na węźle głównym zaprezentowano na rysunku 6.1, a porty otwarte na węźle roboczym na rysunku 6.2.

Porty takie jak 88, 464, 749 związane są z działaniem Kerberosa. Port 53 to serwer DNS. Na węźle roboczym można zaobserwować użycie portu 1004 zamiast 9866 i 1006 zamiast 9864. W kolumnie "PID/Program name" w przypadku niektórych procesów widnieje jsvc.exec zamiast java. JSVC używany jest do bezpiecznego uruchamianiu niektórych procesów platformy hadoop. Te procesy wymagają użycia użytkownika root, który po wstępnej konfiguracji korzysta z JSVC, aby zmienić właściciela tych procesów na innego użytkownika. Skorzystanie z roota wymagane jest do m.in. użycia portów uprzywilejowanych.

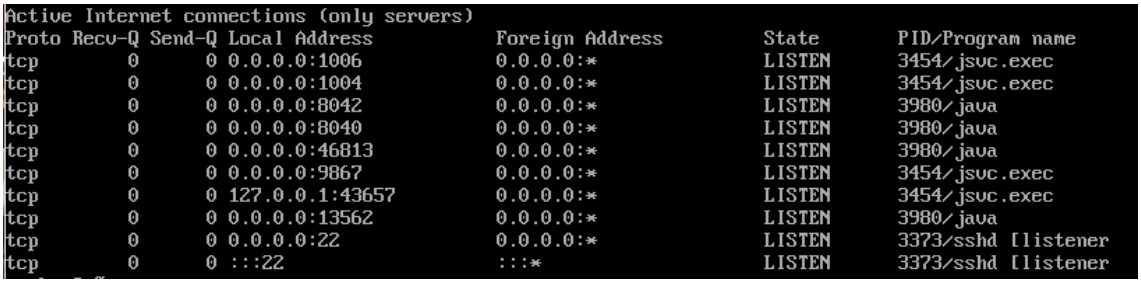
Na tak przygotowanym klastrze przeprowadzono ataki prezentowane w poprzednich rozdziałach.



```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:88              0.0.0.0:*               LISTEN      -
tcp        0      0 192.168.100.100:53      0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:13562           0.0.0.0:*               LISTEN      9886/java
tcp        0      0 0.0.0.0:46325           0.0.0.0:*               LISTEN      9886/java
tcp        0      0 0.0.0.0:464             0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:9867            0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:9870            0.0.0.0:*               LISTEN      4008/java
tcp        0      0 0.0.0.0:749             0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:8031            0.0.0.0:*               LISTEN      9793/java
tcp        0      0 127.0.0.1:39883         0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:8040            0.0.0.0:*               LISTEN      9886/java
tcp        0      0 0.0.0.0:8042            0.0.0.0:*               LISTEN      9886/java
tcp        0      0 0.0.0.0:8032            0.0.0.0:*               LISTEN      9793/java
tcp        0      0 0.0.0.0:8033            0.0.0.0:*               LISTEN      9793/java
tcp        0      0 0.0.0.0:8088            0.0.0.0:*               LISTEN      9793/java
tcp        0      0 192.168.100.100:8030    0.0.0.0:*               LISTEN      9793/java
tcp        0      0 192.168.100.100:9000    0.0.0.0:*               LISTEN      4008/java
tcp        0      0 0.0.0.0:1004            0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:1006            0.0.0.0:*               LISTEN      -
tcp        0      0 :::22                   :::*                    LISTEN      -
tcp        0      0 :::88                   :::*                    LISTEN      -
tcp        0      0 :::464                  :::*                    LISTEN      -
tcp        0      0 :::749                  :::*                    LISTEN      -
```

Rysunek 6.1 Porty otwarte na węźle głównym po zainstalowaniu Kerberosa

Opracowanie własne za pomocą komendy netstat -ltnp



```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:1006            0.0.0.0:*               LISTEN      3454/jsvc.exec
tcp        0      0 0.0.0.0:1004            0.0.0.0:*               LISTEN      3454/jsvc.exec
tcp        0      0 0.0.0.0:8042            0.0.0.0:*               LISTEN      3980/java
tcp        0      0 0.0.0.0:8040            0.0.0.0:*               LISTEN      3980/java
tcp        0      0 0.0.0.0:46813           0.0.0.0:*               LISTEN      3980/java
tcp        0      0 0.0.0.0:9867            0.0.0.0:*               LISTEN      3454/jsvc.exec
tcp        0      0 127.0.0.1:43657         0.0.0.0:*               LISTEN      3454/jsvc.exec
tcp        0      0 0.0.0.0:13562           0.0.0.0:*               LISTEN      3980/java
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      3373/sshd [listener
tcp        0      0 :::22                   :::*                    LISTEN      3373/sshd [listener
```

Rysunek 6.2 Porty otwarte na węźle roboczym po zainstalowaniu Kerberosa

Opracowanie własne za pomocą komendy netstat -ltnp

6.2 Atak na protokół RPC i HDFS

Zastosowaną wcześniej aplikację, która została stworzona do atakowania protokołu RPC, zmodyfikowano w taki sposób, aby dopasować ją do nowej konfiguracji klastra. Próba uruchomienia aplikacji powoduje wywołanie wyjątku:

Exception in thread "main"java.lang.RuntimeException: org.apache.hadoop.ipc.RemoteException(org.apache.hadoop.security.AccessControlException): SIMPLE authentication is not enabled. Available:[TOKEN, KERBEROS].

Oznacza on, że Data Node odrzucił próbę zalogowania za pomocą "prostego uwierzytelniania". Jedyne dostępne opcje to użycie tokena lub protokołu Kerberos. Atakujący musiałby zdobyć dane używane w jednej z tych form uwierzytelniania, aby wykonać atak.

W przypadku próby ataku na hdfs wykonanie zostaje przerwane identycznie wyglądającym wyjątkiem. Wszystkie kanały dostępu opierające się o protokół RPC zostały zabezpieczone i wymagają logowania za pomocą tokenu lub Kerberos.

6.3 Analiza ruchu sieciowego

Samo zastosowanie uwierzytelniania nie zabezpiecza platformy przed podsłuchiwaniem ruchu sieciowego. Wśród przechwyconych pakietów znaleziono taki, który zawierał kod źródłowy aplikacji whodata.jar. Jest on podobny do tego przechwyconego w czasie działania niezabezpieczonego klastra. Nadal możliwe jest odzyskanie pliku jar i jego zdekompilowanie w celu poznania zasady działania wykonywanej aplikacji. Dodatkowo wśród pozostałych przechwyconych pakietów można znaleźć te, w których przesyłane są przetwarzane dane. W czasie tego uruchomienia aplikacji, proces Executora uruchomiony został na węźle master.bgolas.pl. Nie są na nim przechowywane żadne bloki systemu plików HDFS, więc konieczne było ich przesłanie z jednego z węzłów roboczych. Dzięki temu atakujący mógłby poznać, jakie dane były wymagane do przeprowadzenia przetwarzania. Fakt, że dane te przechowywane były jako plik csv, który jest formatem tekstowym, umożliwia ich odczytanie przez zaprezentowanie przesyłanych bajtów za pomocą tablicy ASCII. Pokazano to na rysunku 6.3.

Rysunek 6.3 Pakiet zawierający przetwarzane dane, przedstawiony za pomocą tablicy ASCII

Opracowanie własne

W trakcie analizy wykorzystywanych metod uwierzytelniania zauważono, że protokół Kerberos wykorzystywany jest tylko w wybranych przypadkach. W wielu zapytaniach stosowany jest Hadoop Delegation Token, którego implementacja korzysta z SASL opartego o DIGEST-MD5. Zastosowanie tej technologii utrudnia przechwycenie przesyłanego tokena, ale została ona uznana w RFC 2831 za niewystarczająco bezpieczną. Przykład jej zastosowania do celu uwierzytelniania przedstawiono na rysunku 6.4. Jest to rozpoczęcie komunikacji wykorzystującej ClientProtocol i port numer 9000 węzła głównego. Pierwsze 4 pakiety związane są z przeprowadzeniem uwierzytelnienia. Można odczytać z nich takie szczegóły jak: używane algorytmy, wartości dla nonce, username w Base64, realm, cnonce, numer odpowiedzi czy ustawienia qop. Te dane nie są wystarczające do wygenerowania prawidłowej odpowiedzi, a numer odpowiedzi uniemożliwia wykorzystanie przesyłanego pakietu ponownie, więc zdobycie tych nie zagraża bezpieczeństwu systemu bezpośrednio, jednak mogą one być wykorzystane w próbach złamania szyfrowania. Pozostała część komunikacji nie jest zaszyfrowana i jest możliwe jej odczytanie.

Rysunek 6.4 Proces uwierzytelniania z wykorzystaniem DIGEST-MD5

Opracowanie własne

6.4 Szyfrowanie danych w transporcie

Po uruchomieniu szyfrowania danych w transporcie, wszystkie dane, zarówno te przesyłane za pomocą protokołu RPC jak i przez Data Node'y za pomocą TCP/IP, są w pełni zaszyfrowane. Jedynym fragmentem, który można częściowo odczytać, są pierwsze pakiety, które negocjują sposób szyfrowania. Zaprezentowano to na rysunku 6.5. Do negocjacji szyfru nadal używany jest algorytm md5-sess. Dalsza komunikacja szyfrowana jest przy pomocy 3DES. Po wprowadzeniu tych zmian nie jest możliwe odzyskanie przesyłanych danych w prosty sposób. Wymagałoby to złamania szyfrowania stosowanego

Rysunek 6.5 Pakiet zaszyfrowany za pomocą 3DES

Opracowanie własne

do negocjowania parametrów połączenia lub algorytmu zastosowanego do szyfrowania przesyłanych danych. Nawet w przypadku użycia protokołów opartych o RPC nie można określić jakie implementacje i które metody są wywoływane przez poszczególnych użytkowników. W tym przypadku atakujący nie jest w stanie określić, co jest przesyłane przez sieć i jakie zadania wykonywane są na klastrze.

Powyższe wyniki potwierdzają drugą z postawionych hipotez. Aby osiągnąć wysoki poziom bezpieczeństwa, kluczowe jest zastosowanie właściwej konfiguracji, bezpiecznego uwierzytelniania i autoryzacji. W tym celu konieczne było wykorzystanie algorytmów szyfrujących opartych o kryptografie.

Podsumowanie

W pracy osiągnięto cel, jakim było zbadanie wpływu zastosowanej konfiguracji na poziom zabezpieczenia systemu Big Data. Ataki przeprowadzone w trakcie badań potwierdziły pierwszą hipotezę, że domyślna konfiguracja zapewnia niski poziom bezpieczeństwa i podatna jest na wiele ataków. Przeprowadzenie analizy ruchu sieciowego pozwoliło potwierdzić, że możliwe jest przechwycenie przetwarzanych danych, kodu źródłowego wykonywanego zadania, a także jego wyniku. Aby zapobiec podsłuchiwaniu ruchu sieciowego oraz przeprowadzonym atakom, konieczne było skonfigurowanie uwierzytelniania za pomocą protokołu Kerberos oraz wykorzystanie szyfrowania w trakcie przetwarzania. Po zmodyfikowaniu konfiguracji systemu osiągnięto satysfakcjonujący poziom bezpieczeństwa. Potwierdziło to drugą z postawionych hipotez, co oznacza, że wszystkie postawione hipotezy zostały potwierdzone.

Osiągniete wyniki oznaczają, że choć zarówno Apache Hadoop jak i Apache Spark umożliwiją zminimalizowanie ryzyk i osiągnięcie satysfakcjonującego poziom bezpieczeństwo to oba te rozwiązania ciężar zapewnienia właściwej konfiguracji zrzucają na użytkownika końcowego. Klaster obliczeniowy wykorzystujący te technologie jest podatny na ataki, które są proste w przygotowaniu dla atakującego posiadającego wiedzę z zakresu cyberbezpieczeństwa i programowania. Jest to duże ryzyko, ponieważ takie systemy są zwykle wykorzystywane przez użytkowników, którzy mają doświadczenie związane z tworzeniem oprogramowania. Dodatkowo wykorzystanie protokołu Kerberos utrudnia proces zabezpieczenia takiego systemu. Ten sposób uwierzytelniania nie jest już powszechnie wykorzystywany, a jego zastosowanie wymaga zainstalowania w sieci aplikacji pełniącej rolę serwera oraz instalacje aplikacji klienckich na wszystkich węzłach, oraz komputerach użytkowników. Konfiguracja tych aplikacji jest procesem czasochłonnym i skomplikowanym. Szczególnie rozczarowujący jest brak jakiegokolwiek szyfrowania przy domyślnych ustawieniach i fakt, że nawet gdy już zostanie ono uruchomione to w niektórych zastosowaniach wykorzystywane są słabe algorytmy szyfrujące jak MD5.

Osiągnięte w pracy wyniki pozwalają lepiej zrozumieć ryzyka związane z niewłaściwą konfiguracją systemu opartego o Apache Hadoop i Apache Spark. Wnioski mogą być wykorzystane w praktyce przez administratorów systemów Big Data przy projektowaniu

nowych instalacji i zabezpieczaniu istniejących. W trakcie badań zauważono wykorzystanie implementacji SASL opartej o DIGEST-MD5, więc wskazane byłoby przeprowadzenie analizy wpływu jej wykorzystania na poziom zabezpieczenia takich systemów.

Bibliografia

[1] Aws I. Abueid. „Big Data and Cloud Computing Opportunities and Application Areas". W: *Engineering, Technology & Applied Science Research* 14.3 (2024). DOI: https://doi.org/10.48084/etasr.

[2] *Apache Hadoop Documentation*. URL: https://hadoop.apache.org/docs/stable/index.html. (dostęp: 16.11.2024).

[3] *Apache Spark*. URL: https://spark.apache.org/. (dostęp: 28.09.2024).

[4] *Apache Spark Documentation*. URL: https://spark.apache.org/documentation.html. (dostęp: 16.11.2024).

[5] Yusuf Aytas. *Designing Big Data Platforms*. O'Reilly Media, 2021. ISBN: 9781119690924.

[6] Matei Zaharia Bill Chambers. *Spark: The Definitive Guide. Big Data Processing Made Simple*. O'Reilly Media, 2018. ISBN: 9781491912294.

[7] Bradley TH. Borky JM. „Protecting Information with Cybersecurity." W: *Effective Model-Based Systems Engineering* (wrz. 2018), s. 345–404. DOI: 10.1007/978-3-319-95669-5_10.

[8] Michael Cox i David Ellsworth. „Managing big data for scientific visualization". W: (sty. 1997).

[9] Jeffrey Dean i Sanjay Ghemawat. „Mapreduce: simplified data processing on large clusters". W: *Communications of the ACM* 51 (sty. 2004), s. 107–113. ISSN: 0001-0782. DOI: 10.1145/1327452.1327492.

[10] Jeff Desjardinsh. *How much data is generated each day?* URL: https://www.weforum.org/agenda/2019/04/how-much-data-is-generated-each-day-cf4bddf29f/. (dostęp: 06.10.2024).

[11] *Hadoop Http Authentication*. URL: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/HttpAuthentication.html. (dostęp: 07.12.2024).

[12] *Hadoop in Secure Mode*. URL: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SecureMode.html. (dostęp: 07.12.2024).

[13] Ibrahim Hashem i in. „The rise of “Big Data” on cloud computing: Review and open research issues”. W: *Information Systems* 47 (lip. 2014), s. 98–115. DOI: 10.1016/j.is.2014.07.006.

[14] *HDFS Permission Guide*. URL: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsPermissionsGuide.html. (dostęp: 07.12.2024).

[15] Ishwarappa i J. Anuradha. „A Brief Introduction on Big Data 5Vs Characteristics and Hadoop Technology”. W: *Procedia Computer Science* 48 (2015). International Conference on Computer, Communication and Convergence (ICCC 2015), s. 319–324. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2015.04.188.

[16] Martin Kleppmann. *Designing Data-Intensive Applications. The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. O’Reilly Media, 2017. ISBN: 9781449373320.

[17] G. Markowsky i L. Markowsky. „Survey of Supercomputer Cluster Security Issues”. W: *Proceedings of the 2007 International Conference on Security and Management* (2007), s. 474–480.

[18] *OWASP Mobile Application Security*. URL: https://owasp.org/www-project-mobile-app-security/. (dostęp: 16.11.2024).

[19] *OWASP Top 10 - 2021*. URL: https://owasp.org/Top10/. (dostęp: 16.10.2024).

[20] *OWASP Web Security Testing Guide*. URL: https://owasp.org/www-project-web-security-testing-guide/. (dostęp: 16.11.2024).

[21] *Parquet Files Columnar Encryption - Spark Documentation*. URL: https://spark.apache.org/docs/3.5.1/sql-data-sources-parquet.html#columnar-encryption. (dostęp: 07.11.2024).

[22] Makan Pourzandi i in. „Clusters and security: distributed security for distributed systems”. W: czer. 2005, 96–104 Vol. 1. ISBN: 0-7803-9074-1. DOI: 10.1109/CCGRID.2005.1558540.

[23] *Powered by Apache Hadoop*. URL: https://cwiki.apache.org/confluence/display/hadoop2/PoweredBy. (dostęp: 29.09.2024).

[24] Zaher Ali Al-Sai i in. „Explore Big Data Analytics Applications and Opportunities: A Review”. W: *Big Data and Cognitive Computing* 6.4 (2022). ISSN: 2504-2289. DOI: 10.3390/bdcc6040157.

[25] *Security - Spark Documentation*. URL: https://spark.apache.org/docs/3.5.2/security.html. (dostęp: 07.11.2024).

[26] Andrew Sellers. *Everything You Wanted To Know About Homomorphic Encryption (But Were Afraid To Ask)*. URL: https://www.forbes.com/councils/forbestechcouncil/2021/04/14/everything-you-wanted-to-know-about-homomorphic-encryption-but-were-afraid-to-ask/. (dostęp: 06.10.2024).

[27] Daniel Takabi, James Joshi i Gail-Joon Ahn. „Security and Privacy Challenges in Cloud Computing Environments”. W: *Security & Privacy, IEEE* 8 (sty. 2011), s. 24–31. DOI: 10.1109/MSP.2010.186.

[28] *Transparent Encryption in HDFS*. URL: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/TransparentEncryption.html. (dostęp: 07.12.2024).

[29] Tom White. *Hadoop: The Definitive Guide, 4th Edition*. O'Reilly Media, 2015. ISBN: 9781491901632.

[30] *YARN Secure Containers*. URL: https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/SecureContainer.html. (dostęp: 07.12.2024).

Spis rysunków

Spis tabel