

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

Wydział Inżynierii Metali i Informatyki Przemysłowej



PROJEKT INŻYNIERSKI

pt.

„Immunologiczny system detekcji
zagrożeń w oparciu o dane z logów
aktywności”

| | |
|-----------------------------|---|
| Imię i nazwisko dyplomanta: | Adam Kopeć |
| Kierunek studiów: | Informatyka Stosowana |
| Profil dyplomowania: | Modelowanie i Technologie Informacyjne |
| Nr albumu: | 223132 |
| Opiekun: | dr inż. Gabriel Rojek |

Podpis dyplomanta:

Podpis opiekuna:

Kraków 2012

Oświadczam, świadomy(-a) odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszy projekt inżynierski wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Kraków, dnia

Podpis dyplomanta.....

Spis treści

| | |
|---|----|
| 1. Wprowadzenie..... | 4 |
| 2. Sztuczne systemy immunologiczne..... | 6 |
| 2.1 Krótka charakterystyka mechanizmów immunologicznych człowieka..... | 6 |
| 2.2 Algorytm selekcji klonalnej..... | 8 |
| 2.3 Algorytm negatywnej selekcji..... | 9 |
| 2.4 Pozostałe systemy odpornościowe..... | 10 |
| 3. Wykorzystanie informacji z logów aktywności serwera Apache..... | 10 |
| 3.1 Algorytm negatywnej selekcji jako narzędzie do analizy logów..... | 10 |
| 3.2 Charakterystyka logów aktywności serwera Apache..... | 11 |
| 3.3 Możliwość wykorzystania algorytmu negatywnej selekcji w zakresie analizy logów..... | 12 |
| 4. Projekt systemu bezpieczeństwa bazujący na algorytmie negatywnej selekcji..... | 13 |
| 4.1 Wymagania funkcjonalne..... | 13 |
| 4.2 Wymagania techniczne..... | 14 |
| 4.3 Analiza wymagań..... | 14 |
| 5. Implementacja systemu..... | 15 |
| 5.1 Diagramy wybranych algorytmów programu..... | 15 |
| 5.2 Interfejs programu..... | 17 |
| 5.3 Opis kompetencji i diagramy klas..... | 17 |
| 6. Badania skuteczności programu..... | 23 |
| 6.1 Pochodzenie i organizacja danych testowych..... | 23 |
| 6.2 Charakterystyka ataku na serwis testowy..... | 24 |
| 6.3 Analiza logów przy użyciu przygotowanego programu, z wykorzystaniem algorytmu negatywnej selekcji..... | 26 |
| 6.3.1 Dane kontrolne..... | 26 |
| 6.3.2 Metodologia badań i organizacja wyników..... | 26 |
| 6.3.3 Problemy z danymi zapisanymi jako Unicode..... | 27 |
| 6.3.4 Raporty z analizy dla długości słowa $l=8$ i stałej dopasowania $r=2$ | 28 |
| 6.3.5 Raporty z analizy dla długości słowa $l=10$ i stałej dopasowania $r=4$ | 29 |
| 6.3.6 Analiza otrzymanych wyników..... | 31 |
| 7. Podsumowanie i wnioski..... | 34 |
| 8. Spis tabel..... | 35 |
| 9. Spis rysunków..... | 35 |
| 10. Bibliografia..... | 36 |
| A. Dodatek 1: Zawartość płyty CD..... | 37 |

1. Wprowadzenie

Zagadnienie bezpieczeństwa systemów informatycznych to jeden z ważniejszych problemów informatyki. Urządzenia i oprogramowanie, które wykorzystywane są w wielu różnych dziedzinach życia nie tylko powinny bezbłędnie działać, ale również zapewniać ochronę danych i innych zasobów, z którymi pracują.

Problematyka zabezpieczeń w dużej mierze obejmuje zagadnienie zapobiegania nieuprawnionemu dostępowi. Szczególnie istotne stało się ono w dobie Internetu, kiedy to okazało się, że potencjalny agresor wcale nie musi mieć fizycznego kontaktu z systemem, któremu chce wyrządzić szkody; wystarczy, że połączy się z nim w sposób zdalny.

Przed oprogramowaniem i usługami, które udostępniane są w Internecie, stanęło podwójnie skomplikowane zadanie. Użytkownicy zaczęli się domagać znacznego zwiększenia dynamiki ich rozwoju. Ciągłe utrzymywanie wysokiego jego tempa stało się powodem marginalizacji niegdyś dominującej metody tworzenia aplikacji, polegającej na tworzeniu interfejsu i wszystkich komponentów od początku do końca przez jeden zespół deweloperski. Na popularności zyskało natomiast programowanie z wykorzystaniem wielu obecnych na rynku podsystemów, które jedynie integruje się w celu stworzenia produktu końcowego. Podejście takie ma, oczywiście, wiele zalet, niemniej posiada ono również istotną wadę, której nie sposób zbagatelizować: każda luka w zabezpieczeniach jednego modułu automatycznie zostaje powielona we wszystkich systemach, które go wykorzystują. Wszyscy Ci, którym zależy na wyrządzaniu szkód w serwisach, czy to z powodów osobistych, czy finansowych, szybko zdali sobie z tego sprawę. Informacje o potencjalnych drogach ataku również zaczęły rozprzestrzeniać się błyskawicznie. Utrzymywanie systemu w dobrej kondycji wymaga zatem odtąd ciągłego monitorowania jego stanu, dbania o właściwą częstotliwość wykonywania aktualizacji oraz stałego powiększania swojej wiedzy w zakresie metod ataku. Tylko w ten sposób bowiem można utrzymać swoją pozycję o jeden mały krok przed agresorem.

Rozważania na temat bezpieczeństwa systemów internetowych mogą przynieść skojarzenie, że sytuacja takich systemów jest bardzo podobna do tego, z czym ma do czynienia organizm każdego z nas. Utrzymanie go w dobrej formie również wymaga ciągłych zabiegów, a ewentualne zaniedbania mogą mieć bardzo przykre konsekwencje, od obniżenia jego sprawności aż po poważne choroby, które mogą doprowadzić do śmierci.

Celem niniejszej pracy jest znalezienie takiej metody zabezpieczania systemów, która będzie pewnym analogiem czynności, które organizm człowieka wykonuje samoczynnie, gdy broni się przed chorobami. Obszar poszukiwań zostanie zawężony do serwisów i usług internetowych działających pod kontrolą najpopularniejszego obecnie¹ serwera protokołu HTTP: Apache HTTP Server.

W drugim rozdziale opisane zostaną sztuczne systemy immunologiczne, wraz z odniesieniami do układu immunologicznego człowieka. Omówione zostaną wykorzystywane przez nie mechanizmy, oraz przedstawione zostaną niektóre metody ich zastosowania. Bardziej wnikliwie przeanalizowany zostanie algorytm tzw. negatywnej selekcji.

Problematyka analizy logów serwera Apache pod kątem wykorzystania w niej ww. algorytmu będzie przedmiotem rozważań rozdziału trzeciego. Poruszone zostaną takie kwestie, jak zwyczajowy wygląd tego rodzaju plików oraz charakterystyka danych, które są w nich zawarte. Ponadto udzielona zostanie odpowiedź na pytanie, czy algorytm w ogóle może zostać użyty do ich analizy.

Czwarty oraz piąty rozdział zajmą się kwestiami bardziej technicznymi. Znajdzie się w nich spis wymagań technicznych i funkcjonalnych programu, który stworzony zostanie jako narzędzie analizy logów. Umieszczone zostaną tam również szczegóły jego implementacji, przyjęte założenia i uproszczenia, a także opis jego interfejsu.

Część pierwsza rozdziału szóstego poświęcona zostanie analizie rzeczywistego ataku, jaki miał miejsce na jeden z serwisów internetowych. Następnie logi z tego ataku zostaną poddane obróbce z użyciem stworzonego wcześniej programu. Wyniki tej obróbki zostaną skonfrontowane z ręczną analizą i wraz z komentarzami zostaną umieszczone w drugiej części rozdziału.

Rozdział siódmy poświęcony będzie podsumowaniu osiągniętych wyników. Zaproponowane w nim zostaną możliwe kierunki rozwoju zastosowanego rozwiązania. Przedyskutowany również zostanie procentowy udział błędnych wyników wraz z możliwościami jego minimalizacji.

¹ Prawie 65% udziału w rynku serwerów WWW według badań firmy Netcraft, patrz: <http://news.netcraft.com/archives/2012/01/03/january-2012-web-server-survey.html> [dostęp 08.01.2012r.]

2. Sztuczne systemy immunologiczne

Aby analizować możliwości odwzorowania w informatyce procesów zachodzących w przyrodzie, należy najpierw krótko je scharakteryzować. Pozwoli to bowiem na wyjaśnienie nomenklatury używanej w dalszej części niniejszej pracy, a także przyjętych w niej uproszczeń.

2.1 Krótka charakterystyka mechanizmów immunologicznych człowieka

System odpornościowy człowieka jest bardzo złożonym mechanizmem, zdolnym radzić sobie z zagrożeniami na wiele sposobów. W jego skład wchodzi wiele różnych warstw ochronnych, począwszy od prostych barier fizycznych, takich jak np. skóra, po wyrafinowane zabezpieczenia na poziomie komórkowym.

Istotną cechą tego systemu jest zdolność odróżniania komórek własnych od obcych (ang. *self-nonsel self discrimination*). Należy zauważyć, że znana jest jedynie charakterystyka komórek własnego organizmu. Liczba i rodzaje potencjalnych agresorów nie są dane *a priori*. Do ich rozpoznawania musiała zatem wyewoluować metoda, która zadziała zarówno dla poznanych wcześniej, jak i dla pierwszy raz ujranych zagrożeń. Skalę problemu dodatkowo zwiększa fakt, że komórki znajdujące się zwykle w ciele człowieka oraz te, z którymi mierzyć się musi jego system odpornościowy, zbudowane są z takich samych elementów, czyli z aminokwasów. Szacuje się jednak, że choć układ odpornościowy jest „zaprogramowany” z użyciem 10^5 genów, jest on w stanie z dobrym skutkiem odróżniać ok. 10^5 swoich komórek od nawet 10^{16} możliwych komórek obcych [3].

Kluczową rolę dla zapewnienia takiej skuteczności odgrywa fakt, że cały układ posiada zdolność uczenia się. W jego obrębie funkcjonuje wiele komórek, zwanych antygenami, które są w stanie wywołać jakiś rodzaj reakcji immunologicznej. Tylko pewien ich podzbiór, zwany patogenami, wykazuje cechy chorobotwórcze. Cała reszta antygenów krąży w całym organizmie lub tylko niektórych jego organach, utrzymując system odpornościowy w gotowości do walki z patogenami.

Na poziomie komórkowym układ immunologiczny posiada dwa systemy obrony. Pierwszym z nich jest system nieswoisty, czyli wrodzony. Przez całe życie człowieka pozostaje on w niezmienionej formie. Umożliwia jednak szybkie reagowanie na pojawienie się obcych komórek i zapewnia podstawową ochronę do czasu, gdy system swoisty będzie w stanie poradzić sobie z zagrożeniem. System swoisty, inaczej adaptacyjny, nie tylko posiada zdolność do ciągłego przystosowywania się do nowych warunków mikrobiologicznych, ale

posiada również tzw. pamięć immunologiczną, która pozwala szybciej reagować na te antygeny, które już wcześniej pojawiły się w organizmie. Produkcja przeciwciał w takim przypadku przebiega dużo szybciej i jest bardziej efektywna.

Bezpośrednimi uczestnikami procesów odpornościowych systemu swoistego są komórki zwane limfocytami, które przemieszczają się po całym organizmie dzięki płynom ustrojowym, m.in. limfie. Sprawia to, że rozłożone są dość równomiernie w całej jego objętości, zapewniając dzięki temu zbliżony poziom ochrony w każdym jego punkcie. Istnieje wiele rodzajów limfocytów, mających wpływ na różne mechanizmy systemu. Najważniejszymi z nich są jednak limfocyty T (od łac. *thymus*, czyli grasicy), oraz limfocyty B (od łac. *bursa Fabricii*, czyli kaletki Fabrycjusza, narządu występującego u ptaków, strukturalnie do grasicy podobnego). Nie różnią się one zbytnio budową wewnętrzną, natomiast różna jest ich rola oraz miejsce dojrzewania – grasicą, w przypadku limfocytów T oraz szpik kostny w przypadku limfocytów B.

Aby zrozumieć, jaką rolę odgrywają limfocyty w szeregu procesów obronnych, należy wcześniej pochylić się nad ich metodą rozpoznawania antygenów. Każda z tych komórek (a także same antygeny) otoczona jest specjalną, trójwymiarową strukturą proteinową. W przypadku antygenów struktury te są częścią ich powierzchni i nazywane są epitopami. Każdy limfocyt B z kolei otaczają receptory (inaczej przeciwciała), z których każdy posiada cztery łańcuchy aminokwasów. Każdy z łańcuchów zawiera pewną strukturę, której zadaniem jest dopasowywanie się do epitopów antygeny. Strukturę taką nazywa się paratopem. Ponadto każde przeciwciało posiada również własne epitopy. Każdy limfocyt zawiera tysiące receptorów, z których niektóre są takie same. Występowanie wielu takich samych przeciwciał pozwala na dokładną ocenę, czy limfocyt pasuje do antygeny, czy nie. Im więcej bowiem związanych epitopów, tym lepszy stopień dopasowania. Reakcja immunologiczna zaczyna się od tego, że po przekroczeniu granicznego progu dowiązań limfocyty B, dokonuje on swego rodzaju odwzorowania struktury antygeny na swojej powierzchni. Limfocyty T, które mają zdolność wiązania się z tak odwzorowaną strukturą, przyłączają się wtedy do niego, powodując pełną aktywację komórki. W dalszej kolejności stosowane są inne mechanizmy odpornościowe, z których szczególnie interesujące wydają się dwa: selekcja klonalna i negatywna. Zostaną one opisane dalej, przy okazji analizy opartych na nich algorytmów.

Swoisty system odpornościowy z racji swojej wysokiej skuteczności od dawna fascynuje informatyków. Uproszczone modele jego mechanizmów próbuje się zastosować w różnych dziedzinach obliczeń, również niezwiązanych z bezpieczeństwem, np. przy próbach rozwiązania problemu kolorowania grafów². Wykorzystują je również programy antywirusowe, anty szpiegowskie i inne.

2.2 Algorytm selekcji klonalnej

Aby zrozumieć ideę stojącą za powstaniem algorytmu selekcji klonalnej, należy odwołać się do tego, co dzieje się z limfocytami B po ich aktywacji. Gdy taka komórka zostanie aktywowana, system odpornościowy zaczyna dostosowywać się do nowej sytuacji. Przeciwciała, które wykryły antygen, zostają wyodrębnione i rozmnożone. Aktywne limfocyty B zaczynają się dzielić i produkować swoje wierne kopie. Kopie te ulegają następnie mutacjom, tak by jeszcze lepiej dopasowywać się do antygeny. Proces ten przebiega szybciej niż mutacja genetyczna i zwany jest proliferacją. W dalszej fazie kopie podlegają ocenie, tak by zostawić tylko te, które zapewniają najskuteczniejsze dopasowanie. Skopiowane komórki, które pozytywnie przejdą proces oceny mogą zostać następnie przekształcone na dwa sposoby w procesie dyferencjacji. Część z nich utworzy komórki pamięciowe, przydatne w późniejszym wykrywaniu takich samych antygenów, a z reszty powstaną komórki plazmatyczne, które aktywnie zajmą się eliminacją zagrożenia.

Algorytm selekcji klonalnej w ogólności rozpoczyna się od generacji pewnego zbioru komórek kandydujących i połączenia go z bieżącą populacją. W zależności od potrzeb można również uzupełnić go o jakąś formę reprezentacji komórek pamięciowych. Następnie wybiera się n komórek, które mają najwyższy stopień dopasowania i klonuje się je, zazwyczaj w ilości proporcjonalnej do jakości dopasowania. Kolejną fazą jest mutacja nowo powstałych komórek z zastrzeżeniem, że największym zmianom ulegają te komórki, które miały najniższy stopień dopasowania. Jeżeli korzysta się z komórek pamięciowych, to w tym momencie tworzy się ich zbiór spośród dysponujących najlepszym stopniem dopasowania komórek. Na samym końcu usuwa się ze zbioru pewną ilość komórek i zastępuje je nowo wygenerowanymi klonami. Zachowuje się przy tym prawidłowość, że im wyższy stopień dopasowania, tym niższe prawdopodobieństwo zastąpienia.

2 Dąbrowski Jacek: Parallel immune system for graph coloring. W: SOFSEM'08 Proceedings of the 34th conference on Current trends in theory and practice of computer science. Berlin, Heidelberg, 2008, s. 497-505

2.3 Algorytm negatywnej selekcji

Inspiracją do powstania algorytmu negatywnej selekcji jest powstawanie limfocytów T w organizmie. Spełniają one rolę głównych detektorów w układzie odpornościowym. Każdy z nich jest zdolny rozpoznać z użyciem swoich receptorów pewną grupę antygenów. Receptory te tworzone są w wyniku pseudolosowego procesu genetycznego i często zdarza się, że reagują również na białka własnego organizmu. Z tego powodu limfocyty T przechodzą fazę tzw. negatywnej selekcji w grasicy, kiedy to eliminacji ulegają wszystkie te przeciwciała, które rozpoznają własne komórki. Pozostałe są uwalniane do organizmu. Jako ciekawostkę podać można, że mimo zupełnie innej roli limfocytów B w systemie odpornościowym, one również poddawane są negatywnej selekcji. Zasady funkcjonowania tego procesu nie zostały jednak jeszcze dobrze poznane.

Algorytm negatywnej selekcji w informatyce działa w podobny sposób. Najpierw pseudolosowo generowane są komórki danych (których charakterystyka zależna jest od dziedziny zastosowań algorytmu). Następnie sprawdza się ich zgodność z danymi wejściowymi, odrzucając te z nich, które do nich pasują. Kroki te powtarza się, dopóki liczba detektorów nie osiągnie założonego wcześniej pułapu. Gdy detektory są już wygenerowane, testuje się ich zgodność z danymi napływającymi do systemu. Jeśli znajdzie dopasowanie, wnioskuje się, że napływająca dana nie zachowuje charakterystyki danych własnych i może być potraktowana jako potencjalny agresor.

Takie podejście ma kilka zalet:

- każdy zestaw detektorów jest unikatowy, co pozwala zachować skuteczność ochrony nawet przy instalacji systemu na wielu maszynach – jeśli na jednej z nich znajdzie się luka w systemie obrony lub w zestawie detektorów i jakaś dana zostanie błędnie zakwalifikowana jako własna, to prawdopodobieństwo zaistnienia takiej samej sytuacji na innej maszynie nie jest zbyt wysokie;
- detekcja ma charakter probabilistyczny, co sprawia, że dany atak ma nikłe szanse powodzenia na wielu maszynach, a zabezpieczenie kilku stanowisk nie wymaga zapewnienia komunikacji między nimi;
- nie musimy znać każdego potencjalnego wzorca ataku, co zabezpiecza nas przed nieodkrytymi jeszcze metodami jego przeprowadzania; w teorii algorytm jest w stanie wykryć wszystkie odchylenia od danych własnych.

2.4 Pozostałe systemy odpornościowe

Ostatnią grupą algorytmów immunologicznych stanowią tzw. algorytmy hybrydowe. Są to algorytmy, które u podstaw wykorzystują zazwyczaj selekcję klonalną lub negatywną, ale w niektórych szczegółach łączone są np. z algorytmami genetycznymi. Przykładowo algorytm negatywnej selekcji może generować detektory z użyciem algorytmu genetycznego, zamiast robić to całkowicie losowo.

3. Wykorzystanie informacji z logów aktywności serwera Apache

W poprzednim rozdziale omówione zostały algorytmy inspirowane układem odpornościowym człowieka. Czas teraz by zastanowić się, czy możliwe jest ich zastosowanie do analizy tekstowych raportów z działalności serwera.

3.1 Algorytm negatywnej selekcji jako narzędzie do analizy logów

Aby zastosować algorytm negatywnej selekcji do analizy danych tekstowych, musimy na początku określić, co w takim pliku będzie pełnić rolę komórki danych. Najprostszym podejściem (niezależnym od charakterystyki jego zawartości) wydaje się być podział na łańcuchy tekstowe równej długości. W przypadku pliku zawierającego logi, w którym każda linia reprezentuje jeden logiczny egzemplarz danych, podział na łańcuchy można przeprowadzać liniami. Jeżeli na końcu linii zostaje mniej znaków niż założona długość komórki, należy uzupełnić ją znakami pustymi – zerowymi lub spacjami.

Analiza danych tekstowych wymaga również zdefiniowania pojęcia „dopasowania”. Przyjmijmy, że łańcuchy A i B o długości l pasują do siebie wtedy i tylko wtedy, gdy oba zawierają takie same znaki na przynajmniej r pozycjach, przy czym $r \leq l$.

Przykładowo łańcuch „aabaccdbd” pasuje do „abadccadb” przy $r \leq 4$.

Taką samą definicję dopasowania stosuje się zarówno przy generowaniu detektorów, jak również podczas sprawdzania danych w fazie normalnej pracy algorytmu.

W [2] podano metodę, która pozwala wyliczyć prawdopodobieństwo wystąpienia dopasowania w danej sytuacji. Jeśli za m podstawimy ilość symboli, która może wystąpić na każdej pozycji w danej komórce danych, wtedy wzór na prawdopodobieństwo dopasowania (P_m) przyjmuje postać:

$$P_m \approx m^{-r} [(l-r)(m-1)/m+1]$$

Przyjmuje się, że wzór ten prawdziwy jest tylko pod warunkiem, że $m^{-r} \ll 1$.

3.2 Charakterystyka logów aktywności serwera Apache

Format logów serwera Apache może być dowolnie ustawiany przez administratora, ale często jest on pozostawiany w formie domyślnej. Nie wpływa on na użyteczność, jeśli tylko zawiera wszystkie standardowe pola. Zazwyczaj stosuje się jeden z dwóch formatów; tzw. format prosty (ang. *common*) lub złożony (ang. *combined*), które różnią się poziomem szczegółowości, a co za tym idzie – rozmiarem wynikowego pliku dziennika [5].

Format prosty zawiera tylko podstawowe dane, czyli:

- IP klienta (czasem również nazwę jego komputera, oraz identyfikator zalogowanego użytkownika, szczególnie w sieciach wewnętrznych),
- datę i godzinę dostępu,
- charakterystykę żądania, czyli metodę (GET, POST, PUT, etc.), ścieżkę do zasobu, oraz wersję protokołu HTTP,
- kod odpowiedzi HTTP,
- rozmiar zwróconej odpowiedzi w bajtach.

W najnowszej, stabilnej wersji (2.2) ma on zazwyczaj postać jak poniżej:

```
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif
HTTP/1.0" 200 2326
```

Format złożony charakteryzuje się tym, że zawiera dodatkowe nagłówki żądania. Zazwyczaj są to pola **User-Agent** (nazwa programu klienta) oraz **Referer** (poprzednio odwiedzony adres URL), chociaż mogą to być również dowolne inne nagłówki, zależnie od potrzeb. Informacji nagłówkowych nie należy traktować jako pewne, ponieważ to klient decyduje (w zależności od poziomu wiedzy - mniej lub bardziej świadomie), jaka ma być ich treść. Zdarza się również, że nie są one w ogóle podawane. Niezależnie od poziomu wiarygodności mogą one być przydatne w pracy administratora, gdyż wiele źle zabezpieczonych skryptów się do nich odwołuje, przyjmując zawarte w nich dane za bezpieczne. W takiej sytuacji obecność nagłówków w logach pozwala łatwiej zdiagnozować ewentualne ataki lub próby ich przeprowadzenia. Standardowa postać formatu złożonego przedstawia się następująco:

```
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif
HTTP/1.0" 200 2326 "http://www.example.com/start.html" "Mozilla/4.08 [en]
(Win98; I ;Nav) "
```

3.3 Możliwość wykorzystania algorytmu negatywnej selekcji w zakresie analizy logów

Podczas analizy logów serwera Apache warto zwrócić uwagę na jedną ich szczególną cechę: wysoką zmienność pierwszej części każdej linii, tzn. znacznik daty oraz adres IP. Pola te zmieniają się przy każdym żądaniu, a poza tym dyskusyjny jest ich wpływ na wynik analizy, czy dana linia ma związek z potencjalnym atakiem, czy też nie. Sam adres IP niesie, oczywiście, informację na temat atakującego, ale nie identyfikuje go jednoznacznie i niczego nie mówi o samym ataku. Można go fałszować, albo zmieniać w sposób niemal losowy z użyciem sieci serwerów proxy. Co za tym idzie, może być przyczyną błędnie rozpoznanych ataków, zatem do odsiewu adresów znanych jako potencjalne źródła zagrożeń lepiej jest stosować inne metody niż algorytm negatywnej selekcji, np. czarne listy. W celu poprawy jakości wyników algorytmu można usunąć datę i IP z pliku przy użyciu wyrażeń regularnych, np. przed podziałem każdej linii na podciągi równej długości.

W tym miejscu warto zastanowić się, czym jest podział na *własne* i *obce* wpisy w logach. Za wpisy *własne* uznajemy wszystkie te linie, które dotyczą poprawnych żądań do serwera. Wpisy *obce* to z kolei wszelkie próby wstrzyknięcia do oprogramowania zainstalowanego na serwerze podejrzanych danych, np. poprzez podanie ich w ścieżce do zasobu lub nazwie przeglądarki. Do tej grupy zaliczamy również próby wywołania nieistniejących zasobów i usług, żądania które z jakiegoś powodu otrzymały odpowiedź w niestandardowym rozmiarze etc.

Od razu nasuwa się jedna obserwacja – w ten sposób nie da się odrzucić znacznej części ataków typu DoS (ang. *Denial of Service*, Odmowa wykonania usługi). Są to ataki wykorzystujące poprawne zapytania, które wywoływane są z dużą częstotliwością po to, by przeciążyć serwer i uniemożliwić mu obsługiwanie poprawnego ruchu przez określony czas. Algorytm negatywnej selekcji zazwyczaj nie jest w stanie oddzielić ich od zapisu normalnej pracy systemu. Analogiem tej sytuacji w układzie odpornościowym człowieka jest jego reakcja na nowotwory – jako że są to (również zmutowane) komórki własnego organizmu, nie są traktowane jak potencjalni najeźdźcy.

4. Projekt systemu bezpieczeństwa bazujący na algorytmie negatywnej selekcji

Aby wesprzeć zarządzanie bezpieczeństwem serwera Apache, zaproponowano wykorzystanie do tego celu algorytmu negatywnej selekcji. Narzędzie, w którym zastosowano ten algorytm umieszczono w jednej z dalszych warstw zabezpieczeń serwera – na etapie monitorowania jego pracy, czyli analizy logów. Jest ono używane w momencie, kiedy ewentualna próba ataku miała już miejsce, a administrator zajmuje się czytaniem raportów z działania systemu w poszukiwaniu anomalii. Ponieważ ilość danych, z którymi musi się on zapoznać, jest bardzo duża w przypadku serwisu/usługi o znacznym ruchu, czynność ta jest w takich przypadkach bardzo utrudniona. Jakakolwiek metoda automatyzacji tego zadania pozwoli w założeniu skrócić czas jego trwania, a co za tym idzie – pozwoli administratorowi szybciej zareagować na ewentualne skutki ataku, lub (o ile to możliwe) przerwać go w trakcie.

Analiza logów może okazać się ponadto skuteczniejsza, niż przechwytywanie na bieżąco tych żądań, które wydadzą się podejrzane. Powodów tego stanu rzeczy jest kilka. Przede wszystkim nie można uniemożliwić korzystania z serwisu żadnej grupie prawidłowych użytkowników. Nie może być mowy o fałszywych alarmach, a zatem czułość algorytmu musiałaby zostać bardzo zmniejszona, gdyby miał być on używany w czasie rzeczywistym. Poza tym zdarza się, że anomalię wykrywa się nie po samym żądaniu, a po odpowiedzi na nie. Żeby analizować również odpowiedź serwera, należałoby przepuszczać każde żądanie przez tego rodzaju zaporę, co wydaje się być zdecydowanym zaprzeczeniem sensu jej istnienia. Nie bez znaczenia jest też opóźnienie generowane przez każdy rodzaj skanera. Wszystkie powyższe argumenty zaliczyć można na korzyść rozwiązania z umieszczeniem programu do monitorowania żądań dopiero na etapie analizy logów dot. działania serwisu w przeszłości.

4.1 Wymagania funkcjonalne

1. System umożliwia analizę logów serwera Apache w standardowym ich formacie.
2. System analizuje dane w sposób ogólny, nie zawierając dostosowań dot. konkretnego serwisu internetowego lub innej usługi dostępnej online.
3. Reprezentacją systemu jest program z graficznym interfejsem użytkownika, co ułatwi proces analizy.

4. Program nie wymaga dostępu ani do internetu, ani do zabezpieczanego systemu. Celem jest możliwość jego zastosowania nawet wtedy, gdy logi zostaną, przykładowo, wysłane e-mailem do analizy.
5. Program wymaga podania w pierwszej fazie pracy pewnego zbioru danych bazowych, które przedstawiają poprawną pracę zabezpieczanego systemu. Ilością ładowanych danych musi zarządzać operator. Program nie jest bowiem w stanie samodzielnie określić, czy posiada już wystarczająco kompletny obraz danych *własnych* by poprawnie przeprowadzić analizę.
6. Program umożliwia dowolne ustawianie tych parametrów algorytmu negatywnej selekcji, które mają wpływ na jakość analiz i czas obliczeń.
7. Po wczytaniu danych bazowych i wygenerowaniu detektorów program umożliwia szybką analizę pliku, który się do niego załaduje. Po analizie wyświetla komunikat z informacją, czy jakieś wpisy zakwalifikowano jako *obce*.
8. Wynik analizy daje się zapisać do pliku CSV (*Comma Separated Values*, Wartości rozdzielone przecinkami), aby można go było potem analizować z użyciem narzędzi do obróbki danych, takich jak np. Microsoft Excel lub inne, bardziej specjalistyczne.

4.2 Wymagania techniczne

Program tworzony jest w języku C# dla środowiska .NET w wersji co najmniej 3.5. Pozwala to uruchamiać go w większości systemów operacyjnych z rodziny Microsoft Windows, a także we wszystkich innych systemach, na które przeniesiono platformę .NET, m.in. na niektórych dystrybucjach Linuksa. Wymagania sprzętowe są zatem tożsame z wymaganiami platformy .NET. Komputery o lepszych parametrach technicznych powinny być jedynie w stanie nieco szybciej generować detektory i analizować pliki. W całym projekcie stosuje się techniki programowania obiektowego.

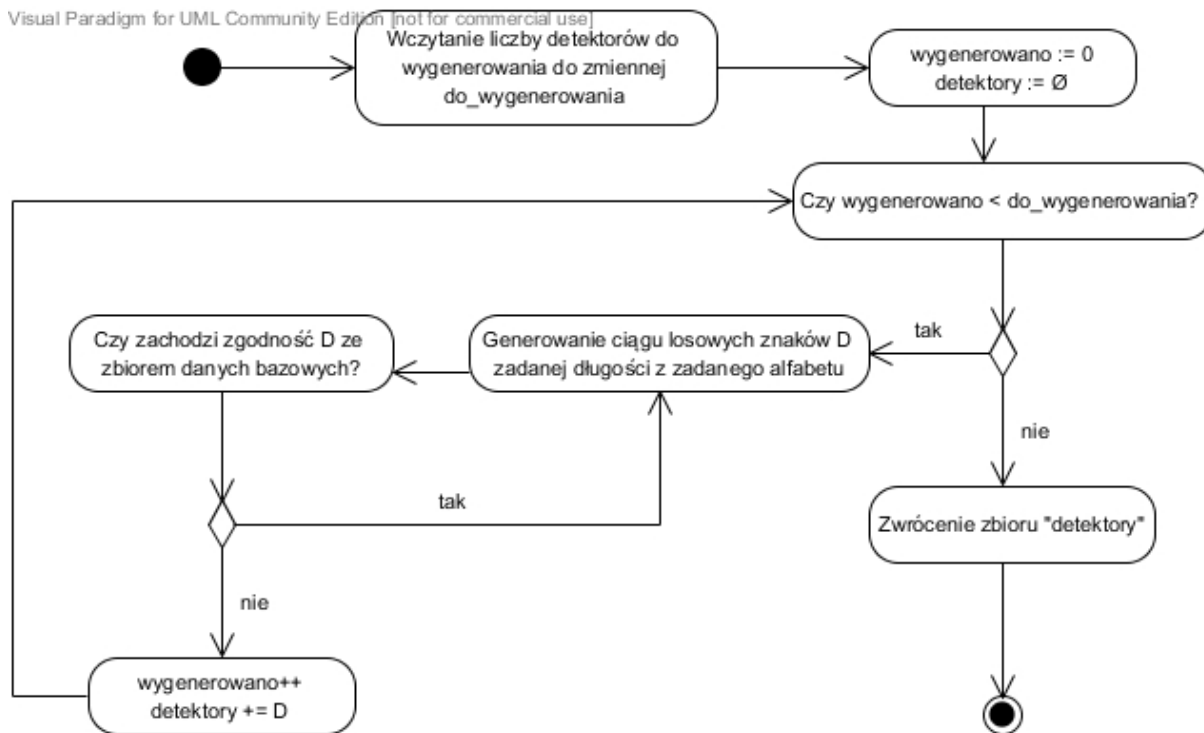
4.3 Analiza wymagań

Wymaganie dotyczące umożliwienia użytkownikowi programu dowolnego ustawiania parametrów algorytmu sugeruje konieczność zaimplementowania jakiegoś mechanizmu kontroli tych parametrów. Dobrym pomysłem jest zatem wyliczanie na bieżąco wg wzoru zamieszczonego w Rozdziale 3.1 prawdopodobieństwa dopasowania na podstawie wprowadzonych parametrów. Zbyt dużą jego wartość, przykładowo większą lub równą 0,1 można oznaczyć kolorem czerwonym, a wartości mniejsze – zielonym. W ten sposób na

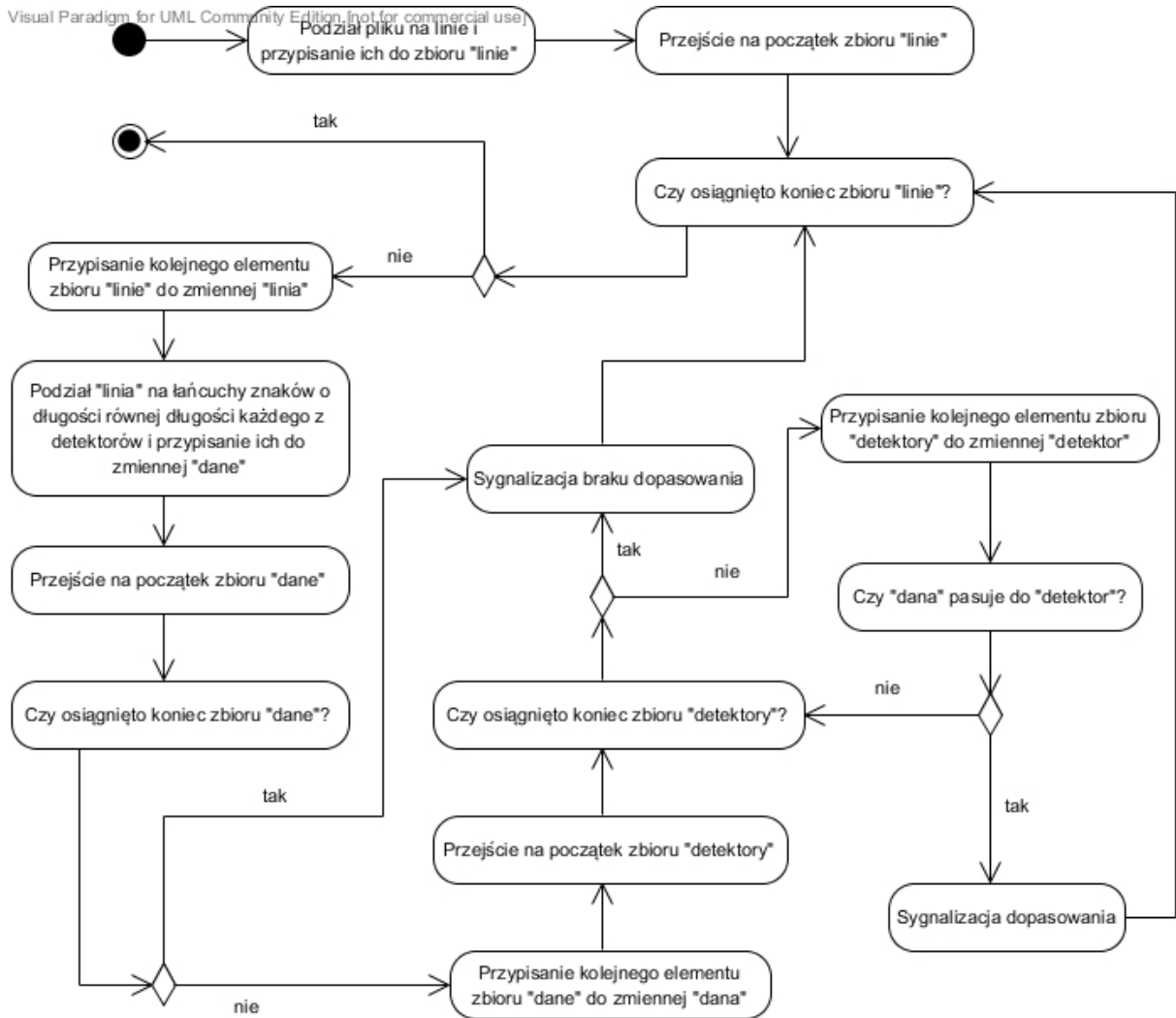
bieżąco można będzie skontrolować, czy analiza danych (która może być wymagająca czasowo), ma szansę zwrócić poprawne wyniki, czy nie.

5. Implementacja systemu

5.1 Diagramy wybranych algorytmów programu

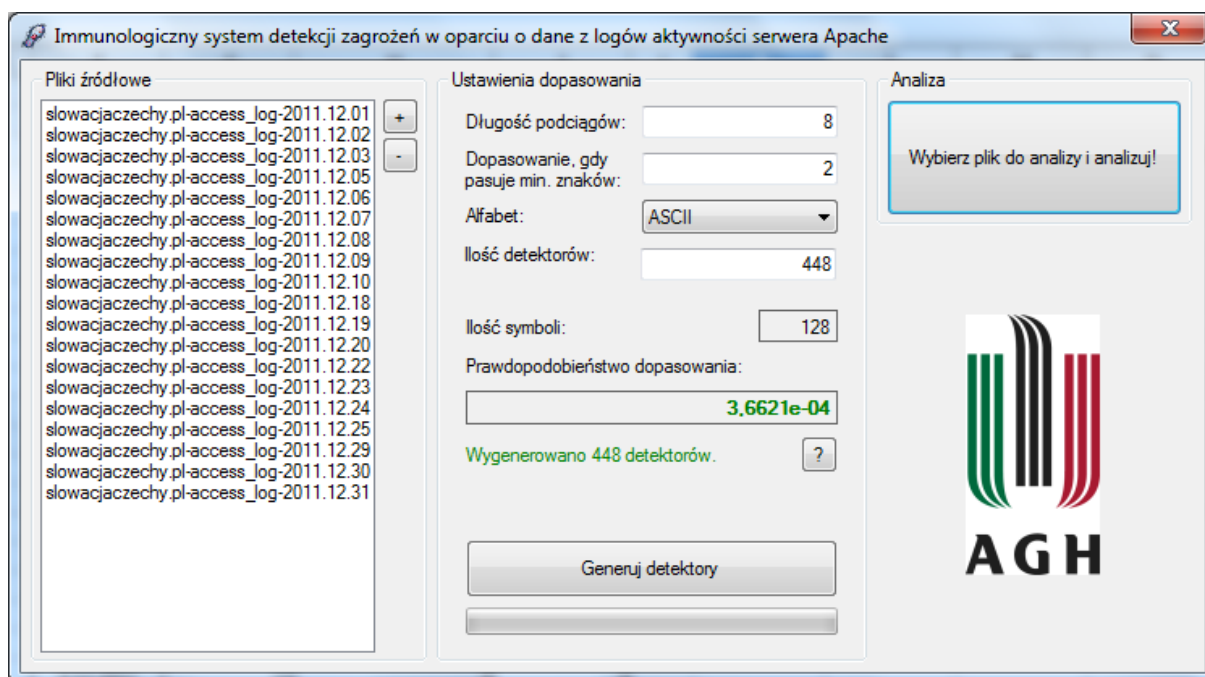


Rysunek 1: Algorytm generowania detektorów



Rysunek 2: Sprawdzanie pliku z danymi na zgodność z danymi bazowymi

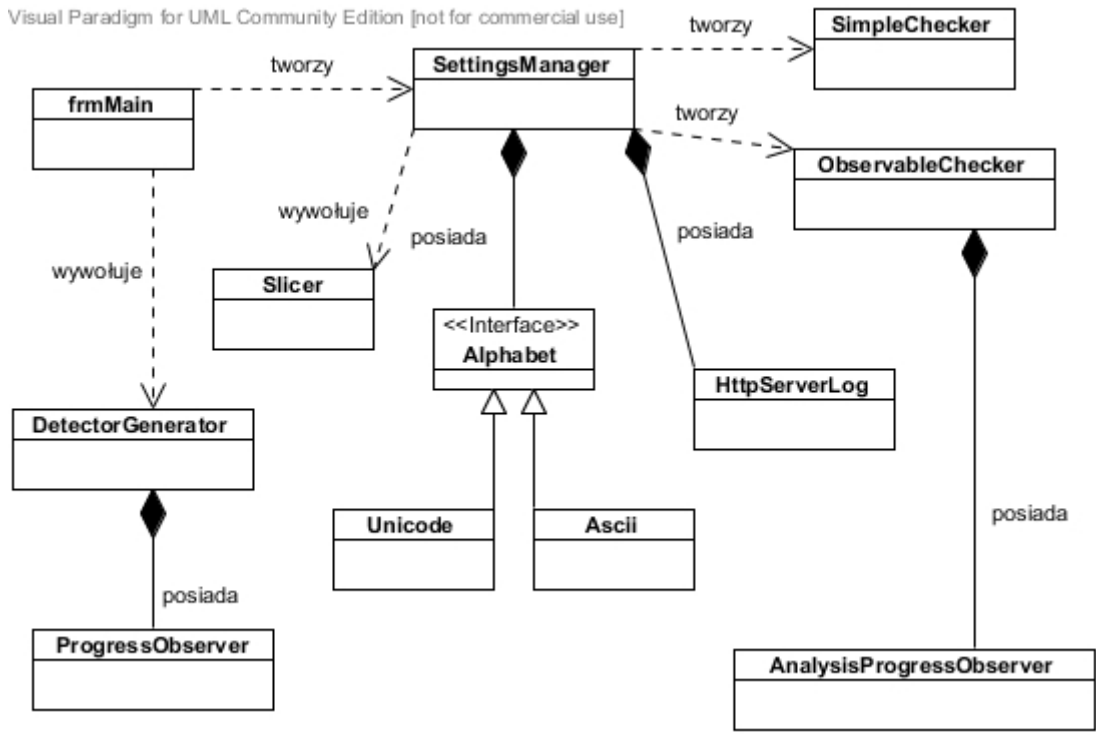
5.2 Interfejs programu



Interfejs programu podzielony jest na trzy kolumny, reprezentujące trzy podstawowe fazy procesu analizy. Pierwsza z lewej odpowiada za wczytywanie plików z danymi źródłowymi. Kliknięcie przycisku ze znakiem „+” umożliwia dodanie jednego lub wielu plików do listy, a kliknięcie przycisku „-” - usunięcie zaznaczonego wpisu. Środkowa kolumna odpowiada procesowi generowania detektorów. Można tam ustawiać parametry algorytmu i na bieżąco obserwować wpływ ich zmiany na prawdopodobieństwo dopasowania. Zakończeniem tej fazy jest kliknięcie przycisku „Generuj detektory”, który po zakończeniu procesu generacji uaktywnia trzecią kolumnę, odpowiedzialną za analizę. Kliknięcie przycisku „Wybierz plik do analizy i analizuj!” powoduje wyświetlenie okna dialogowego wyboru pliku. Akceptacja wyboru powoduje rozpoczęcie procesu analizy. Po jej zakończeniu pokazuje się komunikat z informacją, ile wpisów zostało zakwalifikowanych jako obce oraz monit z pytaniem, czy użytkownik chce dokonać zapisu raportu z analizy do pliku CSV. Kolejne naciśnięcie na ten przycisk umożliwia analizę kolejnego pliku z użyciem tego samego zestawu detektorów.

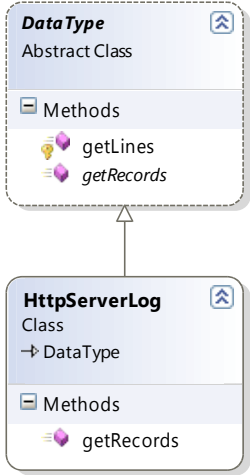
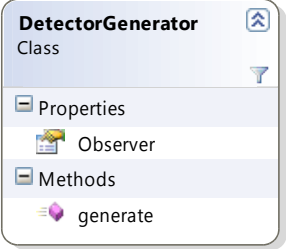
5.3 Opis kompetencji i diagramy klas

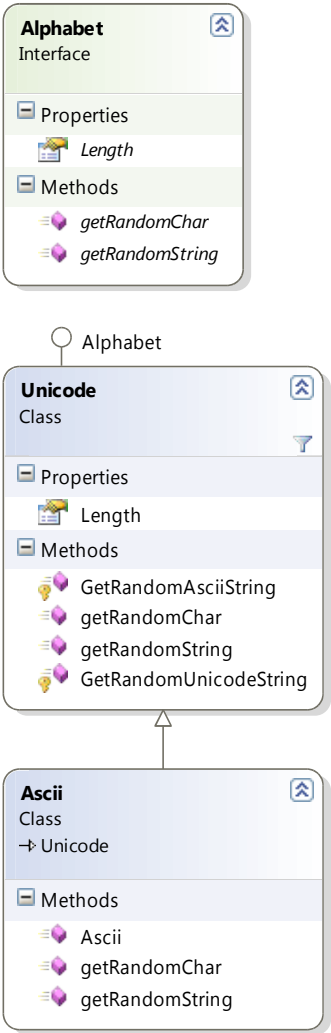
Główne klasy programu odpowiadają głównym fazom algorytmu negatywnej selekcji. Program zawiera także kilka klas pomocniczych, narzędziowych. Ogólny schemat zależności między klasami przedstawiony został na Rysunku 3.

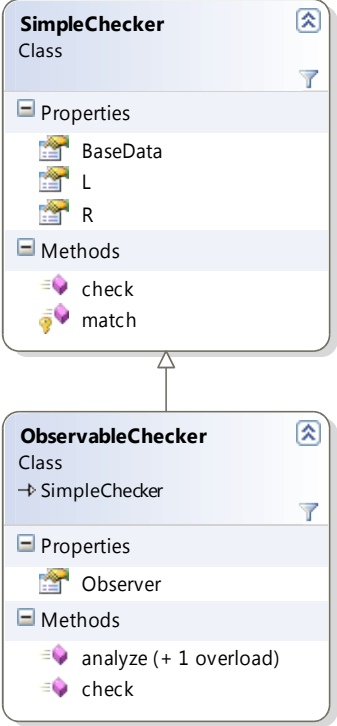


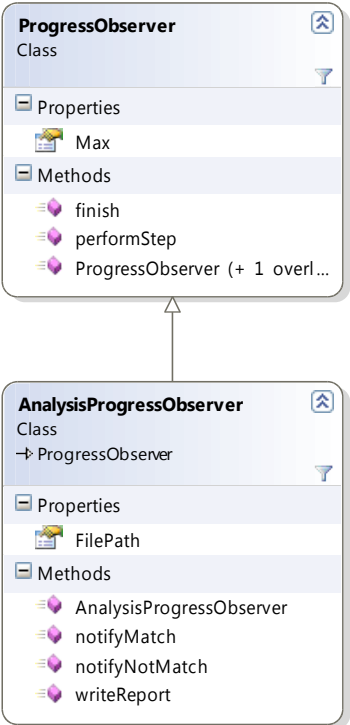
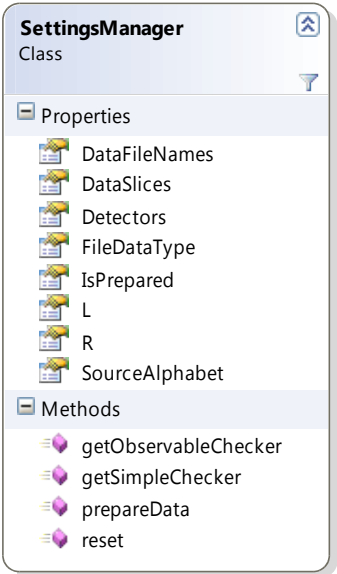
Rysunek 3: Uproszczone zależności między klasami programu

| Klasa / Interfejs | Rola w programie |
|--|--|
| <p>frmMain</p> <pre> classDiagram class frmMain class SettingsManager class Slicer class DetectorGenerator class ProgressObserver class Alphabet class Unicode class Ascii class SimpleChecker class ObservableChecker class HttpServerLog class AnalysisProgressObserver frmMain ..> SettingsManager : tworzy frmMain ..> Slicer : wywołuje Slicer ..> DetectorGenerator : wywołuje DetectorGenerator *-- ProgressObserver : posiada SettingsManager *-- Alphabet : posiada SettingsManager *-- SimpleChecker : tworzy SettingsManager *-- ObservableChecker : tworzy Alphabet < -- Unicode Alphabet < -- Ascii HttpServerLog *-- SettingsManager : posiada AnalysisProgressObserver *-- ObservableChecker : posiada </pre> | <p>Klasa głównego okna. Dziedziczy po klasie Form z przestrzeni nazw System.Windows.Forms. Odpowiada za powiązanie użytkowych klas programu z interfejsem użytkownika i za obróbkę wprowadzonych przez niego danych.</p> <p>Ważniejsze metody to między innymi:</p> <ul style="list-style-type: none"> - calcPm(), która wylicza do celów poglądowych prawdopodobieństwo udanego dopasowania i wyświetla je na jednej z etykiet tekstowych, - btnGenerateDetectors_Click(), która konfiguruje klasy odpowiedzialne za generowanie detektorów, - btnAnalyse_Click(), która konfiguruje proces analizy kolejnego pliku wejściowego. |

| Klasa / Interfejs | Rola w programie |
|---|---|
| <p>DataType i jej pochodna HttpServerLog</p>  <pre> classDiagram class DataType { <<abstract>> +getLines() +getRecords() } class HttpServerLog { +DataType +getRecords() } DataType < -- HttpServerLog </pre> | <p>DataType jest abstrakcyjną reprezentacją formatu pliku wejściowego..</p> <p>Składowe:</p> <ul style="list-style-type: none"> - getLines(), chroniona metoda pozwalająca klasom pochodnym na łatwe wczytywanie pliku liniami; - getRecords(), metoda która z pliku podanego jako parametr ma zwrócić listę komórek danych, czyli łańcuchów znaków. <p>Klasa pochodna HttpServerLog zawiera dostosowania względem formatu logów serwera Apache, tzn. przed zwróceniem linii załadowanego pliku wycina z niego datę i adres IP, jak to opisano wcześniej.</p> |
| <p>DetectorGenerator</p>  <pre> classDiagram class DetectorGenerator { +Observer +generate() } </pre> | <p>Jest to klasa odpowiedzialna za generowanie detektorów. Można do jej właściwości Observer przypisać obiekt klasy ProgressObserver, który powiadamia obiekty interfejsu użytkownika o postępach procesu.</p> <p>Klasa zawiera tylko jedną metodę, generate(). Jej zadaniem jest generowanie podanej jako parametr liczby detektorów, ze znaków podanego parametrem alfabetu. Sprawdzenie, czy detektor nie pasuje do danych własnych odbywa się z użyciem podanego jako parametr obiektu klasy SimpleChecker.</p> |

| Klasa / Interfejs | Rola w programie |
|---|--|
| <p data-bbox="405 212 766 271">Interfejs Alphabet i klasy go implementujące (Unicode, Ascii)</p>  <pre data-bbox="419 315 751 1344"> classDiagram class Alphabet { <<interface>> Length getRandomChar() getRandomString() } class Unicode { Length GetRandomAsciiString() getRandomChar() getRandomString() GetRandomUnicodeString() } class Ascii { Ascii() getRandomChar() getRandomString() } Alphabet < -- Unicode Alphabet < -- Ascii Unicode < -- Ascii </pre> | <p data-bbox="831 212 1303 304">Interfejs Alphabet reprezentuje ogólny schemat klasy alfabetu, czyli zestawu symboli, z których generuje się detektory.</p> <p data-bbox="831 338 946 365">Składowe:</p> <ul data-bbox="879 376 1303 656" style="list-style-type: none"> - właściwość Length (tylko do odczytu), którą należy rozumieć jako ilość symboli, którą zawiera alfabet; - getRandomChar() zwraca losowy symbol alfabetu; - getRandomString() zwraca losowy ciąg znaków, o długości podanej jako parametr. <p data-bbox="831 719 1303 958">Klasa Ascii dziedziczy po klasie Unicode, gdyż metoda generowania losowych znaków i ciągów jest taka sama dla tych dwóch zestawów znaków. Zmieniają się jedynie ograniczenia dot. maksymalnej wartości liczbowej pojedynczego znaku, a dziedziczenie pozwala na uniknięcie dublowania kodu.</p> |

| Klasa / Interfejs | Rola w programie |
|--|--|
| <p data-bbox="408 215 762 271">SimpleChecker i pochodna klasa ObservableChecker</p>  <pre> classDiagram class SimpleChecker { BaseData L R check() match() } class ObservableChecker { Observer analyze(+ 1 overload) check() } SimpleChecker -- > ObservableChecker </pre> | <p data-bbox="831 215 1295 304">SimpleChecker to klasa odpowiedzialna za sprawdzanie zgodności danych ze wzorcami.</p> <p data-bbox="831 338 975 365">Właściwości:</p> <ul data-bbox="879 371 1295 808" style="list-style-type: none"> - L, czyli długość jednej komórki danych; - R, czyli ilość znaków, które muszą być takie same na tych samych pozycjach w dwóch porównywanych ciągach, by zaszło dopasowanie; - BaseData, czyli dane bazowe, z którymi porównujemy dane wysyłane jako parametry opisanych dalej metod; podczas generowania detektorów są to dane źródłowe, podczas analizy zaś – detektory. <p data-bbox="831 842 922 869">Metody:</p> <ul data-bbox="879 875 1295 1032" style="list-style-type: none"> - match(), która decyduje o zajściu dopasowania dwóch ciągów; - check(), która sprawdza, czy dana pojedyncza komórka pasuje do BaseData; <p data-bbox="831 1066 1295 1245">Klasa ObservableChecker pozwala na analizę napływających danych i monitorowanie tego procesu. Zawiera dodatkową właściwość Observer, której można przypisać obiekt klasy AnalysisProgressObserver.</p> <p data-bbox="831 1279 922 1305">Metody:</p> <ul data-bbox="879 1312 1295 1749" style="list-style-type: none"> - uaktualniona metoda check(), która sprawdza, czy dana komórka pasuje do BaseData oraz powiadamia obiekt zapisany w składowej Observer o udanym dopasowaniu, - przeciążona metoda analyze(), która analizuje (w zależności od wariantu) jedną lub wiele linii pliku z danymi i zwraca ilość linii (lub komórek), które nie pasują do bazy; dodatkowo powiadamia ona obiekt Observer, gdy dopasowanie nie zajdzie. |
| <p data-bbox="400 1798 767 1854">CsvFile, CsvRecord, CsvRecords, CsvReader, CsvWriter</p> | <p data-bbox="831 1798 1295 1944">Klasy komponentu³ odpowiedzialnego za zapis danych w formacie CSV. W skrócie umożliwiają one zapis do pliku tekstowego danych przechowywanych w przejrzystej i łatwej w obróbce formie obiektowej.</p> |

| Klasa / Interfejs | Rola w programie |
|--|---|
| <p>ProgressObserver i jej klasa pochodna AnalysisProgressObserver</p>  | <p>ProgressObserver to klasa będąca częścią pewnej wariacji wzorca projektowego Obserwator [6]. Zajmuje się przygotowaniem, zerowaniem oraz uaktualnianiem paska postępu podczas wykonywania czasochłonnej operacji.</p> <p>Klasa AnalysisProgressObserver zawiera kilka dodatkowych metod, które wykorzystywane są tylko w fazie analizy napływających danych. Zajmują się one przygotowaniem raportu z analizy, który można potem zapisać do pliku CSV z wykorzystaniem metody writeReport(). Dane do raportu wysyła się wywołując metody po każdym udanym i nieudanym dopasowaniu.</p> |
| <p>SettingsManager</p>  | <p>Klasa czysto narzędziowa, przechowująca główne dane programu. Jej właściwości przechowują m.in. detektory, nazwy plików źródłowych, wczytane komórki danych, matematyczne parametry analizy etc.</p> <p>Klasa ta zawiera również metody, które pozwalają na szybszy dostęp do danych, np. prepareData(), która inicjuje proces wczytania i podziału plików wejściowych na komórki.</p> |

3 Komponent ten został udostępniony za darmo (na licencji MIT) w formie otwartego kodu źródłowego na stronie <http://www.codeproject.com/KB/cs/CsvReaderAndWriter.aspx> [dostęp 05.01.2012r.]

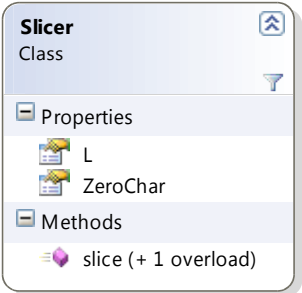
| Klasa / Interfejs | Rola w programie |
|---|--|
| <p style="text-align: center;">Slicer</p>  | <p>Klasa zajmująca się podziałem jednego lub wielu ciągów na podciągi stałej długości. Długość podciągów oraz znak ich uzupełnienia, gdy dane wejściowe są za krótkie, definiuje się poprzez właściwości L i ZeroChar.</p> <p>Przeciążona metoda slice() inicjuje podział i zwraca dane w formie pociętej.</p> |

Tabela 1: Opis kompetencji klas programu

6. Badania skuteczności programu

Przed rozpoczęciem analizy wyników zwróconych przez program, warto pochylić się nad analizą danych, na których pracował. Dokładne omówienie nie tylko obrabianych plików, ale również ataków, o których wzmianki się w nich znajdują, pozwoli na lepsze zrozumienie problemów, które zostały przez niego napotkane.

6.1 Pochodzenie i organizacja danych testowych

Dane testowe zostały zebrane na maszynie działającej pod kontrolą systemu Debian Linux. Jest to serwer o 16GB pamięci RAM, udostępniony kilku niezależnym użytkownikom biznesowym z aktywnymi limitami wykorzystania zasobów (dedykowany serwer współdzielony). Na komputerze tym uruchomiony jest serwer Apache w wersji 2.2. Dane testowe, które poddano dalszej analizie dotyczą zainstalowanego na serwerze portalu, który dostępny jest pod adresem <http://slowacjaczecchy.pl/>. Wybór serwisu internetowego został podyktowany faktem, że jest on słabo zabezpieczony na poziomie skryptów server-side. W wyniku normalnych działań administratora wykryto ponadto, że został on poddany trwającej kilka dni próbie ataku w grudniu 2011. Dane testowe dotyczą okresu od 1 do 31 grudnia. Podzielono je na dwie grupy – te, które przedstawiają tylko normalną pracę serwera i te, które zawierają również ślady ataków. Podziału dokonano ręcznie, na podstawie klasycznej analizy logów przez ich lekturę. Pliki, które nie zawierały śladów ataku, zostały załadowane do programu jako dane bazowe.

6.2 Charakterystyka ataku na serwis testowy

Atak na serwis prawdopodobnie został przeprowadzony z użyciem jakiegoś automatu testującego serwisy na obecność starych wersji oprogramowania. Automat ten w zależności od wyniku testów podejmuje decyzję o ewentualnych dalszych próbach nieautoryzowanego dostępu. Świadczą o tym powtarzające się wywołania następujących adresów przez host o adresie IP 122.255.96.163 i przeglądarce o sygnaturze „Mozilla/5.0 (Windows NT 6.1; WOW64; rv:8.0) Gecko/20100101 Firefox/8.0” (przykłady z 04.12.2011r.):

```
/scripts/awstats.pl?configdir=|echo;echo%20YYAAZ;uname;id;echo%20YYY;echo|
/stats/awstats.pl?configdir=|echo;echo%20YYAAZ;uname;id;echo%20YYY;echo|
/apps/phpAlbum/main.php?cmd=setquality&var1=1%27.passthru%28%27id
%27%29.%27;
/phpAlbum/main.php?cmd=setquality&var1=1%27.passthru%28%27id%27%29.%27;
/main.php?cmd=setquality&var1=1%27.passthru%28%27id%27%29.%27;
/phpalbum/main.php?cmd=setquality&var1=1%27.passthru%28%27id%27%29.%27;
/apps/phpalbum/main.php?cmd=setquality&var1=1%27.passthru%28%27id
%27%29.%27;
```

Przedstawione powyżej wpisy świadczą o próbach ataku na stare wersje oprogramowania do analizy statystyk AWStats⁴ i na popularny skrypt internetowych galerii obrazów phpAlbum⁵. Analiza geograficzna adresu IP sugeruje, że znajduje się on na terenie Malezji, w Kuala Lumpur⁶.

W dniu 11.12.2011r. atak kontynuowany był z trzech adresów IP. Analiza geograficzna identyfikuje żądania jako pochodzące z miast Gramado (Brazylia), Kuala Lumpur (Malezja, inny adres niż poprzednio) i Astrachań (Rosja). Do wykorzystywanych metod dołączył kolejny typ żądania:

```
/?file=../../../../../../../../proc/self/environ%00"
```

Ten atak był ciekawszy, z uwagi na dodatkową wartość zawartą w nagłówku User-Agent:

```
<?php system(\"id\"); ?>
```

Była to ewidentna próba wykorzystania potencjalnej luki w skrypcie, która miałaby prowadzić do wykonania dowolnego kodu zamieszczonego w nagłówku. Mogłoby do tego

4 Opis testowanych luk w oprogramowaniu AWStats znajduje się pod adresem <http://blog.spiderlabs.com/2011/12/honeypot-alert-awstats-command-injection-scanning-detected.html>

5 Opis testowanych luk w oprogramowaniu phpAlbum znajduje się pod adresem <http://blog.spiderlabs.com/2011/12/honeypot-alert-phpalbum-php-code-execution-attacks.html>

6 Analiza wykonana przy użyciu serwisu <http://www.ip-address.org/lookup/ip-locator.php> w dniu 05.01.2012r.

doprowadzić mylne założenie autora skryptu, że dane tam zawarte należy traktować jako bezwarunkowo bezpieczne.

Począwszy od 11.12.2011r., ataki tego rodzaju miały miejsce prawie codziennie aż do 29.12.2011r., z wyjątkiem przerw od 18. do 20.12.2011r. i od 22. do 25.12.2011r.

15.12.2011r. zaobserwowano próbę przełamania zabezpieczeń systemu zarządzania treścią Joomla. Zasada działania ataku była taka sama jak 4 dni wcześniej.

16.12.2011r. do wcześniej zaobserwowanych ataków dołączył nowy, polegający na skanowaniu, czy na serwerze znajduje się publicznie dostępny skrypt administracji bazą danych MySQL phpMyAdmin w nieaktualnej wersji.

21.12.2011r. po kilkudniowej przerwie spektrum podejrzanych żądań poszerzyło się o skanowanie serwera w poszukiwaniu nieaktualnych wersji systemów zarządzania treścią, między innymi WordPress i Typo3.

26.12.2011r. po kolejnej przerwie zaobserwowano kontynuację testowania występowania skryptów – starych wersji edytora TinyMCE, nierozwijanego już systemu CMS⁷ Mambo, biblioteki do wytwarzania miniaturk zdjęć phpThumb oraz pluginów do systemu CMS Joomla.

27.12.2011r. rozpoczęto sprawdzanie występowania popularnych skórek do systemu WordPress, które zawierają luki w zabezpieczeniach.

28.12.2011r. przeszukiwanie serwera w postaci niezabezpieczonej wersji phpMyAdmin zostało przeprowadzone bardziej szczegółowo – przetestowano prawie wszystkie wersje od 1.1.0 do 3.4.9. Oprócz tego przetestowano standardowe nazwy katalogów przeznaczonych na część administracyjną systemów, m.in. admin, administrator, admin1 etc.

29.12.2011r. ataki ustały i nie powtórzyły się aż do chwili obecnej (07.01.2012r.)

Wszystkie podejrzane żądania charakteryzowały się:

- krótkim czasem trwania serii (maks. ok. 10 sek.),
- szerokim spektrum adresów IP (praktycznie z każdego kontynentu),
- typowymi przejawami automatyzacji, tzn. jeśli przetestowanie listy standardowych luk niczego nie dało, atak nie był kontynuowany ręcznie.

7 CMS – Content Management System - System zarządzania treścią

Wszystkie wymienione powyżej cechy potwierdzają teorię o działaniu automatu. Można bowiem uznać za mało prawdopodobne, by internauci z kilku kontynentów mogli cokolwiek zyskać dzięki atakowaniu niezbyt popularnego, niepełniącego funkcji zarobkowej serwisu na terenie Polski. Możliwym celem ataku mogło być przekształcenie źle zabezpieczonego serwisu w część botnetu⁸, lub w automat spamowy.

6.3 Analiza logów przy użyciu przygotowanego programu, z wykorzystaniem algorytmu negatywnej selekcji

6.3.1 Dane kontrolne

Aby móc właściwie ocenić skuteczność stworzonego programu, warto zapoznać się z charakterystyką danych testowych. W Tabeli 2 przedstawiono, ile linii zawiera plik z każdego dnia miesiąca i ile z nich dotyczy ataków, a ile reprezentuje normalny ruch. Dane te uzyskano dzięki analizie plików przez człowieka, tak aby mogły z powodzeniem odgrywać rolę próby kontrolnej.

| Dzień | Linii | Atak | Popr. | Dzień | Linii | Atak | Popr. | Dzień | Linii | Atak | Popr. |
|-------|-------|------|-------|-------|-------|------|-------|-------|-------|------|-------|
| 01.12 | 45 | 0 | 45 | 11.12 | 135 | 75 | 60 | 21.12 | 110 | 40 | 70 |
| 02.12 | 190 | 0 | 190 | 12.12 | 2188 | 125 | 2063 | 22.12 | 23 | 0 | 23 |
| 03.12 | 34 | 0 | 34 | 13.12 | 113 | 50 | 63 | 23.12 | 19 | 0 | 19 |
| 04.12 | 93 | 20 | 73 | 14.12 | 177 | 25 | 152 | 24.12 | 52 | 0 | 52 |
| 05.12 | 176 | 0 | 176 | 15.12 | 289 | 25 | 264 | 25.12 | 112 | 0 | 112 |
| 06.12 | 45 | 0 | 45 | 16.12 | 100 | 19 | 81 | 26.12 | 373 | 120 | 253 |
| 07.12 | 78 | 0 | 78 | 17.12 | 53 | 26 | 27 | 27.12 | 152 | 80 | 72 |
| 08.12 | 92 | 0 | 92 | 18.12 | 2 | 0 | 2 | 28.12 | 254 | 205 | 49 |
| 09.12 | 44 | 0 | 44 | 19.12 | 2155 | 0 | 2155 | 29.12 | 47 | 0 | 47 |
| 10.12 | 48 | 0 | 48 | 20.12 | 368 | 0 | 368 | 30.12 | 102 | 0 | 102 |
| | | | | | | | | 31.12 | 59 | 0 | 59 |

Tabela 2: Charakterystyka danych testowych

6.3.2 Metodologia badań i organizacja wyników

Badania zostały przeprowadzone według następującego schematu:

1. Spośród plików z logami wyodrębnione zostały te, w których nie występują żadne ślady ataków.
2. Wyodrębnione pliki zostały załadowane do programu jako dane bazowe.
3. Parametry algorytmu zostały ustawione w sposób opisany w dalszej części rozdziału.

⁸ Botnet, czyli sieć komputerów zarządzanych przez programy-roboty, najczęściej sterowana przez organizacje przestępcze lub pojedynczych hakerów. Wykorzystywana głównie do rozsyłania spamu lub ataków DDoS.

4. Wygenerowany został zestaw detektorów.
5. Każdy plik zaklasyfikowany jako posiadający wpisy dot. ataku został kolejno poddany analizie, a raport z niej został zapisany do pliku w odpowiednim katalogu.
6. Każdy plik raportu został następnie poddany analizie przez człowieka, aby poznać ilość fałszywych alarmów oraz poprawnie wykrytych i pominiętych zagrożeń.

Ilości detektorów używane w testach mają związek z ilością możliwych symboli używanego alfabetu i innymi ustawieniami algorytmu. Oczywiście jest bowiem, że matematycznie zależne jest od nich m.in. prawdopodobieństwo dopasowania. Aby móc miarodajnie porównywać wyniki, zaproponowano zatem pewną miarę ilości detektorów, która bierze te czynniki pod uwagę. Przyjęto, że bazowa ilość detektorów c dana jest wzorem:

$$c=r \cdot l \cdot \log_2(a)$$

gdzie l to długość podciągu, r to ilość pasujących symboli, niezbędnych by zaszło dopasowanie, a a to ilość symboli alfabetu.

Eksperymenty przeprowadzono dla następujących ilości detektorów: c , $2c$ i $4c$.

Przy każdej testowanej liczbie detektorów, wszystkie próby analizy plików, które podano jako źródło danych bazowych (czyli nie zawierających informacji o atakach) wykazały poprawny wynik: 0 wykrytych i niewykrytych anomalii. Z tego powodu w tabelach nie uwzględniono logów z tych dni, kiedy żaden atak nie miał miejsca.

6.3.3 Problemy z danymi zapisanymi jako Unicode

Testowane logi zapisane są w formacie ASCII, co ma związek z ustawieniami serwera, na którym je zebrano. Stosowany przez niego format logów nie przewiduje bowiem czytelnego dla człowieka formatu znaków spoza zakresu ASCII. Jeśli taki znak pojawiłby się w żądaniu, zostałby poddany kodowaniu w taki sam sposób, jak kodowane są adresy URL, przykładowo polskiej literze „ł” odpowiadałby ciąg „%C5%82”. Gdyby zaś taki znak pojawił się w jednym z nagłówek, np. w polu User-Agent, zostałby on zapisany jako kod znaku poprzedzony ciągiem „/u”.

Warto w tym momencie zaznaczyć, że umieszczanie w nagłówkach żądania protokołu HTTP znaków spoza zestawu ISO-8859-1 bez odpowiedniego kodowania jest niezgodne ze specyfikacjami RFC 2616 i RFC 2047⁹. Z tego powodu każdy, kto takie znaki wyśle, nie

⁹ RFC odpowiada ang. *Request For Comments*, czyli dosłownie: Prośba o komentarze. Są to zbiory technicznych dokumentów związanych z Internetem, z których każdy ma swój unikalny numer. Wiele z nich przekształciło się w oficjalne standardy, inne traktowane są w ten sposób nieoficjalnie. Dokumenty, na które

może liczyć na to, że zostaną one prawidłowo rozpoznane. Jako że Unicode zawiera znaki dwubajtowe, to bez odpowiednich oznaczeń zostaną one potraktowane po prostu jako dwa oddzielne znaki ASCII.¹⁰

Serwer Apache pozwala na instalację własnego modułu zapisu logów, który może w inny sposób traktować rozszerzone znaki Unicode. Dla takich przypadków stworzony program posiada opcję wyboru alfabetu, na którym operuje.

Jako ciekawostkę podać można, że niepotrzebne rozszerzenie alfabetu do Unicode wpływa niekorzystnie na wyniki. Przy użyciu programu przetestowano pliki z wykrytymi ręcznie atakami dla następujących ilości detektorów: c (240), $2c$, $4c$ i $10c$. Każda próba przyniosła 0 wykrytych anomalii. Zwiększeniu uległ jedynie czas generowania detektorów i sprawdzania każdej linii, z uwagi na większy zbiór danych do obróbki.

Z wymienionych powyżej powodów wszystkie tabele zamieszczone poniżej dotyczą zestawu znaków ASCII.

6.3.4 Raporty z analizy dla długości słowa $l=8$ i stałej dopasowania $r=2$

| Dzień | Wykrytych anomalii | Niewykrytych anomalii | Falszywych alarmów |
|-------|--------------------|-----------------------|--------------------|
| 04.12 | 11 | 9 | 7 |
| 11.12 | 36 | 39 | 0 |
| 12.12 | 60 | 65 | 1 |
| 13.12 | 24 | 26 | 1 |
| 14.12 | 12 | 13 | 6 |
| 15.12 | 12 | 13 | 1 |
| 16.12 | 0 | 19 | 3 |
| 17.12 | 12 | 14 | 0 |
| 21.12 | 27 | 13 | 0 |
| 26.12 | 60 | 60 | 0 |
| 27.12 | 47 | 33 | 5 |
| 28.12 | 0 | 205 | 0 |

Tabela 3: Działanie programu dla $l=8$, $r=2$ i 112 (c) detektorów

powołano się w pracy, dostępne są pod adresami <http://www.ietf.org/rfc/rfc2047.txt> oraz <http://www.ietf.org/rfc/rfc2616.txt> [dostęp 05.01.2012r.]

¹⁰ Kolejną ciekawostką jest fakt, że mimo tego, że specyfikacja RFC 2616 zezwala na zakodowane w sposób zgodny z RFC 2047 znaki Unicode, to jest to wewnętrznie sprzeczne z wykorzystywaną przez nią specyfikacją RFC 822 (dot. ogólnego formatu listy nagłówków). Co za tym idzie, żadna współczesna przeglądarka internetowa nie akceptuje zakodowanych w ten sposób danych. Sam serwer Apache jeśli dostanie w nagłówku dane zakodowane jako RFC 2047, zdaje się ignorować ten system kodowania. Zastosowanie silnika zapisu logów obsługującego Unicode może zatem co najwyżej ułatwić czytanie logów administratorom posługującym się językami takimi jak np. chiński, czy koreański, ale nie ma wpływu na bezpieczeństwo.

| Dzień | Wykrytych anomalii | Niewykrytych anomalii | Falszywych alarmów |
|--------------|---------------------------|------------------------------|---------------------------|
| 04.12 | 11 | 9 | 3 |
| 11.12 | 47 | 28 | 2 |
| 12.12 | 81 | 44 | 2 |
| 13.12 | 30 | 20 | 3 |
| 14.12 | 15 | 10 | 13 |
| 15.12 | 15 | 10 | 1 |
| 16.12 | 0 | 19 | 6 |
| 17.12 | 15 | 11 | 0 |
| 21.12 | 40 | 0 | 4 |
| 26.12 | 105 | 15 | 1 |
| 27.12 | 75 | 5 | 4 |
| 28.12 | 7 | 198 | 0 |

Tabela 4: Działanie programu dla $l=8$, $r=2$ i 224 (2c) detektorów

| Dzień | Wykrytych anomalii | Niewykrytych anomalii | Falszywych alarmów |
|--------------|---------------------------|------------------------------|---------------------------|
| 04.12 | 15 | 5 | 6 |
| 11.12 | 57 | 18 | 1 |
| 12.12 | 95 | 30 | 1 |
| 13.12 | 38 | 12 | 1 |
| 14.12 | 20 | 5 | 17 |
| 15.12 | 19 | 6 | 3 |
| 16.12 | 5 | 14 | 7 |
| 17.12 | 18 | 8 | 6 |
| 21.12 | 40 | 0 | 4 |
| 26.12 | 120 | 0 | 0 |
| 27.12 | 80 | 0 | 8 |
| 28.12 | 40 | 165 | 1 |

Tabela 5: Działanie programu dla $l=8$, $r=2$ i 448 (4c) detektorów

6.3.5 Raporty z analizy dla długości słowa $l=10$ i stałej dopasowania $r=4$

| Dzień | Wykrytych anomalii | Niewykrytych anomalii | Falszywych alarmów |
|--------------|---------------------------|------------------------------|---------------------------|
| 04.12 | 9 | 11 | 2 |
| 11.12 | 33 | 42 | 2 |
| 12.12 | 55 | 70 | 1 |
| 13.12 | 22 | 28 | 1 |
| 14.12 | 10 | 15 | 13 |
| 15.12 | 10 | 15 | 1 |

| Dzień | Wykrytych anomalii | Niewykrytych anomalii | Falszywych alarmów |
|--------------|---------------------------|------------------------------|---------------------------|
| 16.12 | 3 | 16 | 1 |
| 17.12 | 10 | 16 | 2 |
| 21.12 | 40 | 0 | 1 |
| 26.12 | 120 | 0 | 0 |
| 27.12 | 80 | 0 | 7 |
| 28.12 | 8 | 197 | 0 |

Tabela 6: Działanie programu dla $l=10$, $r=2$ i 140 (c) detektorów

| Dzień | Wykrytych anomalii | Niewykrytych anomalii | Falszywych alarmów |
|--------------|---------------------------|------------------------------|---------------------------|
| 04.12 | 12 | 8 | 22 |
| 11.12 | 48 | 27 | 1 |
| 12.12 | 80 | 45 | 1 |
| 13.12 | 32 | 18 | 1 |
| 14.12 | 16 | 9 | 20 |
| 15.12 | 16 | 9 | 0 |
| 16.12 | 7 | 12 | 7 |
| 17.12 | 16 | 10 | 1 |
| 21.12 | 40 | 0 | 1 |
| 26.12 | 120 | 0 | 1 |
| 27.12 | 80 | 0 | 6 |
| 28.12 | 184 | 21 | 2 |

Tabela 7: Działanie programu dla $l=10$, $r=2$ i 280 (2c) detektorów

| Dzień | Wykrytych anomalii | Niewykrytych anomalii | Falszywych alarmów |
|--------------|---------------------------|------------------------------|---------------------------|
| 04.12 | 20 | 0 | 3 |
| 11.12 | 75 | 0 | 1 |
| 12.12 | 125 | 0 | 3 |
| 13.12 | 50 | 0 | 2 |
| 14.12 | 25 | 0 | 22 |
| 15.12 | 25 | 0 | 1 |
| 16.12 | 1 | 18 | 3 |
| 17.12 | 24 | 2 | 3 |
| 21.12 | 40 | 0 | 8 |
| 26.12 | 120 | 0 | 0 |
| 27.12 | 80 | 0 | 9 |
| 28.12 | 193 | 12 | 0 |

Tabela 8: Działanie programu dla $l=10$, $r=2$ i 560 (4c) detektorów

6.3.6 Analiza otrzymanych wyników

| Ilość detektorów | I=8 | I=10 |
|------------------|-----|------|
| <i>c</i> | 43% | 52% |
| <i>2c</i> | 59% | 73% |
| <i>4c</i> | 73% | 92% |

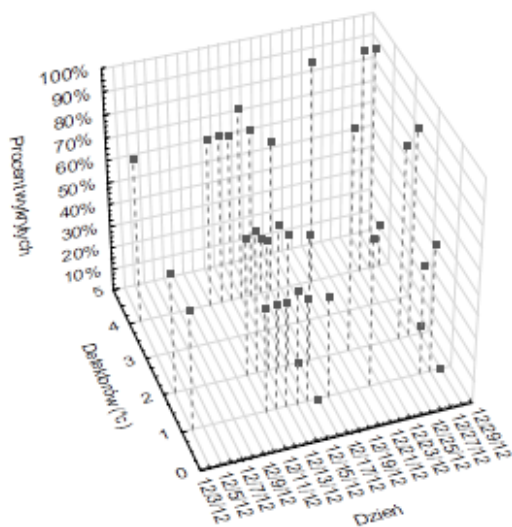
Tabela 9: Średnia skuteczność wykrywania anomalii

| Ilość detektorów | I=8 | I=10 |
|------------------|-------|-------|
| <i>c</i> | 1,54% | 1,88% |
| <i>2c</i> | 2,31% | 4,29% |
| <i>4c</i> | 3,86% | 3,38% |

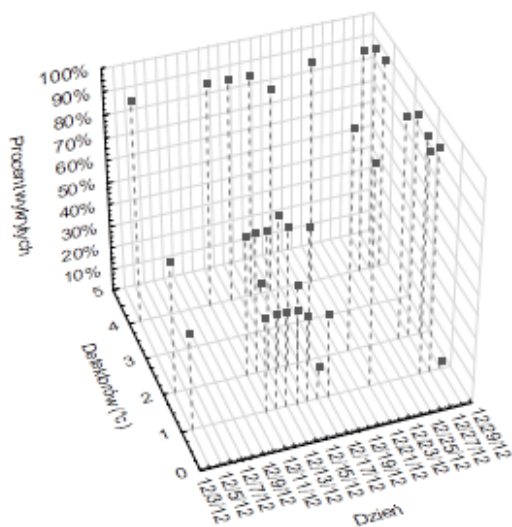
Tabela 10: Średni udział fałszywych alarmów we wszystkich analizowanych liniach

Jak pokazują powyższe tabele, już przy bardzo małej liczbie detektorów skuteczność programu jest na tyle duża, że może on z powodzeniem pełnić rolę „pierwszej linii obrony” administratora, tzn. wydawać krótki werdykt, czy plik z logami zawiera coś podejrzanego, czy nie. Przy wyższych liczbach detektorów narzędzie jest w stanie wykryć znaczną część anomalii, na które administrator powinien zareagować. Jeśli nawet nie są to wszystkie podejrzone wpisy, to z pewnością ukażą one główne trendy w zakresie zmian charakterystyki ruchu sieciowego w serwisie.

Zależność wykrywalności od pliku i liczby detektorów, skategoryzowana wzgl. długości podciągu



l: 8

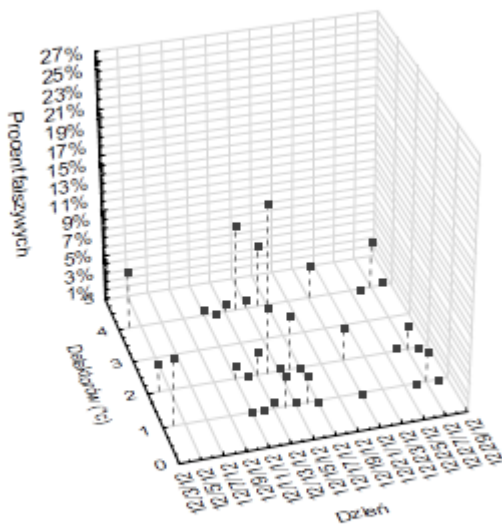


l: 10

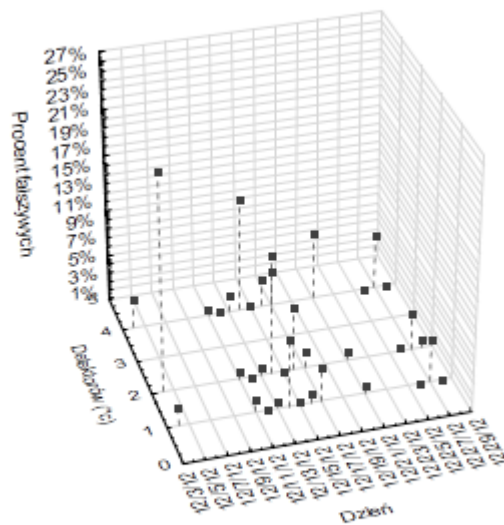
Rysunek 4: Charakterystyka wykrywalności

Jak ukazano na powyższych wykresach, wykrywalność zagrożeń zwiększa się w miarę zwiększania liczby detektorów, niezależnie od używanej długości podciągu. Wzrost nie następuje jednak liniowo, lecz zależny jest od charakterystyki konkretnego pliku, który poddawany jest analizie. Od charakterystyki pliku zależy również maksymalna i minimalna wykrywalność, jaką udało się osiągnąć z wykorzystaniem programu. Dla niektórych plików, jak np. dziennika z dnia 26.12, udało się osiągnąć dobry wynik już przy małej ilości detektorów. Dla innych zaś dla zbliżonego efektu należało tę ilość zwiększyć minimum dwukrotnie.

Zależność liczby fałszywych alarmów od pliku i ilości detektorów wzgl. długości podciągów



I: 8



I: 10

Rysunek 5: Charakterystyka fałszywych alarmów

Ilość fałszywych alarmów również rośnie w miarę zwiększania ilości detektorów, choć w tym przypadku wykazuje znacznie większy poziom losowości. Zdarzają się duże ilości błędnych wskazań przy małej ilości detektorów (c) oraz zerowe przy dużej ($4c$). Ogólny ich udział oscyluje zwykle w granicach kilku procent, tylko cztery razy przekraczając granicę 10%. Charakterystyczne jest zwiększenie się ilości fałszywych alarmów po zwiększeniu długości podciągu, a zachowaniu takiej samej stałej dopasowania r . Podczas eksperymentów stwierdzono także, że z uwagi na losowy charakter procesu generowania detektorów, ilość fałszywych alarmów może zmieniać się z każdym wygenerowanym ich zbiorem.

Mała liczba błędnie rozpoznanych anomalii jest bardzo dobrą cechą programu. Gdyby było ich dużo, to ciągłe informowanie o nich administratora mogłoby skończyć się uspienieniem jego czujności. Tak jednak nie jest i powinny one co najwyżej skłaniać go do rozszerzenia zbioru danych bazowych dla algorytmu przed rozpoczęciem dalszej analizy.

Kilku słów komentarza wymagają te wpisy, które kilkakrotnie były nieprawidłowo zgłaszane przez program. W większości przypadków były to raporty o działalności botów wyszukiwarek, m.in. Yandex oraz Googlebot-Images, które nie pojawiły się w zbiorze danych

bazowych. Kilka błędnych zgłoszeń spowodowało także przeglądanie serwisu na urządzeniach mobilnych, m.in. firm Apple i Nokia. Tego rodzaju alarmy mogłyby zostać wyeliminowane, gdyby zbiór danych bazowych obejmował dłuższy przedział czasowy.

7. Podsumowanie i wnioski

Otrzymane wyniki potwierdzają, że wykorzystanie algorytmu negatywnej selekcji do analizy logów serwera Apache nie tylko jest możliwe, ale także zapewnia wysoką skuteczność detekcji zagrożeń. Warto jednak przy tym podkreślić istotność dokładnej, ręcznej obróbki logów w początkowej fazie. Im więcej bowiem rekordów przedstawiających poprawną pracę zabezpieczanego serwisu uda się zebrać w charakterze danych bazowych dla programu, tym mniej fałszywych alarmów zgłaszać on będzie w analizowanych plikach podczas normalnej pracy.

Wyniki analiz generowane przez program udowadniają również, że możliwe jest odwoływanie się do technik stosowanych przez organizm człowieka w zagadnieniach bezpieczeństwa systemów.

Utworzony program można dodatkowo ulepszyć na kilka sposobów. Przede wszystkim można głębiej pochylić się nad zagadnieniem generowania detektorów i odejść nieco od obecnie stosowanej losowości. Dobrym pomysłem wydaje się być jakiś rodzaj połączenia z niektórymi elementami algorytmu selekcji klonalnej. Przykładowo zbiór detektorów mógłby „ewoluować”, rozwijając najskuteczniejsze elementy, a marginalizując te, które zgłaszają najwięcej fałszywych alarmów. Mógłby on także być rozszerzany o wygenerowane na innych systemach detektory, które skutecznie wychwytyją typowe ataki. Można wyobrazić sobie system rozproszony, w którym stacje robocze generują raporty o najskuteczniejszych detektorach i testują, czy ich zastosowanie na innej maszynie przynosi zwiększenie skuteczności, czy też przeciwnie – zwiększa ilość fałszywych alarmów lub wręcz nie spełnia kryterium niezgodności z danymi własnymi. Możliwości są potencjalnie nieograniczone.

Możliwe jest również ulepszenie interfejsu programu tak, by po początkowej fazie dostosowywania parametrów przechodził on w tryb automatyczny i bez ingerencji użytkownika analizował wszystkie napływające raporty. Administrator mógłby być jedynie powiadamiany e-mailowo o nieprawidłowościach. Mógłby także mieć możliwość odpowiedniego oznaczania pominiętych anomalii oraz fałszywych alarmów, przykładowo poprzez okresową, wrywkową, ręczną analizę logów.

8. Spis tabel

| | |
|--|----|
| Tabela 1: Opis kompetencji klas programu..... | 23 |
| Tabela 2: Charakterystyka danych testowych..... | 26 |
| Tabela 3: Działanie programu dla $l=8$, $r=2$ i 112 (c) detektorów..... | 28 |
| Tabela 4: Działanie programu dla $l=8$, $r=2$ i 224 (2c) detektorów..... | 29 |
| Tabela 5: Działanie programu dla $l=8$, $r=2$ i 448 (4c) detektorów..... | 29 |
| Tabela 6: Działanie programu dla $l=10$, $r=2$ i 140 (c) detektorów..... | 30 |
| Tabela 7: Działanie programu dla $l=10$, $r=2$ i 280 (2c) detektorów..... | 30 |
| Tabela 8: Działanie programu dla $l=10$, $r=2$ i 560 (4c) detektorów..... | 30 |
| Tabela 9: Średnia skuteczność wykrywania anomalii..... | 31 |
| Tabela 10: Średni udział fałszywych alarmów we wszystkich analizowanych liniach..... | 31 |

9. Spis rysunków

| | |
|--|----|
| Rysunek 1: Algorytm generowania detektorów..... | 15 |
| Rysunek 2: Sprawdzanie pliku z danymi na zgodność z danymi bazowymi..... | 16 |
| Rysunek 3: Uprozczone zależności między klasami programu..... | 18 |
| Rysunek 4: Charakterystyka wykrywalności..... | 32 |
| Rysunek 5: Charakterystyka fałszywych alarmów..... | 33 |

10. Bibliografia

- [1] Świtalska, Anna: Sztuczne systemy immunologiczne - zastosowanie w optymalizacji kombinatorycznej. <http://www.ipipan.waw.pl/~stw/ais/ks/index.html> [dostęp 02.01.2012r.].
- [2] Forrest, Stephanie i in.: Self-nonsel self discrimination in a computer. W: Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy, Los Alamitos, CA: IEEE Computer Society Press, s. 202-212.
- [3] Forrest, Stephanie i in.: A Change-Detection Algorithm Inspired by the Immune System. W: IEEE Transactions on Software Engineering, 1995.
- [4] Mayer, Gene: Immunology - Chapter One: Innate (non-specific) Immunity. <http://pathmicro.med.sc.edu/ghaffar/innate.htm> [dostęp 04.01.2012r.]
- [5] <http://httpd.apache.org/docs/2.2/logs.html> [dostęp 20.12.2011r.]
- [6] Gamma, Erich i in.: Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku. Helion, 2010, s. 269-279. ISBN 978-83-246-2662-5.
- [7] Śmiałek, Michał: Zrozumieć UML 2.0 Metody modelowania obiektowego. Helion, 2005, s.255. ISBN: 83-7361-918-6
- [8] <http://searchsecurity.techtarget.com/tip/How-to-spot-attacks-through-Apache-Web-server-log-analysis> [dostęp 08.01.2012r.]

A. Dodatek 1: Zawartość płyty CD

Dołączony do niniejszej pracy dysk CD zawiera następujące dane:

| Plik/folder | Zawartość | Uwagi |
|---|---|--|
| Immunologiczny system detekcji zagrożeń w oparciu o dane z logów aktywności.pdf | Treść pracy w formie gotowego do druku pliku PDF. | |
| Immunologiczny system detekcji zagrożeń w oparciu o dane z logów aktywności.doc | Treść pracy w formie źródłowej, przygotowana w programie LibreOffice Writer 3.3.2 | |
| Program/ | Folder z instalatorem programu | Do działania niezbędny jest .NET Framework w wersji co najmniej 3.5. |
| Źródła/ | Folder ze źródłami programu | Przygotowane w programie Microsoft Visual Studio 2010. |
| Dane testowe/ | Folder z danymi testowymi | W formacie tekstowym. |
| Wyniki/ | Folder z wynikami analiz | Wygenerowane przez program pliki CSV |