INSTYTUT BADAŃ SYSTEMOWYCH
POLSKIEJ AKADEMII NAUK

ANALIZA SYSTEMOWA W FINANSACH I ZARZĄDZANIU

Wybrane problemy
Tom 4

Pod redakcją
Jerzego HOŁUBCA

Warszawa 2002

INSTYTUT BADAŃ SYSTEMOWYCH
POLSKIEJ AKADEMII NAUK

ANALIZA SYSTEMOWA W FINANSACH I ZARZĄDZANIU

Wybrane problemy
Tom 4

Pod redakcją
Jerzego HOŁUBCA

Warszawa 2002

Wykaz opiniodawców artykułów zamieszczonych w tomie:

doc. dr hab. Mieczysław KŁOPOTEK

prof. dr hab. Stanisław PIASECKI

prof. dr Elżbieta RAKUS-ANDERSON

prof. dr hab. Andrzej STRASZAK

doc. dr hab. Sławomir WIERZCHOŃ

dr Sławomir ZADROŻNY

Publikacja dofinansowana przez
Agencję Wydawniczo-Poligraficzną "ARGRAF", Warszawa

ISBN 83-85847-74-X

Wydawca: INSTYTUT BADAŃ SYSTEMOWYCH PAN
ul. Newelska 6 01-447 Warszawa

Redakcja: Dział Informacji Naukowej i Wydawnictw
Barbara Kotuszewska, Joanna Runowska, tel. 837-68-22
Druk: Agencja Wydawniczo-Poligraficzna "ARGRAF", Warszawa
Nakład 200 egz., 15 ark.wyd.; 12,8 .ark. druk.

PRZYKŁADY ZASTOSOWANIA DETEKCJI ANOMALII DO ZABEZPIECZANIA SYSTEMÓW INFORMATYCZNYCH

Przemysław Ogonowski
Zaoczne Studia Doktoranckie IBS PAN

Jednym z większych wyzwań w informatyce jest problem bezpieczeństwa systemów komputerowych. W artykule tym opisany jest systemy bezpieczeństwa oparty na teorii działania układu odpornościowego człowieka (AIS – Artificail Immune System). Przedstawione w artykule teorie poparte zostały badaniami laboratoryjnymi, których wyniki zostały załączone w celu przedstawienia skuteczności nowej metody.

1. Wstęp

Obecne podejście do informatyki jest podejściem starym opartym na założeniach wywodzących się z lat 70-tych ubiegłego stulecia. Powstały one w czasach gdy najsilniejsze komputery miały moc obliczeniową dużo mniejszą od obecnych kalkulatorów. Niewiele więcej też od nich wymagano. Tak postawione wymagania pociągnęły za sobą metodologie tworzenia wczesnych systemów operacyjnych i aplikacji. Metodologia ta okazała się na tyle dobra, że przez wiele lat nikt nie zastanawiał się czy jest to dobry kierunek rozwoju.

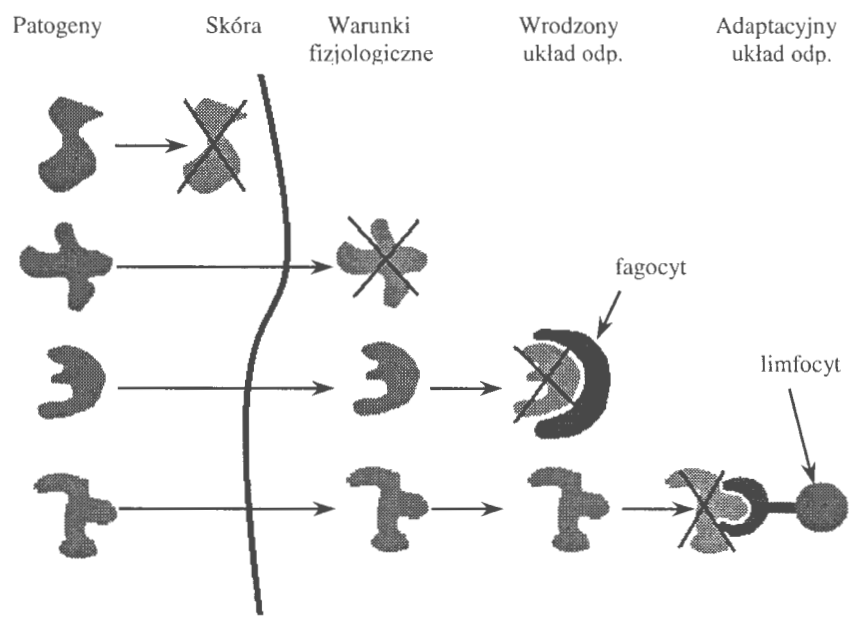
Obecny poziom komplikacji systemów i aplikacji sprawia, że dotychczasowe podejście okazuje się niewystarczające. Dlatego też grupa naukowców z *The University of New Mexico* rozpoczęła prace nad zastosowaniem mechanizmów wypracowanych przez naturę w wyniku milionów lat doświadczeń do usprawnienia pracy systemów informatycznych. Do jednej z pierwszych pełniejszych prac z tej dziedziny należy [1]. W opracowaniu tym autorzy opisują możliwość zastosowania teorii opartej na układzie odpornościowym człowieka do zabezpieczenie systemów informatycznych przed wirusami. Artykuł zawiera teoretyczne podstawy, opis algorytmu wykrywania anomalii (zachowań odbiegających od uznanych za prawidłowe dla danego systemu informatycznego) oraz matematyczne podstawy (łączenie z analizą prawdopodobieństwa wykrycia wirusa tą metodą). Praca ta praktycznie zapoczątkowała rozwój kierunku wykrywania anomalii (Anomaly detection) w systemach informatycznych,

opartego na pomyśle wykorzystania mechanizmów układu odpornościowego człowieka.

2. Podstawy teoretyczne

Wszystkie informacje zawarte w tym rozdziale oparte są na opracowaniu [2].

W układzie odpornościowym człowieka można wyróżnić cztery poziomy zabezpieczające: skóra, warunki fizjologiczne, wrodzony układ odpornościowy oraz adaptacyjny układ odpornościowy. Każdy z tych poziomów zabezpiecza organizm ludzki w inny sposób.



Rys. 1. Poziomy zabezpieczeń w układzie immunologicznym człowieka.

Skóra jest pierwszym poziomem zabezpieczającym i pełni bardzo ważną rolę. Oddziela ona organizm człowieka od zewnętrznych zagrożeń. Jest to poziom zabezpieczeń statycznych. Oznacza to, że w momencie próby przełamania tego systemu (przejścia wirusa przez skórę) nie wykona on żadnych dodatkowych akcji utrudniających atak. W systemach informatycznych rolę skóry pełnią systemy wejścia/wyjścia. Żaden atak na system informatyczny nie powiedzie się, jeżeli napastnik nie będzie miał

jakiegokolwiek dostępu do atakowanego systemu oraz stosował się do protokołów obowiązujących w danym systemie. Przykładem może być dowolny atak na komputer z wykorzystaniem sieci komputerowych. Bez znajomości odpowiednich protokołów sieciowych nie jesteśmy w stanie skontaktować się z innym systemem, a co dopiero włamać się do niego.

Drugim poziomem zabezpieczeń jest poziom warunków fizjologicznych. Przykładem takich warunków jest temperatura organizmu. Wirus aby mógł zaatakować musi być tak zbudowany aby dobrze rozwijał się w temperaturze ok. 36,6 st. cel. W systemach informatycznych za warunki fizjologiczne możemy uznać platformę, typ i wersję systemu operacyjnego. Jak wiadomo, żaden program napisany na system Windows nie będzie pracował na systemach UNIX. To samo odnosi się do wirusów komputerowych. Innym przykładem zabezpieczenia fizjologicznego jest wybór klienta pocztowego. Jeżeli wybrany klient nie wspiera danego języka skryptowego to napisany w tym języku skrypt wirusa nie zostanie uruchomiony.

Trzecim poziomem zabezpieczeń jest poziom wrodzonego układu odpornościowego. Poziom ten działa w ten sposób, że specjalizowane komórki sprawdzają wszystkie podejrzane komórki w organizmie i rozpoznają czy są to komórki własne czy obce. Następuje rozpoznanie typu: swój-obcy. Szerszy opis tego poziomu zabezpieczeń znajduje się w rozdziale trzecim.

Czwarty i ostatni poziom jest to poziom adaptacyjnego układu odpornościowego. Zasada działania polega na tym, że układ ten uczy się struktury ciał obcych w momencie ataku. W wyniku tego układ odpornościowy przy ponownym ataku szybciej rozpoznaje napastnika i wie jak z nim walczyć. Dlatego ponowne ataki często nie są nawet zauważane (brak objawów takich jak gorączka, bóle, ...). Jest to poziom dzięki któremu działają szczepionki. Pewne próby wprowadzenia tego poziomu zabezpieczeń można zaobserwować w programach antywirusowych i zabezpieczających. Różnią się one od układu immunologicznego tym, że wiedzę o wirusach (atakach) dostarczamy do tych systemów z zewnątrz (nie mają one własnego mechanizmu samouczącego).

3. Teoria wykrywania anomalii (algorytm)

Najpełniejszy opis teorii wykrywania anomalii w informatyce można znaleźć w [5]. W [6] opisany jest algorytm działania systemu AIS (Artificial Immune System) polegający na odwzorowaniu zasad działania układu odpornościowego człowieka na system komputerowy.

Teoria zaproponowana przez Hofmeyr'a i Forrest zakłada odwzorowanie elementów układu odpornościowego w ciągi bitowe równej długości (pojedynczy ciąg bitowy reprezentuje patogen). Zbiór wszystkich możliwych ciągów danej długości oznaczamy przez U (Universe). Dodatkowo wyróżniamy jeszcze dwa zbiory: S (Self - swój) i N (Nonself - obcy). Zbiór S jest zbiorem wszystkich ciągów powstałych w wyniku analizy poprawnej pracy systemu. Zbiór N jest to pozostała część zbioru U ($N = U \setminus S$).

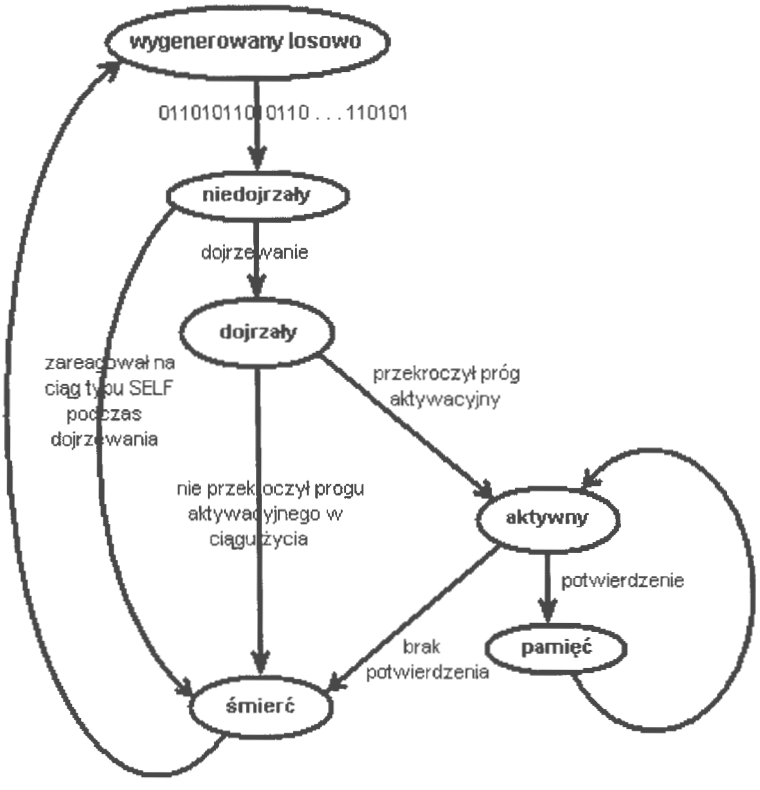
W proponowanej teorii wykrywanie nieprawidłowych zachowań systemu (ataki, wirusy, ...) polega na rozpoznawaniu czy w aktualnie wykonywanym kodzie nie występują obce ciągi. Wszystkie operacje wykonywane przez system układane są w ciąg kolejnych komend. Jest on potem dzielony na ciągi o stałej długości l. Każdy z tych ciągów jest potem analizowany pod względem przynależności do zbioru S. Wykrywanie (analiza) jest wykonywane przy pomocy tzw. detektorów.

W opracowaniu tym [6] zaproponowano regułę selekcji obcych ciągów (*matching rule*) polegającą na porównywaniu czy badany ciąg pokrywa się z detektorem na r-kolejnych pozycjach (*r-continues bit rule*). Jeżeli to nastąpi to uznajemy, że sprawdzany ciąg należy do zbioru obcych N. Aby system immunologiczny działał optymalnie musimy dobrać odpowiednie r tak aby uzyskać jak najmniejszą liczbę detektorów przy jak najwyższym stopniu wykrywania obcych ciągów.

Rysunek 2 pokazuje algorytm opisujący pracę systemu AIS zaproponowany w [6]. System pracuje w ten sposób, że najpierw generowany jest w sposób losowy detektor (ciąg o długości l, $l > r$). W tej fazie mówimy, że detektor jest niedojrzały. Ponieważ został on utworzony losowo może równie dobrze wykrywać ciągi ze zbioru N jak i z S. Oznacza to, że nie wiemy czy przy porównaniu z ciągiem ze zbioru S nie zareaguje tak jak przy ciągach ze zbioru N. Taki niedojrzały detektor przez jakiś czas jest obserwowany. Przy pomocy tego detektora wykonujemy porównania na wybranym zbiorze ciągów swoich, jednak jeżeli w czasie dojrzewania reguła selekcji da wynik pozytywny detektor taki jest odrzucany. Jeżeli w czasie dojrzewania detektor nie zareaguje na żaden ciąg uznajemy, że jest to detektor dojrzały. Zamykamy w ten sposób pierwszą fazę algorytmu.

Każdy z powstałych w ten sposób detektorów ma swój okres życia. Jeżeli w tym czasie nie zostanie aktywowany to umiera i jest zastępowany przez nowy, wygenerowany losowo. Aktywacja detektora polega na wykryciu obcych ciągów. Aktywacja następuje jednak dopiero po przekroczeniu minimalnej ilości (progu aktywacyjnego) dla danego

detektora (np. 10). Jeżeli próg ten zostanie przekroczona następuje aktywacja.



Rys. 2. Algorytm działania systemu zabezpieczającego opartego o założenia układu odpornościowego człowieka.

Niestety pomimo etapu dojrzewania nie mamy pewności, że tak wygenerowany i aktywowany detektor nie spowoduje fałszywego alarmu (nie zareaguje na ciąg ze zbioru S). Dzieje się tak, ponieważ nie wszystkie ciągi są używane podczas etapu dojrzewania. Dlatego też dla prawidłowej pracy systemu potrzebna jest korelacja informacji od detektora z innym źródłem. Mówimy wtedy, że detektor wykrył ciąg obcy, jeżeli zostało to potwierdzone przez inny sygnał. W układzie odpornościowym człowieka taką informacją może być fakt zaatakowania komórek. W opisywanym algorytmie Hofmeyr i Forrest nie zaproponowali automatycznego systemu potwierdzającego, a jedynie interakcję z operatorem, którego zadaniem jest

potwierdzenie, czy to ma być uznane za atak czy nie. Przy braku potwierdzenia uznajemy, że detektor wykrył ciąg swój a w związku z tym kasujemy go i zastępujemy nowym wygenerowanym losowo. W momencie potwierdzenia detektor zostaje zapisany do pamięci systemu. Detektor zapisany do pamięci ma dwie cechy które odróżniają go od zwykłych: jego czas życia jest nieograniczony oraz próg aktywacyjny jest zmniejszony (np. z 10 na 1). Jeżeli pamięć jest zapełniona, to ostatni z listy (najdawniej utworzony) jest usuwany. Istnienie pamięci detektorów rozumianej w ten sposób daje możliwość zaimplementowania kolejnego mechanizmu z układu odpornościowego człowieka: mechanizmu adaptacyjnego. Tworzymy w ten sposób system zabezpieczający, który może się przystosowywać do nowych warunków (np. uczyć nowych aplikacji dodanych do systemu).

Ponieważ zaproponowany algorytm nie opisuje w jaki sposób system AIS miałby reagować w momencie wykrycia ataków lub wirusa, w dalszej części opisane zostaną przykłady możliwych zastosowań wraz z opisem sposobów reakcji systemu.

4. Modyfikacje i rozszerzenia algorytmu

Przedstawiony przez Hofmeyr i Forrest algorytm nazywać będziemy „Generuj i testuj". W tym rozdziale opisane zostaną kolejno algorytmy: czasu rzeczywistego, zachłanny, optymalny. Pierwsze dwa algorytmy zostały opisane przez D'haeseleer'a, Forrest i Helman'a w [3]. Trzeci algorytm został zaproponowany przez Wierzchonia w [8].

Algorytm czasu rzeczywistego nazywa się tak ponieważ jest to pierwszy algorytm, który mógł pracować w czasie rzeczywistym. Głównym założeniem przy tworzeniu było to, że podstawowy algorytm „Generuj i testuj" nie jest efektywny (większość detektorów jest odrzucana). Opisany algorytm jest dwufazowy. W pierwszej fazie tworzona jest tabela w której zapisane są informacje o ilości prawidłowych detektorów wygenerowanych dla wszystkich podciągów o długości r ustawiając te podciągi na kolejnych pozycjach ciągu badanego. W ten sposób otrzymujemy wyjściową tabelę, która będzie pomocna przy generacji detektorów. W drugiej fazie algorytmu na podstawie tabeli generujemy detektory. Najpierw liczymy liczbę detektorów – x (suma liczb z pierwszej kolumny tabeli). Następnie generujemy liczbę z zakresu [0, (x-1)] (np. 17). Na końcu na podstawie wygenerowanej liczby przechodzimy przez utworzoną w pierwszej fazie tabelę składając detektor odpowiadający wygenerowanej liczbie (w tabeli 1 przedstawiony jest przebieg generacji detektora; detektor nr 17 według załączonej tabeli ma postać 011011 i składa się z czterech receptorów: 011***, *110**, **101*, ***011). Ponieważ nie jest to generacja czysto

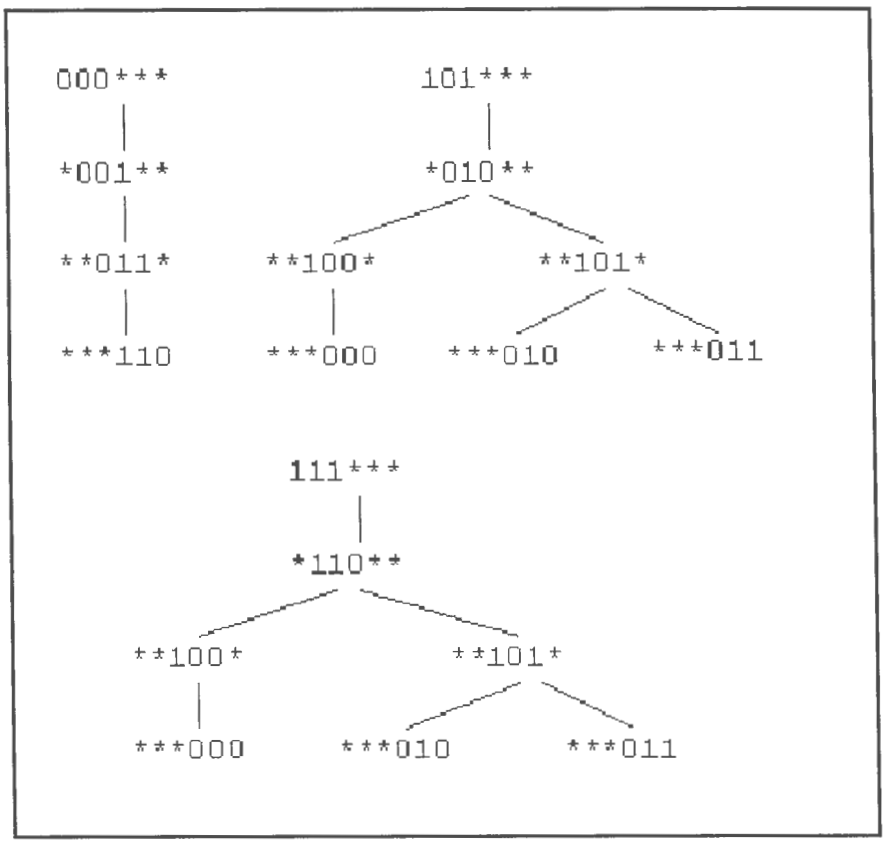
losowa, jest to algorytm dużo wydajniejszy a wszystkie detektory wygenerowane w fazie drugiej są detektorami dobrymi.

Kolejny algorytm opisany w [3] został przez autorów opisany jako „zachłanny" (Greedy Algorith). Założeniem tworzenia algorytmu była minimalizacja ilości generowanych detektorów przy zachowaniu tej samej skuteczności (ilości dziur – ciągów ze zbioru N nie wykrytych przez żaden detektor) co w algorytmie czasu rzeczywistego. Algorytm ten został zresztą zbudowany na bazie algorytmu czasu rzeczywistego. Główna różnica polega na takim doborze detektorów aby były oddalone od siebie jak najdalej. Detektor jest wybierany w taki sposób aby pokrywał jak najwięcej niepokrytych przez inne detektory ciągów ze zbioru N. W ten sposób wykluczamy sytuację w której ten sam ciąg jest wykrywany przez kilka detektorów. Prowadzi to do minimalizacji liczby generowanych detektorów oraz do maksymalnego rozproszenia detektorów. Algorytm w pierwszej fazie oprócz generacji tablicy na podstawie zbioru S, generuje drugą na podstawie aktualnych detektorów. Przez porównanie wyników z obu tablic uzyskujemy zawsze detektor optymalny. Pomimo postawienia dużej uwagi na uzyskanie algorytmu wydajnego, który mógłby pracować w rzeczywistych systemach komputerowych, zaproponowane rozwiązanie przestaje być wystarczająco wydajne dla dużych wielkości ciągów.

s	zakres	$C_3[s]$	$C_2[s]$	$C_1[s]$	$C_0[s]$
000	0-3	4	4	2	1
001	4-9	6	0	2	1
010	10-13	4	4	0	1
011	14-19	6	2	2	1
100	-	0	4	2	0
101	20-25	6	0	2	0
110	-	0	4	0	1
111	26-31	6	2	2	1

Tabela 1. Przykład tabeli utworzonej w algorytmie czasu rzeczywistego [3]

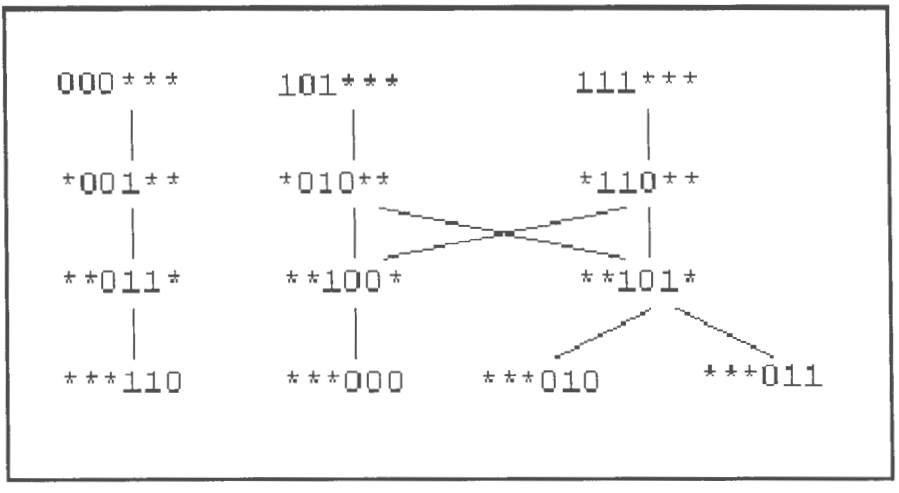
Trzecią modyfikację mającą na celu optymalizację oryginalnego algorytmu opisaną w tej pracy zaproponował Wierzchoń w [8]. W pierwszej kolejności autor zaproponował z wygenerowanych detektorów utworzyć drzewo.



Rys. 3. Przykład drzewa detektorów [8].

Idąc po gałęziach tego drzewa od góry do dołu widzimy, że w zaproponowanym przez autora przykładzie zostało wygenerowanych siedem detektorów (siedem zakończeń drzewa na dole). Następnie autor zauważył, że powyższe drzewo można połączyć (rys. 4). I teraz, idąc od góry do dołu po połączeniu zostają nam tylko cztery detektory. Powoduje to zmniejszenie wielkości pamięci detektorów oraz przyśpieszenie procesu porównywania badanych ciągów z detektorami. Zmniejszenie ilości detektorów uzyskujemy ponieważ detektory (z początkowego zbioru) pokrywają się w pewnych obszarach. I tak, para detektorów 101000 i 101010 pokrywa się na pierwszych czterech pozycjach, para detektorów 111000 i 111010 również

pokrywa się na pierwszych czterech pozycjach. Jeżeli jednak weźmiemy pierwszy detektor z pierwszej pary i pierwszy z drugiej to zauważamy, że pokrywają się one na czterech ostatnich pozycjach (to samo ma miejsce z drugimi detektorami z obu par). Wynika z tego, że aby wykryć wszystkie ciągi początkowo wykrywane przez cztery detektory wystarczy nam para pierwszy detektor z pierwszej pary i drugi z drugiej (lub odwrotnie).



Rys. 4. Przykład połączonego drzewa receptorów [8].

Następnie autor zaproponował modyfikację warunków wejściowych. Ponieważ część receptorów (receptor jest to ciąg o długości r będący częścią detektora, np. *001**) jest nieefektywna, co znaczy, że ponieważ ich rodzice (receptory uzupełnione z lewej strony o „0" lub „1" z odłączonym ostatnim elementem – np. 000*** i 100***) i dzieci (receptory z odłączonym pierwszym elementem uzupełnione z prawej strony o „0" lub „1") wykrywają ciągi ze zbioru S nie można przy ich pomocy przygotować efektywnych detektorów, należy potraktować je jak receptory wykrywające ciągi swoje. Przy takim założeniu uzyskujemy jeszcze mniejszą liczbę detektorów. Mogą jednak pojawić się dodatkowe ciągi obce nie rozpoznawane przez żaden detektor. W przykładach autor udowodnił, że liczba dodatkowych nierozpoznanych ciągów obcych nie jest duża, a przy wzroście długości ciągów l i receptorów r szybko spada do zera. Opis doświadczeń wraz z wynikami znajduje się w [8, 9]. Podsumowując, przy długości ciągu $l = 10$ liczba nie rozpoznawanych ciągów spada do zera dopiero przy receptorach o wielkości $r = 10$; przy długości ciągów równej $l = 12$ liczba nie rozpoznanych ciągów spada do zera przy receptorach

o długości $r = 9$; a przy długości ciągów równej $l = 20$ liczba nie rozpoznanych ciągów spada do zera przy receptorach o długości $r = 14$.

5. Przykłady zastosowania

W tej części przedstawione zostaną przykłady zastosowania opisanej powyżej idei w zabezpieczaniu systemów informatycznych. Są to przykłady laboratoryjne i wskazują na duże możliwości zastosowania w systemach rzeczywistych. Wymaga to jednak dopracowania algorytmów. Przedstawione zostaną kolejno trzy grupy zastosowań: systemy antywirusowe, monitorowanie włamań w systemach sieciowych i monitorowanie włamań w systemie UNIX.

W [1] przestawiona jest analiza możliwości użycia teorii AIS do monitorowania i zabezpieczania systemów informatycznych przed wirusami. Najpierw przestawiono wyniki testów dla ciągów generowanych losowo. Potem przedstawiono opis modyfikacji pliku wygenerowanego na podstawie kodu C kompilowanego na platformie SPARC. Testy wykonano w ten sposób, że na początku przygotowano zbiór receptorów przy użyciu pliku utworzonego przez zdekodowanie skompilowanego pliku na zbiór kodów procesora SPARC (każdą komendę systemową przedstawiono w postaci pojedynczego znaku). Następnie wykonano jedną z trzech modyfikacji: drobna zmiana w kodzie źródłowym i rekompilacja; zmiana jednego kodu w skompilowanym i zdekodowanym pliku; dopisanie 24 dodatkowych kodów na końcu segmentu kodu w zdekodowanym pliku. Ostatnia metoda jest najbliższa rzeczywistego zarażenia pliku wirusem. Przy tej metodzie modyfikacji (zarażenia) przy tylko 8 receptorach uzyskano 82% prawdopodobieństwa wykrycia wirusa; 100% uzyskano już przy 32 receptorach (każdy po 32 znaki). Na końcu przedstawiono analizę próby wykrycia rzeczywistego wirusa TIMID na plikach COM systemu DOS 5.0. Wirus działa w ten sposób, że modyfikuje pierwsze pięć bajtów i dopisuje 300 na końcu pliku. Testy wykonano dla różnych wielkości r przy długości ciągów równej 32 znaków. We wszystkich testach udało się uzyskać prawdopodobieństwo wykrycia wirusa równe 100%.

Z analizy tabeli wynika, że (dla wykrycia wirusa TIMID) bardziej opłaca się stosować mniejsze r ponieważ stuprocentowe prawdopodobieństwo wykrycia wirusa otrzymujemy przy mniejszej liczbie detektorów. Zgodnie z ogólną teorią AIS minimalizacja ilości detektorów może powodować powstanie tak zwanych dziur (obcych ciągów które nie mogą być wykryte przez żaden detektor), jednak teoria zakłada równomierny losowy rozkład ciągów, a w rzeczywistych przykładach mamy do czynienia z ograniczeniami tej równomierności. Ponieważ jest to kod

(napisany przez programistę o pewnych nawykach; w języku programowania, który narzuca szablony pisania) i jest on skompilowany przy pomocy konkretnego kompilatora (każdy kompilator ma wbudowany wewnętrzne szablony) nie możemy mówić tutaj o losowości wyboru ciągów do zbioru *S*. Prawdopodobnie większość ciągów jest zgrupowana w paru obszarach co sprzyja skutecznemu wykrywaniu przy małych wielkościach *r*.

Test	r	N_R	P_f
SPARC – modyfikacja źródła	1	2	0,26
SPARC – pojedyncza modyfikacja	1	100	0,24
SPARC – modyfikacja segmentu	1	32	0,00
DOS – TIMID	9	10	0,00
DOS – TIMID	10	25	0,00
DOS – TIMID	13	125	0,00

Tabela 2. Porównanie zastosowania AIS do wykrywania wirusów. We wszystkich testach długość ciągów wynosiła 32 znaki.

Drugim obszarem przebadanym w testach jest monitorowanie ataków na systemy informatyczne poprzez sieci komputerowe. Pierwsze testy opisane są w [6]. W [11] znajduje się uzupełniona i rozszerzona wersja analizy. W celu przeprowadzenia testów przyjęto następujące założenia: testy zostały przeprowadzone na rzeczywistych danych ale w trybie off-line, długości ciągów wynosiła 49 bitów (składała się z adresu IP źródłowego i docelowego oraz numeru portu docelowego), $r = 12$, każdy z monitorowanych komputerów będzie miał 100 detektorów, testy zostaną wykonane na sieci składającej się z 50 komputerów. Warto zauważyć że testy przeprowadzono z pominięciem analizy zawartości pakietów sieciowych. Dodanie tego poziomu może znacząco zwiększyć skuteczność i uniwersalność metody. Przy takich założeniach wykonano zebranie dwóch zbiorów danych: podczas normalnej pracy środowiska i podczas ataków. W wyniku generacji detektorów stworzono system który w etapie testowania detektorów wykazywał 74 fałszywe ataki (alarmował pomimo braku ataku). Dodanie warunku aktywacji przy dziesięciu wystąpieniach spowodowało spadek fałszywych alarmów do 8 dziennie. Dodanie warunku potwierdzenia przez operatora/użytkownika spowodowało ostateczny spadek fałszywych alarmów do zera. Następnie wykonano testy polegające na próbach 8 różnych ataków. W podstawowej wersji system wykrył 7. Jednak przy nałożeniu na wszystkie detektory dedykowanych masek permutacji system wykrył wszystkie 8 ataków.

Trzecim i ostatnim przebadanym obszarem jest monitorowanie nieprawidłowego zachowania się procesów systemowych (na przykładzie

procesów UNIX'a). Szereg takich analiz można znaleźć w następujących publikacjach [4, 10, 12]. Najszerszy opis można znaleźć w [13]. Znajduje się tam porównanie skuteczności różnych algorytmów generacji detektorów przy zastosowaniu do monitorowania procesów systemu UNIX. W wyniku przeprowadzonych testów autorzy stwierdzili, że żadna metoda nie dała całkowitej pewności ze względu na to, że próby zwiększenia prawdopodobieństwa wykrycia zachowań nienormalnych powodowały wzrost pojawiających się błędów polegających na uznaniu ciągów własnych za obce. Dodatkowo autorzy stwierdzili, że więcej było zbieżności w wynikach pomiędzy testami dla tych samych ataków niż dla danego algorytmu. Ogólnie wyniki można uznać za bardzo dobre (ponad 90% skuteczności).

Przedstawione powyżej testy nie uwzględniały problemu co zrobić z wykrytą anomalią.

6. Reakcje na wykrycie anomalii w AIS

W [14] Somayaji i Forrest przedstawiają propozycję automatycznego reagowania na wykryte anomalie poprzez wstawianie opóźnień w ciągi odwołań systemowych które są uznane za obce. Uruchomienie systemu odpornościowego na testowanym serwerze spowodowało niewielki spadek wydajności systemu (do 5% [14]). Wprowadzenie akcji polegającej na opóźnieniu każdego odwołania systemowego spowodowało znaczne spowolnienie atakowanych procesów. Spowolnienie wynosiło od kilkudziesięciu sekund do dwóch godzin. Spowolnienia te nie zawsze doprowadzały do udaremnienia ataku, jednak trzeba cierpliwego włamywacza aby czekał na odpowiedź systemu do dwóch godzin. Autorzy dodatkowo wprowadzili w swoim systemie zabezpieczającym drugi typ reakcji polegający na zabiciu procesu w którym występują anomalie. Ten typ reakcji zapewnił stuprocentową skuteczność obrony, wywołał jednak dodatkowe niepożądane skutki (utratę procesów, które zostały zaatakowane).

7. Dyskusja

Prawdopodobnie dlatego, że teoria wykrywania anomalii w systemach informatycznych nie jest jeszcze dostatecznie rozwinięta problem reakcji systemu odpornościowego AIS na wykrycie anomalii jest rzadko poruszany. Problem ten nie jest trywialny ponieważ mamy narzucone pewne ograniczenia. I tak z jednej strony istnieje potrzeba wykrycia anomalii w przypadku ataku natychmiast. W przypadku wirusa możemy pozwolić sobie na niewielkie opóźnienia w reakcji. Z drugiej strony żaden atak albo wirus

nie jest w stanie działać tylko na ciągach ze zbioru S. Problemem jest ustalenie granicznej ilości ciągów obcych przy której stwierdzamy, że jest to zachowanie systemu nieakceptowane przez nas. Musimy tu również mieć na uwadze to, że poziomy aktywacji będą różne w zależności od typu ataku jak i również od serwera (różne zbiory S). Kolejnym problemem jest określenie rodzaju reakcji. Z jednej strony mamy ataki polegające np. na kradzieży danych. Wymagają one natychmiastowej i drastycznej reakcji (np. zabicie procesu). Z drugiej strony mamy ataki, które mogą wprawdzie doprowadzić do unieruchomienia systemu, jednak wystarczająco dobrze można poradzić sobie z nimi poprzez wstawienie opóźnienia [14] w procesie w którym wykryto anomalię. Kolejnym ważnym problemem przed jakim stajemy jest kwestia przystosowywania się systemu odpornościowego AIS do akceptowalnych zmian w zachowaniu monitorowanego systemu (np. dogranie nowego sterownika po podłączeniu urządzenia do serwera, instalacja dodatkowego oprogramowania).

Z jednej strony chcemy mieć system który będzie wykrywał 100% anomalii, a z drugiej strony nie chcemy aby serwer który jest monitorowany przez nasz system przy każdej drobnej zmianie musiał być przenoszony do środowiska laboratoryjnego w celu przygotowania zbioru ciągów opisujących zachowanie normalne serwera, lub reagował na normalną pracę generując alarmy.

8. Wnioski

Teoretyczne możliwości wykrywania anomalii są bardzo duże. Zastosowanie tej teorii, jak również próba przeniesienia innych aspektów zapożyczonych z układu odpornościowego człowieka (opisanych w [15]) mogłoby całkowicie zmienić sposób postrzegania bezpieczeństwa systemów informatycznych. Przygotowanie skutecznego wirusa lub ataku wymagałoby dogłębnej znajomości systemu (konkretnego serwera) jak i jego konfiguracji zabezpieczeń (zbiór S). Oznacza to konieczność uzyskania autoryzowanego dostępu do systemu w celu zdobycia tych informacji. Przy uwzględnieniu ciągłej zmienności zbioru S dodatkowym utrudnieniem jest warunek szybkiego przygotowania ataku lub wirusa.

Literatura

[1] Stephanie Forrest, Alan S. Perelsen, Laurence Allen, Rajesh Cherukuri (1994): Self-Nonself Discrimination in a Computer. *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy.* IEEE Computer Society Press.

[2] Steven A. Hofmeyr (2000): An Interpretative Introduction to the Immune System. *Design Principles for the Immune System and other Distributed Autonomous Systems*. Oxford University Press, Eds, I. Cohen and L. Segel.

[3] Patrik D'haeseleer, Stephanie Forrest, Paul Helman (1997): A Distributed Approach to Anomaly Detection. *ACM Transactions on Information System Security*.

[4] Steven A. Hofmeyr, Stephanie Forrest, Anil Somayaji (1998): Intrusion Detection using Sequences of System Calls. *Journal of Computer Security*.

[5] Patrik D'haeseleer (1996): An Immunological Approach to Change Detection: Theoretical Results. *Proceedings of the 9th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press.

[6] Steven A. Hofmeyr, Stephanie Forrest (2000): Intrusion Detection: Architecture for an Artificial Immune System. *Evolutionary Computation Journal*.

[7] P. D'haeseleer, S. Forrest, P. Helman, (1996): An immunological approach to change detection: algorithms, analysis and implementations. *Proceedings of the 1996 IEEE Symposium on Computer Security and Privacy. IEEE Press.*

[8] S.T. Wierzchoń (2000): Generating optimal repertoire of antibody strings in an artificial immune system. M. Kłopotek, M. Michalewicz and S.T. Wierzchoń *Intelligent Information Systems*. Advances in Soft Computing Series of Physica-Verlag/Springer Verlag, Heidelberg/ New York 2000, Physica-Verlag, pp. 119-133

[9] S.T. Wierzchoń (2000): Discriminative power of the receptors activated by *k*-contiguous bits rule. *Journal of Computer Science and Technology. Special Issue on Research in Computer Science*, vol. 1, no. 3, 2000, pp. 1-13.

[10] Steven A. Hofmeyr, Stephanie Forrest, Anil Somayaji (July 1997): Lightweight Intrusion Detection for Networked Operating Systems. *Journal of Computer Security*.

[11] Steven A. Hofmeyr, Stephanie Forrest (1999):Immunity by Design: An Artificial Immune System. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO).*

[12] Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, Thomas A. Longstaff (1996): A Sense of Self for Unix Processes. *Proceedings of the 1996 IEEE Symposium on Computer Security and Privacy.* IEEE Press.

[13] Christina Warrender, Stephanie Forrest, Barak Pearlmutter, (1999): Detecting Intrusions Using System Calls: Alternative Data Models. *1999 IEEE Symposium on security and Privacy.*

[14] Anil Somayaji, Stephanie Forrest (2000): Automated Response Using System-Call Delays. *Usenix 2000.*

[15] Stephanie Forrest, Anil Somayaji, David H. Ackley, (1997):Building Diverse Computer Systems. *Proceedings of the 6th Workshop on Hot Topics in Operating systems.* IEEE Computer Society Press.

[16] S.T. Wierzchoń (2001): Sztuczne systemy immunologiczne. Teoria i zastosowania. *Akademicka Oficyna Wydawnicza EXIT,* Warszawa 2001

ISBN 83-85847-74-X