



Wyższa Szkoła Przedsiębiorczości i Administracji w Lublinie
Wydział Nauk Technicznych
Kierunek: Informatyka

System monitoringu serwerów „ServerMonitor”
(Server Monitoring System “ServerMonitor”)

Autor: Konrad Komada

Nr albumu: 6532

Promotor: dr Beata Pańczyk

Oświadczenie kierującego pracą.

Oświadczam, iż niniejsza praca została przygotowana pod moim kierunkiem i stwierdzam, że spełnia ona warunki do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

.....

(data)

.....

(podpis)

Oświadczenie autora pracy

Świadomy odpowiedzialności prawnej oświadczam, iż niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej podmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Ponadto oświadczam, iż niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

.....

(data)

.....

(podpis)

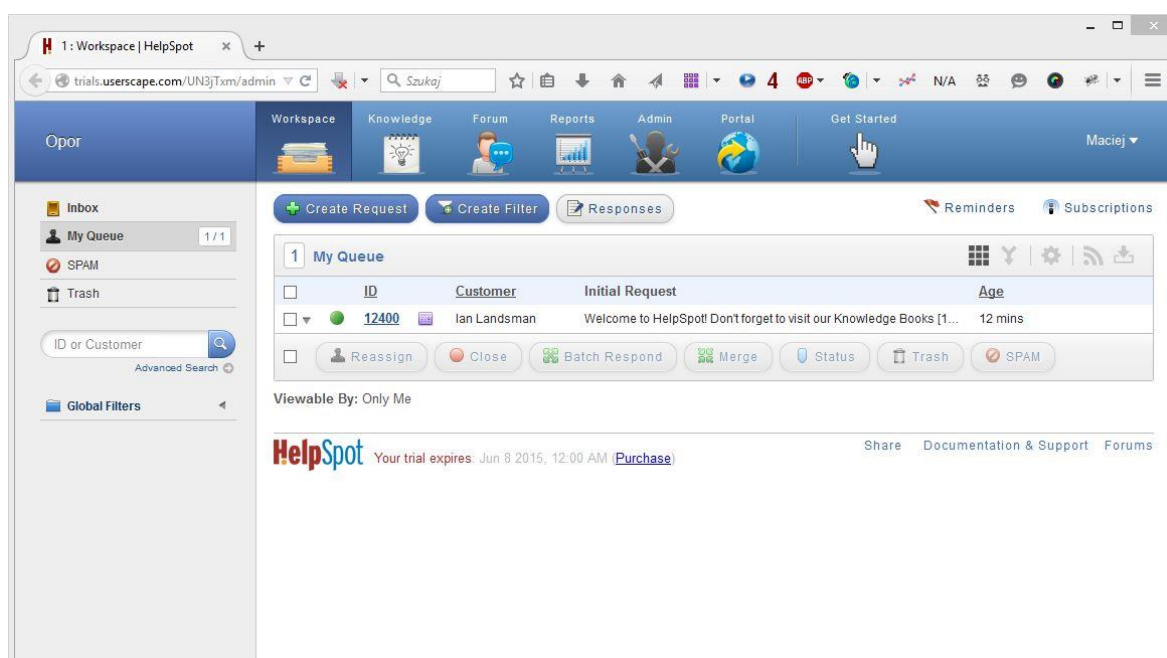
Spis treści

1.	Wstęp	3
2.	Technologie i narzędzia	15
3.	Projekt systemu.....	18
3.1.	Założenia systemu	18
3.2.	Specyfikacja wymagań funkcjonalnych.....	21
3.3.	Specyfikacja wymagań niefunkcjonalnych.....	21
3.4.	Projekt struktury systemu.....	22
3.4.1.	Część skryptowa	22
3.4.2.	Część aplikacyjna „ServerMonitor”	23
3.5.	Projekt interfejsu graficznego	23
3.5.1.	Interfejs	24
4.	Realizacja systemu.....	26
4.1.	Implementacja	26
4.2.	Testowanie programu	53
4.3.	Instrukcja instalacji programu	55
4.4.	Bezpieczeństwo systemu.....	55
5.	Wnioski.....	57
6.	Bibliografia	58
7.	Spis rysunków.....	60
8.	Spis listingów.....	61

1. Wstęp

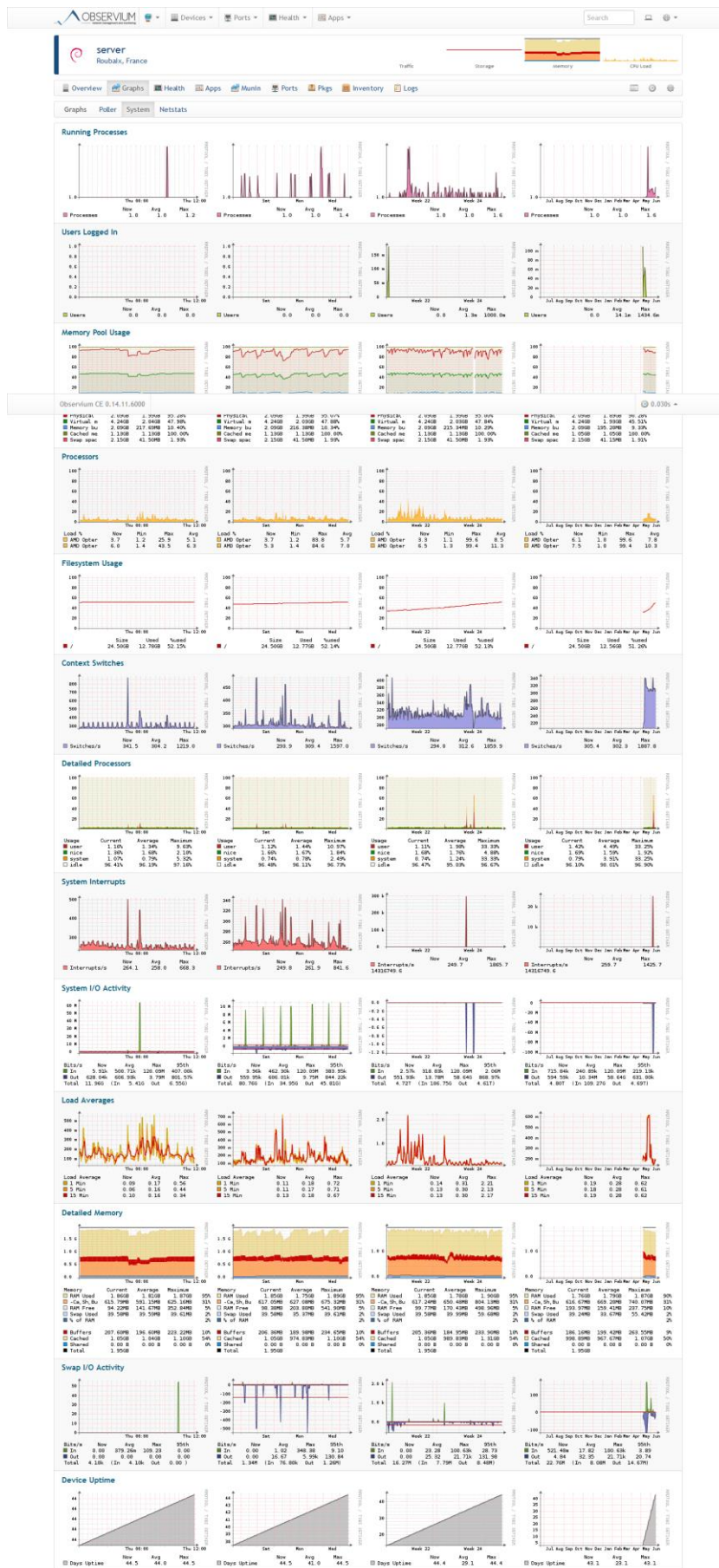
Temat dotyczący systemu wsparcia monitorowania i utrzymania serwerów jest bardzo szerokim zagadnieniem, obejmującym swoim zakresem wiele kwestii w skład, których mogłyby wejść:

- bazodanowe systemy wspierające zarządzanie infrastrukturą i oprogramowaniem w centrach przetwarzania danych,
- systemy obsługi i katalogowania zgłoszeń awarii i incydentów,
- systemy zbierające i przetwarzające informacje o stanie monitorowanych parametrów serwerów,
- systemy typu ticket dla centrów pomocy (helpdesk), itp. (Rysunek 1.1).



Rysunek 1.1 – system typu ticket – HelpSpot.

Jednakże powyższe przykłady, to przykłady oprogramowania integrującego pracę administratorów i pomagające zarządzać incydentami w sposób bardziej zorganizowany. Systemy tego typu pozwalają koordynować pracę administratorów, wydajnie kierować takim zespołem, mierzyć wydajność pracy administratorów, a co za tym idzie zapewnić optymalizację kosztów zatrudnienia i optymalizację kosztów przedsięwzięcia jakim jest centrum przetwarzania danych.



Rysunek 1.2 – system wykresów – Observium.

Dla samych administratorów oprogramowanie takie mimo, że jest bardzo przydatne a nawet niezbędne, to jednak samo w sobie nie zaspokaja ich najważniejszych potrzeb dotyczących szybkiego pozyskiwania informacji awariach, które właśnie miały miejsce, o potencjalnych awariach, zdarzeniach, które mogą pojawić się wkrótce, jeśli nie zostaną podjęte działania zaradcze, oraz o takich, które trwają niezauważone, i których skutki nie powodują oczywistych zmian na wykresach parametrów systemu (Rysunek 1.2). Bez względu na rodzaj awarii, wszystkie awarie należy niezwłocznie usunąć tuż po ich wykryciu, zanim powstaną większe szkody lub straty. Im wcześniej awaria zostanie zauważona, tym jej skutki będą mniejsze.

W związku z powyższym i z myślą o usprawnieniu pracy administratorów w obszarze wykrywania awarii, incydentów oraz zbierania informacji dotyczących zagrożeń, jak również w celu automatycznego powiadomienia administratora o tego typu sytuacjach, niniejsza praca będzie dotyczyła systemu, który w założeniu ma być odpowiedzią na nakreśloną potrzebę.

W celu zapewnienia prawidłowego działania serwera i oprogramowania na nim działającego, każdy serwer musi być monitorowany. Zakres monitorowania serwera uwarunkowany jest wieloma czynnikami. Do tych czynników należą:

- potrzeby klienta,
- polityka firmy,
- możliwości inwestycyjne,
- opłacalność,
- wiedza administratorów,
- ilość serwerów przypadająca na jednego administratora.

Ten ostatni punkt jest tutaj bardzo istotny. Jeden administrator może monitorować jednocześnie kilka dużych serwerów i wykonywać wszelkie prace związane z ich obsługą bezpośrednio. Robi to między innymi poprzez obserwację i analizę parametrów serwera rysowanych w czasie rzeczywistym na wykresach (Rysunek 1.2).

Wykresy parametrów serwera mogą przedstawiać różne dane w zależności od funkcji serwera jakie pełni. Najważniejsze i najczęściej spotykane to:

- Wykorzystanie łącza (transfer);
- Czas odpowiedzi na komunikaty kontrolne ICMP – ping (serwery dostępne, www, poczta, ftp, routery, proxy, zapory ogniowe – firewalls);
- Obciążenie dysku (macierze dyskowe, NFS);
- Ilość procesów na serwerze;

- Średnie obciążenie CPU – LOAD;
- Chwilowe obciążenie CPU mierzone w procentach;
- Ilość zajętej pamięci RAM w procentach;
- Ilość użytej pamięci wirtualnej SWAP (w megabajtach);
- Ilość procesów w stanie R (uruchomionych lub oczekujących na uruchomienie);
- Procent zajętości głównego systemu plików „/”;
- Procent zajętości innych zamontowanych systemów plików takich jak „/home”;
- Temperatury poszczególnych dysków (macierze dyskowe, NFS);
- Temperatura CPU (zbiorcza oraz dotycząca poszczególnych rdzeni/procesorów);

Istnieje możliwość monitorowania stanu kilku serwerów bezpośrednio (bez użycia dodatkowych systemów wsparcia). Do tego celu używa się standardowych narzędzi systemowych oraz korzysta się z interfejsów informacyjnych monitorowanych aplikacji np.: Apache – server-status (Rysunek 1.3). Można korzystać ze standardowych aplikacji dostępnych w środowiskach serwerowych Unix/Linux takich jak: top, htop (Rysunek 1.4), du, df, iostat, trafshow, iftop, dmesg, tmux, screen oraz poprzez monitoring dzienników systemowych – logów, parametrów systemów pozyskanych z „proc/procfs” (systemu plików procesów), „sys/sysfs” (wirtualny system plików dostarczony przez jądro Linux), „sysctl” (interfejs monitorowania i dokonywania zmian w parametrów pracy systemów uniksowych z rodziny BSD) .

Apache Server Status

```
Server Version: Apache/2.4.12 (Unix) OpenSSL/1.0.1e  
Server MPM: event  
Server Built: Mar 13 2015 17:44:41
```

```
Current Time: Friday, 24-Apr-2015 12:10:46 CEST
Restart Time: Friday, 24-Apr-2015 12:10:41 CEST
Parent Server Config. Generation: 3
Parent Server MPM Generation: 2
Server uptime: 5 seconds
Server load: 0.40 0.24 0.17
Total accesses: 0 - Total Traffic: 0 kB
CPU Usage: u0 s0 cu0 cs0
0 requests/sec - 0 B/second -
1 requests currently being processed, 127 idle workers
```

PID	Connections		Threads		Async connections		
	total	accepting	busy	idle	writing	keep-alive	closing
19370	0	yes	0	64	0	0	0
19372	0	yes	1	63	0	0	0
Sum	0		1	127	0	0	0

$\frac{W}{\dots}$

Scoreboard Key:

"_" Waiting for Connection, "s" Starting up, "R" Reading Request, "w" Sending Reply, "R" Keepalive (read), "d" DNS Lookup, "c" Closing connection, "L" Logging, "G" Gracefully finishing, "I" Idle cleanup of worker, "." Open slot with no current process

Srv	PID	Acc	M	CPU	SS	Req	Conn	Child	Slot	Client	VHost	Request
4-2	19372	0/0/0	W	0.00	0	0	0.0	0.00	0.00	83.238.166.213	localhost:80	GET /server-status HTTP/1.0

Srv Child Server number - generation

PID OS process ID

Acc Number of accesses this connection / this child / this slot

M Mode of operation

CPU CPU usage, number of seconds

SS Seconds since beginning of most recent request

Req Milliseconds required to process most recent request

Conn Kilobytes transferred this connection

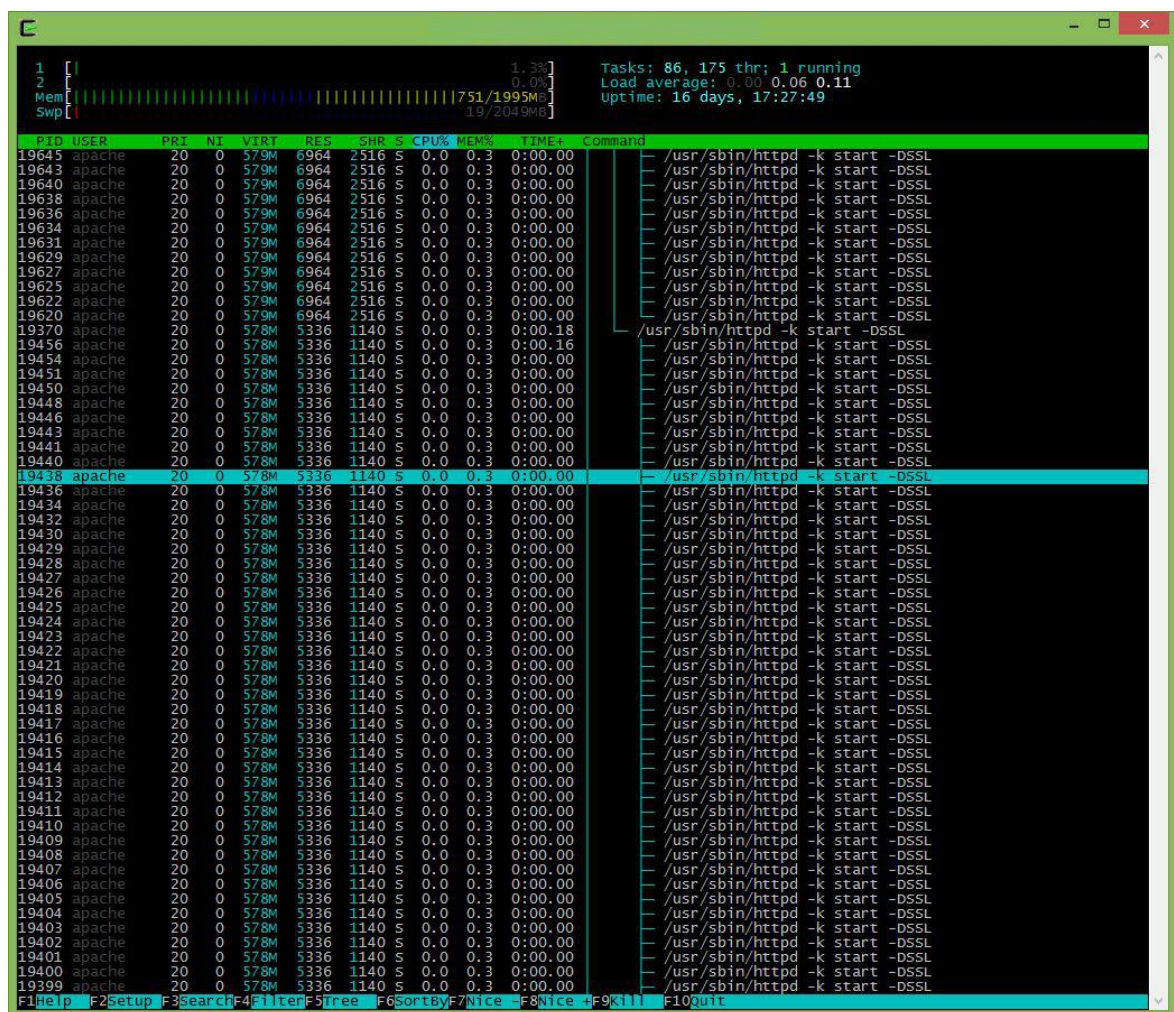
Child Megabytes transferred this child

Slot Total megabytes transferred this slot

SSL/TLS Session Cache Status:

```
cache type: SHMCB, shared memory: 512000 bytes, current entries: 0
subcaches: 32, indexes per subcache: 88
index usage: 0%, cache usage: 0%
total entries stored since starting: 0
total entries replaced since starting: 0
total entries expired since starting: 0
total (pre-expiry) entries scrolled out of the cache: 0
total retrieves since starting: 0 hit, 0 miss
total removes since starting: 0 hit, 0 miss
```

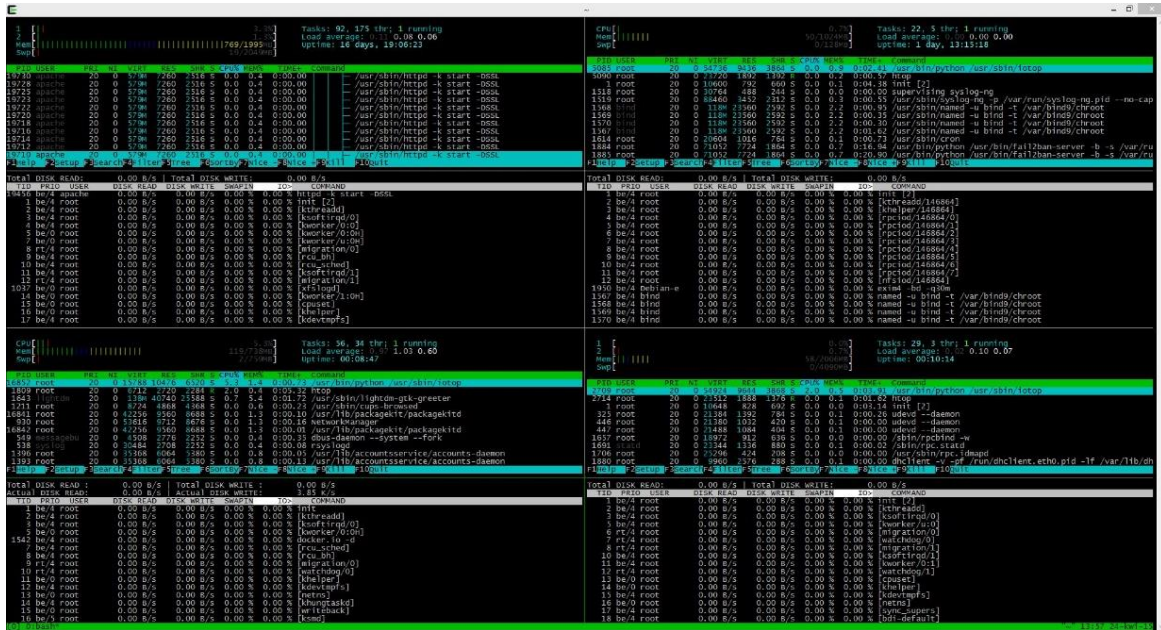

Tego typu podejście do bezpośredniego monitorowania serwerów jest możliwe tylko w przypadku ograniczonej ich liczby od kilku do kilkunastu sztuk. W przypadku większej ilości serwerów niż kilka sztuk w centrach przetwarzania danych, takie działania są już zwyczajnie niemożliwe, ze względu na ograniczenia ludzkiej percepcji. Jeden człowiek nie jest w stanie skutecznie monitorować i reagować na zdarzenia jeśli występują one zbyt często oraz dotyczą kilku systemów/serwerów jednocześnie. Nie jest też w stanie skutecznie obserwować więcej niż kilka serwerów (Rysunek 1.5). Dlatego też chcąc zwiększyć wydajność oraz zmniejszyć czas niedostępności usług/serwerów podczas awarii, należy zautomatyzować znaczną część standardowych działań. Działania automatyzuje się po to aby administrator mógł efektywniej obsługiwać awarie i zagrożenia kiedy wystąpią lub jeszcze zanim się pojawią.



Rysunek 1.4 – monitoring obciążenia CPU – Htop.

Jeśli nastąpi awaria, administrator musi być osobą, która jako pierwsza zostanie o takiej awarii powiadomiona. Powiadomienie takie jest realizowane poprzez dedykowany do tego celu system monitorujący. System, o którym mowa będzie tematem niniejszej pracy.

Większość centrów przetwarzania danych, tworzy systemy monitoringu, na swój własny użytek, gdyż muszą być one „uszyte na miarę” oraz muszą zostać wdrożone do konkretnych, często bardzo specyficznych zastosowań.



Rysunek 1.5 – konsolowy monitoring wielu zasobów/serwerów jednocześnie – Tmux.

Systemy tego typu zgodnie z nomenklaturą ITIL [1], możemy podzielić na reaktywne i pro-aktywne. Mówiąc obrazowo, systemem wspierającym reaktywny monitoring serwerów będą popularne wykresy przedstawiające aktualne parametry pracy serwerów (Rysunek 1.2).

W przypadku wystąpienia anomalii na wykresach, często nie wiadomo, czy anomalia pochodzi ze środowiska serwera i jest związana z awarią sprzętu lub oprogramowania, czy pochodzi z zewnątrz (atak na serwer), czy też jest związana ze specyficznym zachowaniem klientów, które może wywołać awarię lub być normalną sytuacją (np. większy ruch/obciążenie usług w związku z jakąś kampanią informacyjną w mass-mediach).

Administrator widząc anomalię na wykresach, musi podjąć działania sprawdzające jej przyczynę a następnie określić, czy potrzebne są jakieś działania zaradcze. W związku z tym, są to w większości przypadków działania reaktywne. *Działania reaktywne to takie, które są reakcją na zdarzenie, jakim jest anomalia zaobserwowana w systemie (np.: na wykresach obrazujących stan serwera/usługi).* W szczególnych przypadkach działanie tego

typu można zaklasyfikować jako pro-aktywne. Dzieje się tak kiedy zostanie zaobserwowana anomalia, która potencjalnie, w wyniku eskalacji, może doprowadzić do przeciążenia serwera lub do jego awarii.

Administrator obserwując wykresy, widząc tego typu anomalie, w większości przypadków działa reaktywnie. Wykresy same w sobie nie podają konkretnej przyczyny problemu, a jedynie sygnalizują jakiś skutek. Tak więc system przedstawiający administratorowi bardziej szczegółowe i przetworzone informacje o zaistniałych incydentach lub potencjalnych problemach, pozwoliłyby w wielu przypadkach zaoszczędzić wiele czasu potrzebnego, na wykrycie źródła problemu, a co za tym idzie pozwoliły szybciej rozwiązać problem lub podjąć odpowiednie działania zmierzające w tym celu.

Metody powiadomień administratora o awariach lub zdarzeniach mogą być różne. Generalnie możemy podzielić je na: „natychmiastowe” i „nie natychmiastowe” (opóźnione). Natychmiastowe powiadomienia to takie, które informują o awariach w momencie ich wystąpienia (dopuszczalne kilka sekund opóźnienia). Przykładem będzie tutaj wiadomość SMS wysłana z serwera, lub powiadomienie na konsoli systemowej obserwowanej w czasie rzeczywistym przez administratora.

Powiadomienia opóźnione to takie, które informują o awariach lub ich skutkach ze znacznym opóźnieniem od kilkunastu sekund do kilku/kilkunastu minut, a nawet godzin. Przykładem może być wiadomość e-mail wysłana na nienadzorowane konto pocztowe, wykresy rysujące się z kilkuminutowym opóźnieniem lub telefoniczne powiadomienie przez klienta.

Można także podejść do tego zagadnienia od strony wagi zgłoszeń dzieląc je na te „ważniejsze” i „mniej ważne”. Oba podejścia (dotyczące wagi informacji i szybkości z jaką informacje tego typu są dostarczane do administratorów) są w tym przypadku kluczowe. Następstwem obu tych podejść jest pojawienie się potrzeby określenia istotności powiadomień, poprzez wstępną ich klasyfikację, dotyczącą pilności reakcji na poszczególne zdarzenia. Kluczową w tym punkcie będzie wstępna selekcja (automatyczny wybór) metody powiadomienia o zdarzeniu, w celu przeciwdziałania dostarczaniu administratorowi informacji, które nie są istotne z racji rangi problemu (nie będą „zalewać” administratora informacjami zdublowanymi lub nieistotnymi na daną chwilę).

System monitoringu musi być odpowiednio przemyślany i zaimplementowany tak by zapewnić możliwie bezawaryjną pracę i powiadamianie nawet gdy monitorowany serwer ulegnie całkowitej awarii i sam nie będzie w stanie wysłać żadnej informacji o zdarzeniach, w tym o jego awariach.

Dobłą praktyką jest posiadanie osobnego/niezależnego serwera, który będzie pełnił funkcję serwera monitoringowego (monitorującego pracę innych serwerów). Serwer taki może integrować oprogramowanie monitorujące i zarządzające powiadomieniami o zdarzeniach dotyczących innych serwerów. Może pełnić także dodatkowe funkcje w miarę możliwości wydajnościowych (poczta techniczna, systemy bilingowe, itp). Im jest to bardziej złożony system i im więcej serwerów monitoruje, tym jego znaczenie dla całości systemu jest większe. W wielu przypadkach taki system monitoringowy powinien być, dla bezpieczeństwa, systemem składającym się z dwóch lub więcej niezależnych środowisk znajdujących się na niezależnych maszynach, robiących dokładnie to samo (ale niezależnie od siebie).

Innymi rozwiązaniami zapobiegającymi awariom, o których należy wspomnieć, aczkolwiek wykraczają one poza ramy niniejszej pracy, są tzw. rozwiązania typu „High Availability” (HA) [2], „Network Load Balancing” (NLB) [2]. Są to rozwiązania oparte o klastry redundantnych serwerów oraz rozwiązania CLOUD (oparte o chmurę obliczeniową). Zapewniają one wysoki poziom dostępności i niezawodności, poprzez przełączanie działania usług, z maszyny, która uległa awarii na maszynę zapasową. Mimo tego, że zapobiegają one awariom świadczonych usług jako całości, to jednak opieranie się tylko o te rozwiązania, nie gwarantuje nam optymalnego wykorzystania już istniejącego sprzętu/zasobów w pełni. Rozwiązania te nie zwalniają administratorów z monitorowania stanu poszczególnych serwerów wchodzących w skład klastrów, które mimo redundancji należy monitorować. Dlatego system wspomagający utrzymanie serwerów jest bardzo przydatny a niekiedy nawet niezbędny administratorom serwerów, zarówno w przypadku monitorowania pojedynczych maszyn fizycznych, redundantnych serwerów, klastrów składających się z wielu wyspecjalizowanych jednostek, czy chmur serwerowych (współpracujących ze sobą wielu klastrów) [2].

Istnieją dwa rodzaje środowisk serwerowych, którymi administruje się w innym zakresie. Serwery dedykowane lub serwery fizyczne, charakteryzują się tym, że administrator zarządzający takim serwerem posiada dostęp do funkcji systemu operacyjnego odpowiedzialnych za dostęp do fizycznego sprzętu, na którym funkcjonuje serwer.

Drugi rodzaj to serwery wirtualne VPS, których oprogramowanie oddzielone jest od sprzętu serwera poprzez oprogramowanie zarządzające (np.: hypervisor lub kontener).

Serwery VPS, nie mają pełnego i bezpośredniego dostępu do fizycznych zasobów sprzętowych serwera na którym są uruchomione, ale mają przydzielone wirtualne zasoby sprzętowe lub oddelegowany częściowy dostęp pośredni. W związku z powyższym, jako że

są to maszyny wydzielone (odizolowane) od fizycznych zasobów serwera, nie monitoruje się na nich zasobów, które dotyczą serwera macierzystego, na którym są one uruchomione. Za sprzęt fizyczny i monitorowanie jego zasobów odpowiedzialny jest administrator serwera fizycznego, na którym działają te maszyny wirtualne. Dla serwerów dedykowanych (fizycznych) zakres monitorowanych zasobów będzie inny (szerszy), niż dla maszyn VPS.

Takie parametry jak temperatura (CPU, HDD, RAM, obudowy), integralność macierzy RAID, błędy fizyczne na dyskach, realokowane sektory (S.M.A.R.T.), monitoruje się tylko na sprzęcie fizycznym (na serwerach dedykowanych).

Parametry dotyczące obciążenia procesora, zajętość pamięci RAM, limity użytkowników, obciążenie dysków, itp., należy monitorować zarówno na maszynach dedykowanych jak i na serwerach VPS.

Elementy monitorowane typowo sprzętowe (warstwa hardware) dzielą się na kilka obszarów. Pierwszy z nich to zasoby sprzętowe, na które zazwyczaj nie oddziałuje się programowo ze strony systemu operacyjnego serwera. Zasoby te są bezpośrednio związane ze sprzętem, na który oddziałuje się fizycznie, z zewnątrz lub przez specjalne oprogramowanie techniczne (np.: możliwe do uruchomienia przed włączeniem serwera, lub dotyczące układów około-serwerowych). Do tej warstwy sprzętowej zalicza się monitoring temperatury CPU, monitoring temperatury dysków, monitoring sprzętowych macierzy RAID (ich integralności), monitoring temperatury pamięci RAM, monitoring temperatury innych podzespołów wewnątrz obudowy serwera, prędkość przepływu powietrza/płynów chłodzących, prędkość obrotowa wentylatorów, itp. Te parametry utrzymuje się poprzez zewnętrzne czynniki (np.: chłodzące).

Inne czynniki zewnętrzne, które należy monitorować w serwerowni to: wilgotność powietrza, ciśnienie, napięcie w sieci energetycznej, zużycie energii, moc czynna, moc bierna, stan akumulatorów UPS, parametr PUE (power usage effectiveness – dla serwerowni jako całości) [3]. Jednakże te zagadnienia znajdują się poza zakresem niniejszej pracy, więc nie będą szczegółowo poruszane, ani rozwijane. Z powyższych zagadnień omówione zostaną kwestie dotyczące monitoringu temperatury dotyczącej poszczególnych podzespołów serwera, gdyż tylko one mogą być monitorowane poprzez system operacyjny działający na serwerze.

Temperatura wewnątrz obudowy, ma wpływ na żywotność podzespołów elektronicznych. Temperatura dysków twardych ma wpływ na żywotność dysków. Temperatura CPU ma wpływ na stabilność działania serwerów (zbyt wysoka powoduje zawieszanie się serwera, wyłączenie CPU w wyniku przegrzania). Należy monitorować

temperaturę podzespołów, tak by wychwycić zagrożenia z nimi związane (głównie niebezpieczny jej wzrost). Niewłaściwa temperatura grozi niestabilnością lub uszkodzeniem sprzętu.

Po wykryciu nieprawidłowych wartości temperatury administrator musi właściwie zareagować na przykład wymieniając uszkodzony wentylator, lub zmieniając parametry chłodzenia serwera/pomieszczenia/korytarza. W tej warstwie należy monitorować dyski twarde oraz ich zużycie (degenerację), którego wyznacznikiem jest ilość już istniejących błędów, ilość realokowanych sektorów oraz ilość pojawiających się nowych błędów fizycznych na tych dyskach (szybkość ich przyrastania). W tej warstwie znajdują się także sprzętowe jak i programowe macierze RAID (software RAID).

Skrypty i programy monitorujące dla tej warstwy będą miały zastosowanie tylko na fizycznym sprzęcie, nie będą natomiast przydatne na serwerach wirtualnych VPS.

Drugi obszar dotyczy także warstwy sprzętowej (hardware), ale od strony systemu operacyjnego. Do tej warstwy należy zaliczyć zarówno sprzęt fizyczny jak i wirtualny, gdyż w obu przypadkach należy monitorować jego zasoby. Do takich zasobów należą chwilowe procentowe obciążenie „CPU USAGE” i średniego obciążenie „CPU LOAD”, od których zależy stabilność i wydajność działania serwera. Przeciążony procesor na serwerze praktycznie uniemożliwia lub bardzo ogranicza możliwość korzystania z serwera jego użytkownikom. W takiej sytuacji serwer przestaje właściwie spełniać swoją rolę i staje się w znacznej mierze bezużyteczny. Należy także monitorować bieżącą zajętość pamięci operacyjnej RAM oraz pamięci wirtualnej SWAP. Bieżące obciążenie dysków twardych (w stosunku do ich wydajności). Niedobór wolnej pamięci RAM oraz SWAP będzie skutkował usuwaniem z serwera działających procesów, dla których tej Pamięci nie wystarczy. W niektórych przypadkach korzystanie z pamięci SWAP, może spowodować znaczne zmniejszenie wydajności całego serwera, a co za tym idzie spowodować utrudnienia związane z korzystaniem z jego usług.

Skrypty dotyczące monitoringu tych zasobów, należy stosować zarówno na serwerach fizycznych, jak i na serwerach wirtualnych.

W warstwie oprogramowania, czyli stricte software, należy monitorować integralność danych w systemach plików, zasoby dotyczące limitów (CPU, HDD, RAM, ilości procesów, przepustowości sieci, ilości jednoczesnych połączeń sieciowych, itp.) przydzielonych użytkownikom. W tej warstwie należy również uwzględnić skrypty dotyczące obsługi tworzenia kopii bezpieczeństwa plików oraz baz danych. Skrypty porządkujące dane i automatycznie kasujące zbędne tymczasowe pliki zalegające

w obszarach wspólnych. Skrypty dotyczące monitorowania tej warstwy należy stosować zarówno na serwerach fizycznych, jak i na serwerach wirtualnych.

Monitoring elementów sprzętowych i programowych serwera jest niezbędny aby zapobiegać niestabilności serwera, zawłaszczeniu zasobów serwera przez jednego użytkownika. System operacyjny jak i oprogramowanie wspomagające mają za zadanie zapewnić właściwy przydział zasobów serwera jego użytkownikom oraz powinny aktywnie przeciwdziałać czynnikom próbującym doprowadzić do destabilizacji i przeciążenia serwera lub jego usług.

2. Technologie i narzędzia

Technologia w jakiej zostaną zaimplementowane rozwiązania zaprezentowane w niniejszej pracy będzie opierać się o skryptowe języki programowania takie jak: BASH, SH, AWK, jak również język C++ z framework'iem QT oraz aplikacje serwerowe takie jak: Apache, MySQL, PHP, Exim, Dovecot, ProFtpd, Rsync. W pracy tej zostaną także omówione kwestie związane z oprogramowaniem komercyjnym takim jak DirectAdmin, choć nie zostanie ono dołączone do tej pracy ze względów licencyjnych. Omówione zostaną także już istniejące narzędzia dostępne na rynku, ich użyteczność, zalety i wady [4].

Niniejsza praca zostanie podzielona na dwie części. W pierwszej części zostaną przedstawione skrypty w w/w językach służące do monitorowania stanu serwera i reagowania na zdarzenia. Skrypty te można umieścić na serwerze docelowym (monitorowanym) lub niektóre z nich po drobnej modyfikacji można również umieścić na serwerze monitorującym (rozwiązanie zalecane w niektórych przypadkach ze względu na bezpieczeństwo i oddzielenie zasobów administratora od zasobów jego klienta). W skład tych skryptów wejdą także aplikacje skryptowe testujące poprawność działania niektórych z nich lub generujące sztuczne obciążenie zasobów serwera w celach testowych.

W drugiej części pracy zostanie przedstawiona aplikacja napisana w języku C++ w oparciu o QT framework (biblioteka QT), która będzie zbierała najważniejsze informacje z powyższych skryptów i w formie komunikatów, w zależności od ich istotności/priorytetu będzie wysyłała je z serwera do administratora przy pomocy protokołu TCP/IP na jego zdalną stację roboczą (komputer typu desktop) [5].

Główną technologią w jakiej zostaną zrealizowane programy monitorujące stan serwera zarówno od strony sprzętowej jak i oprogramowania, będzie skryptowy język programowania w środowiskach uniksowych o nazwie BASH. Drugim językiem dla prostszych skryptów będzie programowanie w powłoce SH (prostsza, oryginalna powłoka, na której podstawie stworzono później BASH, jest szybsza od BASH). Możliwe, że wykorzystane zostaną także inne technologie takie jak skrypty PHP, AWK, Python, Perl.

Dlaczego BASH? BASH jest najbardziej rozpowszechnionym językiem skryptowym w środowiskach GNU/Linux, dostępny jest także, obok powłoki TCSH (TC shell) w środowiskach BSD takich jak FreeBSD, które są także wykorzystywane na dużą skalę w środowiskach hostingowych.

BASH jest także stosowany w środowisku CYGWIN, które może zostać zainstalowane na dowolnym serwerze działającym w oparciu o oprogramowanie Microsoft Server. BASH więc jest to język bardzo uniwersalny i wieloplatformowy. Skrypty nie wymagają kompilacji i po naniesieniu drobnych poprawek lub w wielu przypadkach nawet bez nich, będą w stanie działać w różnych środowiskach, na różnych platformach sprzętowych, wirtualnych i w różnych systemach operacyjnych.

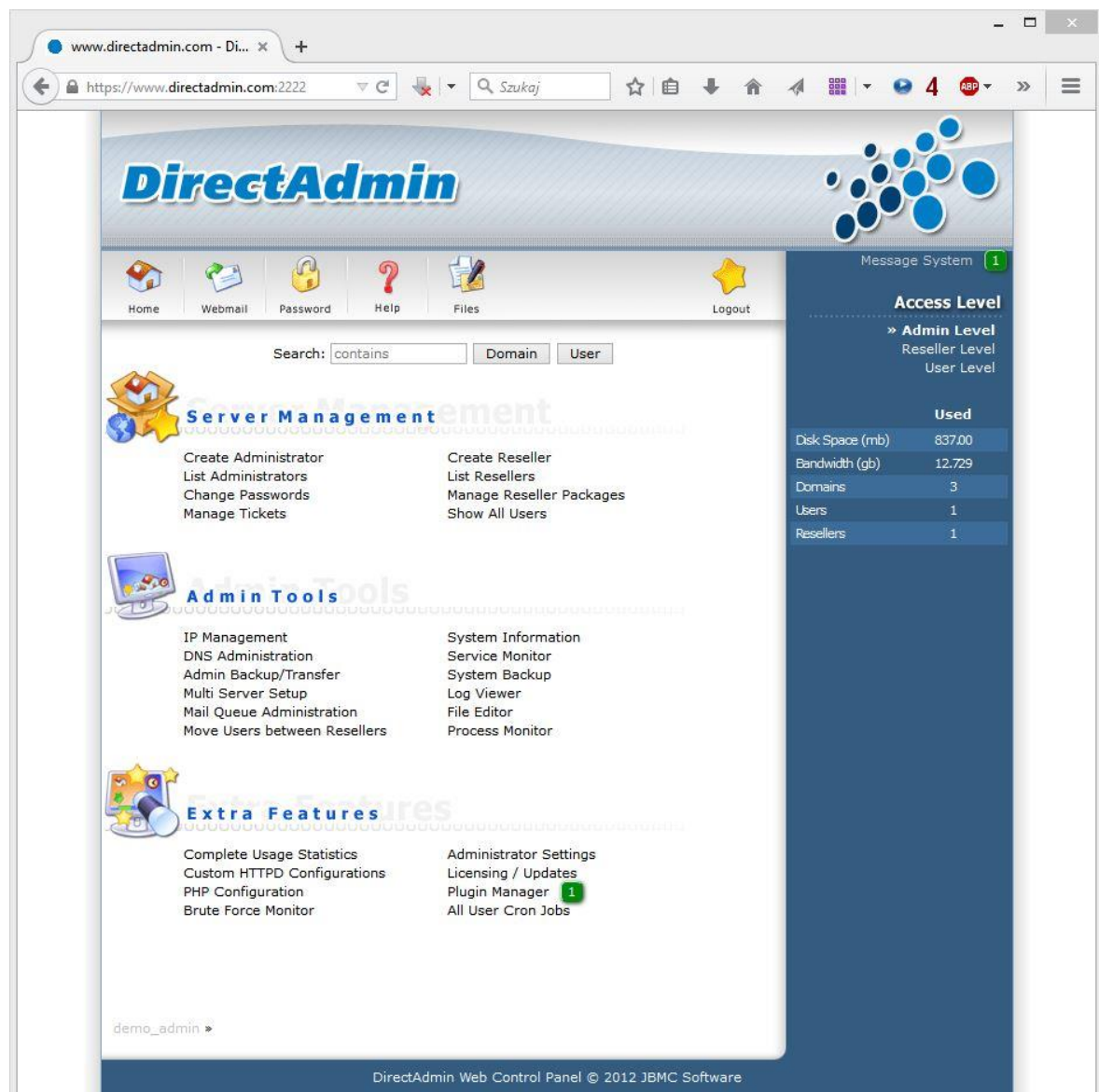
BASH może wykorzystywać wbudowane polecenia systemowe jak i zewnętrzne programy dostępne w środowisku. Jest to więc bardzo użyteczny dla administratorów język programowania i jednocześnie bardzo łatwy do nauki. Posiada wiele mechanizmów zgodnych ze standardem POSIX [6] opisanym w dokumencie IEEE 1003.2. *Docelowo BASH ma stać się powłoką systemową w pełni zgodną z tym standardem, niemniej jednak, w niektórych przypadkach można użyć opcji --posix aby wymusić tryb pracy BASH, który będzie najbardziej zbliżony z POSIX [7].*

W drugiej części pracy wykorzystana zostanie technologia programowania obiektowego w języku C++ przy użyciu biblioteki QT. Aplikacje pisane przy pomocy C++ i QT, także są aplikacjami wieloplatformowymi. Dzięki bibliotece QT zazwyczaj bez większych modyfikacji aplikacje takie można skompilować pod różne systemy operacyjne i na różne platformy sprzętowe. Jest to bardzo ważny atut, ponieważ środowiska serwerowe są zazwyczaj środowiskami heterogenicznymi. Uniwersalność języka i jego wieloplatformowość oszczędza czas administratora, a co za tym idzie zwiększa wydajność pracy administratorów w centrach przetwarzania danych.

Ponadto dzięki takiemu podejściu można skompilować aplikację po stronie serwera z systemami Linux, FreeBSD [8], Windows Serwer (są to najczęściej używane środowiska operacyjne w centrach przetwarzania danych), a po stronie klienta (będzie to stacja PC z GUI) można uruchomić aplikację w dowolnym wybranym środowisku (np.: Windows, MacOS/X, Linux), które obecnie używa administrator na swoim stanowisku roboczym. Aplikacja napisana w języku C++, gwarantuje pracę w trybie natywnym, a więc zapewnia większą niezawodność i wydajność (szybkość), co w wielu przypadkach może mieć duże znaczenie. Ponadto biblioteka QT oferuje wiele dobrze przetestowanych mechanizmów, klas, które są gwarantem stabilności już na samym początku pracy deweloperskiej, co w dużym stopniu ogranicza konieczność wielokrotnego kompilowania, testowania oprogramowania w trakcie jego rozwoju [9].

Ponadto do stworzenia systemu wsparcia monitorowania i utrzymania serwerów „ServerMonitor”, zostaną użyte następujące narzędzia wspomagające pracę deweloperską:

- QtCreator wersja 5.4 – kompletne środowisko programistyczne (IDE, kompilator, debugger, itp.) [10];
- Edytory Vim i Mcedit z pakietu Midnight-commander oraz Notepad++;
- Debian Gnu/Linux v. 7.8 (Wheezy), 64-bit oraz 8.1 (Jessie);
- FreeBSD 9, 64-bit;
- DirectAdmin (Rysunek 2.1) [11];
- GitHub;
- VirtualBox;
- Cygwin;
- OpenOffice, MS Office;
- DrawIO [12];



Rysunek 2.1 – panel zarządzania – DirectAdmin.

3. Projekt systemu

Celem niniejszej pracy jest stworzenie zbioru skryptów oraz aplikacji służących do monitorowania stanu serwera oraz umożliwiających reagowanie lub/i powiadamianie administratora o zdarzeniach dotyczących monitorowanych parametrów serwera, jego zasobów, uruchomionych procesów, zdarzeń, na które administrator powinien zwrócić uwagę.

Do wytworzenia niniejszego systemu zostanie użyty kaskadowy model wytwarzania aplikacji, który będzie składał się z pięciu etapów:

- określenie wymagań użytkownika;
- projektowanie (część skryptowa i część komunikacyjna);
- implementacja/kodowanie (skrypty robocze, skrypty testujące, aplikacja prezentująca pozyskane dane);
- testowanie (na maszynach wirtualnych);
- konserwacja i utrzymanie.

3.1. Założenia systemu

System będzie składał się z części skryptowej – monitorującej i podejmującej działania zaradcze (w miarę możliwości) oraz z aplikacji prezentującej informacje pozyskane przez skrypty do administratora.

System powinien rozgraniczać zdarzenia mniej istotne oraz te bardziej istotne. Jeśli jest to możliwe powinien rozwiązać problem autonomicznie, w przeciwnym razie powinien wysłać powiadomienie jedną z kilku dróg komunikacji lub wszystkimi (w zależności od jego istotności).

Do dróg komunikacji można zaliczyć standardowe powiadomienie przy pomocy poczty elektronicznej, powiadomienie przy pomocy wiadomości SMS, lub powiadomienie przy pomocy prezentacji danych w aplikacji „ServerMonitor”.

W związku z tym, że założenia aplikacja „ServerMonitor” ma prezentować informacje o bieżącym stanie serwera szybciej niż powiadomienia wysyłane poprzez SMS, ostatecznie ma ona zastąpić tą drogę komunikacji. Dlatego w niniejszy projekt skupi się tylko na komunikacji za pomocą poczty elektronicznej oraz na prezentacji informacji przy pomocy projektowanej aplikacji „ServerMonitor”.

Podstawowym wymogiem dla części skryptowej jest wstępne odfiltrowanie zbędnych informacji i wyselekcjonowanie tylko tych, które są najbardziej istotne, do przekazania administratorowi serwera, poprzez aplikację „ServerMonitor”. Być może częściowo uda się to uzyskać na etapie projektowania skryptów lub ich wdrażania.

Ponadto w przypadku niektórych czynności, na przykład tych pilnujących poczynań użytkowników, ich limitów, użycia zasobów serwera, system powinien działać całkowicie autonomicznie (samodzielnie). Oznacza to, że powinien podejmować działania zaradcze od razu po ich wykryciu bez powiadamiania administratora lub z powiadomieniem o zdarzeniu (jeśli będzie to poważne naruszenie) oraz z zapisaniem takich informacji w dziennikach systemowych (logach).

Drugim wymaganiem jest stosunkowo szybkie dostarczenie informacji na temat monitorowanych funkcji serwera do administratora przez aplikację „ServerMonitor”, tak by administrator mógł w miarę możliwości szybko podjąć działania zaradcze i rozwiązać problem. Informacje mniej istotne, co do których reakcja może zostać odsunięta w czasie bez uszczerbku dla monitorowanych serwerów, będą wysyłane tylko poprzez pocztę elektroniczną. Informacje bardziej istotne również zostaną przesłane pocztą elektroniczną przy jednoczesnym prezentowaniu ich poprzez aplikację komunikacyjną „ServerMonitor”.

Podstawowym wymogiem dla aplikacji będzie szybkie wyświetlenie informacji na monitorze stacji roboczej administratora (lub na urządzeniu mobilnym).

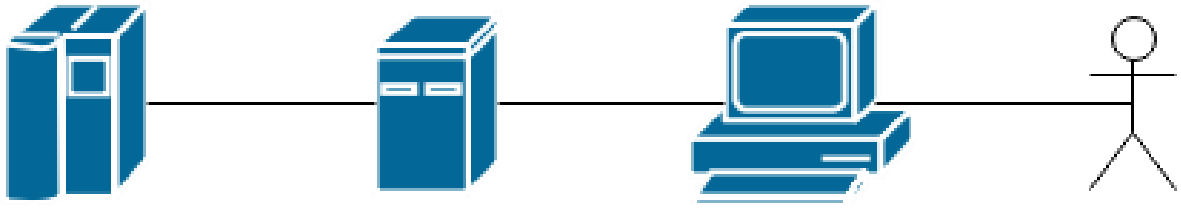
Drugorzędnym wymaganiem dla systemu, lecz także bardzo istotnym będzie możliwość obsługi wielu różnych serwerów poprzez uruchomienie dla nich wielu instancji aplikacji monitorującej „ServerMonitor”.

Tabela 3.1 - Identyfikacja zdarzeń w systemie 'ServerMonitor (SM)'

Lp.	Zdarzenie	Serwer fizyczny	Serwer VPS	Powiad. poprzez e-mail	Powiad. poprzez aplikację	Metoda uzyskania informacji
1 .	Temperatury CPU > limit	+		+	+	SNMP
2 .	Obciążenie CPU > 90%	+	+	+	+	SNMP
3 .	CPU LOAD > limit	+	+	+	+	SNMP
4 .	Temperatury HDD > limit	+		+	+	SNMP
5 .	Obciążenie HDD > limit	+	+	+	+	skrypt
6 .	Błędy na HDD	+		+		skrypt
7 .	Rośnie ilość błędów na HDD	+		+	+	skrypt
8 .	Zdegradowana macierz RAID	+		+	+	oprogramowanie syst.

9 .	RAID w trakcie integracji/weryfikacji	+		+		oprogramowanie syst.
10 .	Zajętość RAM > 90%	+	+	+	+	SNMP
11 .	Swap > limit	+	+	+		SNMP
12 .	Trwa backup plików	+	+	+		oprogramowanie syst.
13 .	Zakończono backup plików	+	+	+		oprogramowanie syst.
14 .	Trwa backup baz danych MySQL	+	+	+		oprogramowanie syst.
15 .	Zakończono backup baz danych MySQL	+	+	+		oprogramowanie syst.
16 .	Użytkownik przekracza limit CPU	+	+			skrypt
17 .	Zabito proces użytkownika	+	+			skrypt
18 .	Obniżono priorytet procesu użytkownika	+	+			skrypt
19 .	Użytkownik przekracza limit ilości procesów	+	+	+	+	user_process.sh
20 .	Wymagana aktualizacja oprogramowania	+	+	+		oprogramowanie syst.
21 .	Proces apache nie działa	+	+	+	+	skrypt
22 .	Proces exim nie działa	+	+	+	+	skrypt
23 .	Proces dovecot nie działa	+	+	+	+	skrypt
24 .	Proces proftpd nie działa	+	+	+	+	skrypt
25 .	Proces mysql nie działa	+	+	+	+	skrypt
26 .	Proces php-fpm nie działa	+	+	+	+	skrypt
27 .	Proces bind nie działa	+	+	+	+	check-bind.sh
28 .	Serwer został zrestartowany	+	+	+		rc.local
29 .	Zapytanie MySQL zostało zabite	+	+	+		skrypt
30 .	Testowa baza MySQL nie odpowiada	+	+	+	+	skrypt
31 .	Serwer nie odpowiada na ping	+	+	+	+	skrypt
32 .	Użytkownik dodał domenę {nazwa_domeny}	+	+	+		(skrypt DirectAdmin)
33 .	Użytkownik skasował domenę {nazwa_domeny}	+	+			(skrypt DirectAdmin)
34 .	Ilość połączeń do usługi apache > limit					brak
35 .	Ilość połączeń do usługi exim > limit					brak
36 .	Ilość połączeń do usługi dovecot > limit	+	+	+	+	skrypt

Zasadę działania aplikacji można prześledzić na diagramie kontekstowym:



Rysunek 3.1 – diagram kontekstowy.

Diagram kontekstowy (Rysunek 3.1) aplikacji 'ServerMonitor' składa się z obiektów:

- Serwer monitorowany;
- Serwer monitorujący;
- Komputer PC administratora (ewentualnie urządzenie mobilne);
- Administrator – osoba w firmie obsługująca system 'ServerMonitor', mająca możliwość reagowania na zdarzenia i decydująca, czy podejmować w związku z nimi działania;

3.2. Specyfikacja wymagań funkcjonalnych

Podstawowe funkcje systemu 'ServerMonitor':

- Możliwość zdefiniowania nowych skryptów, które w ustandaryzowany sposób będą mogły korzystać z aplikacji komunikacyjnej SM lub wysyłać powiadomienia poprzez e-mail;
- Klasyfikacja zgłaszanych problemów poprzez nadawanie im priorytetów;
- Możliwość zdefiniowania drogi przesłania powiadomienia do administratora lub jego braku;
- Łatwe dodawanie obsługi nowego serwera dla istniejącego zestawu gotowych skryptów.

3.3. Specyfikacja wymagań niefunkcjonalnych

Wymagania niefunkcjonalne systemu:

Skalowalność, możliwość łatwego dołączania nowych serwerów i usług do monitoringu;

Wieloplatformowość – możliwość funkcjonowania na różnych platformach sprzętowych i systemach operacyjnych;

Możliwość uruchomienia zarówno na serwerach dedykowanych jak i na serwerach wirtualnych;

3.4. Projekt struktury systemu

3.4.1. Część skryptowa

Do przedstawionych wcześniej zdarzeń użyte zostaną skrypty przedstawione w tabeli (Tabela 3.2 – lista skryptów.)

Tabela 3.2 – lista skryptów.

Skrypt	Opis
hardlink-backup-server.sh	Tworzenie kopii zapasowych plików
(Automysqlbackup)	Tworzenie kopii zapasowych baz danych
check-apache.sh	Sprawdzanie działania usługi apache
check-bind9.sh	Sprawdzanie działania usługi bind
check-dovecot.sh	Sprawdzanie działania usługi dovecot
check-exim.sh	Sprawdzanie działania usługi exim
check_mysql.sh	Sprawdzanie działania usługi mysql
check_php-fpm.sh	Sprawdzanie działania usługi php-fpm
check-ftp.sh	Sprawdzanie działania usługi proftpd
-	Sprawdzanie aktualizacji oprogramowania
check_all_snmp.sh	Sprawdzanie średniego obciążenia CPU
check_all_snmp.sh	Sprawdzanie temperatury CPU
check_all_snmp.sh	Sprawdzanie chwilowego obciążenia CPU
check_hdd_usage.sh	Sprawdzanie obciążenia dysków twardych
-	Sprawdzanie błędów na dyskach
check_all_snmp.sh	Sprawdzanie temperatury dysków
-	Zabijanie powolnych zapytań SQL
-	Sprawdzanie działania testowej bazy danych
pingtest-ipv4.sh (...-ipv6)	Ping testowy do serwera
-	Sprawdzanie integralności macierzy RAID
check_all_snmp.sh	Sprawdzanie użycia pamięci RAM
rc.local	Skrypty startowe
check_all_snmp.sh	Sprawdzanie wykorzystania pamięci wirtualnej SWAP

-	Sprawdzanie limitu połączeń do usługi dovecot
onekiller.sh	Pilnowanie limitów CPU użytkownika
-	Pilnowanie limitu procesów użytkownika

Ponadto zostaną użyte następujące skrypty generujące sztuczne obciążenie w celu dokonania testów prawidłowego funkcjonowania systemu „ServerMonitor”:

- count_pi.sh – skrypt obliczający liczbę π zadaną ilość razy.
- count_fi.sh – skrypt obliczający liczbę ϕ zadaną ilość razy.

Skrypty działają na zasadzie zapisywania dokonywanych pomiarów/sprawdzeń na dysku twardym serwera w określonej lokalizacji, z której to przy pomocy aplikacji rsync/ssh, dane te zostaną przesłane do aplikacji „ServerMonitor”. Takie podejście zapewnia wysoki poziom bezpieczeństwa, ponieważ na serwerze nie są instalowane dodatkowe aplikacje, ani nie są otwierane dodatkowe porty, co w dużym zakresie zwalnia administratora z zabezpieczenia dodatkowej usługi.

3.4.2. Część aplikacyjna „ServerMonitor”

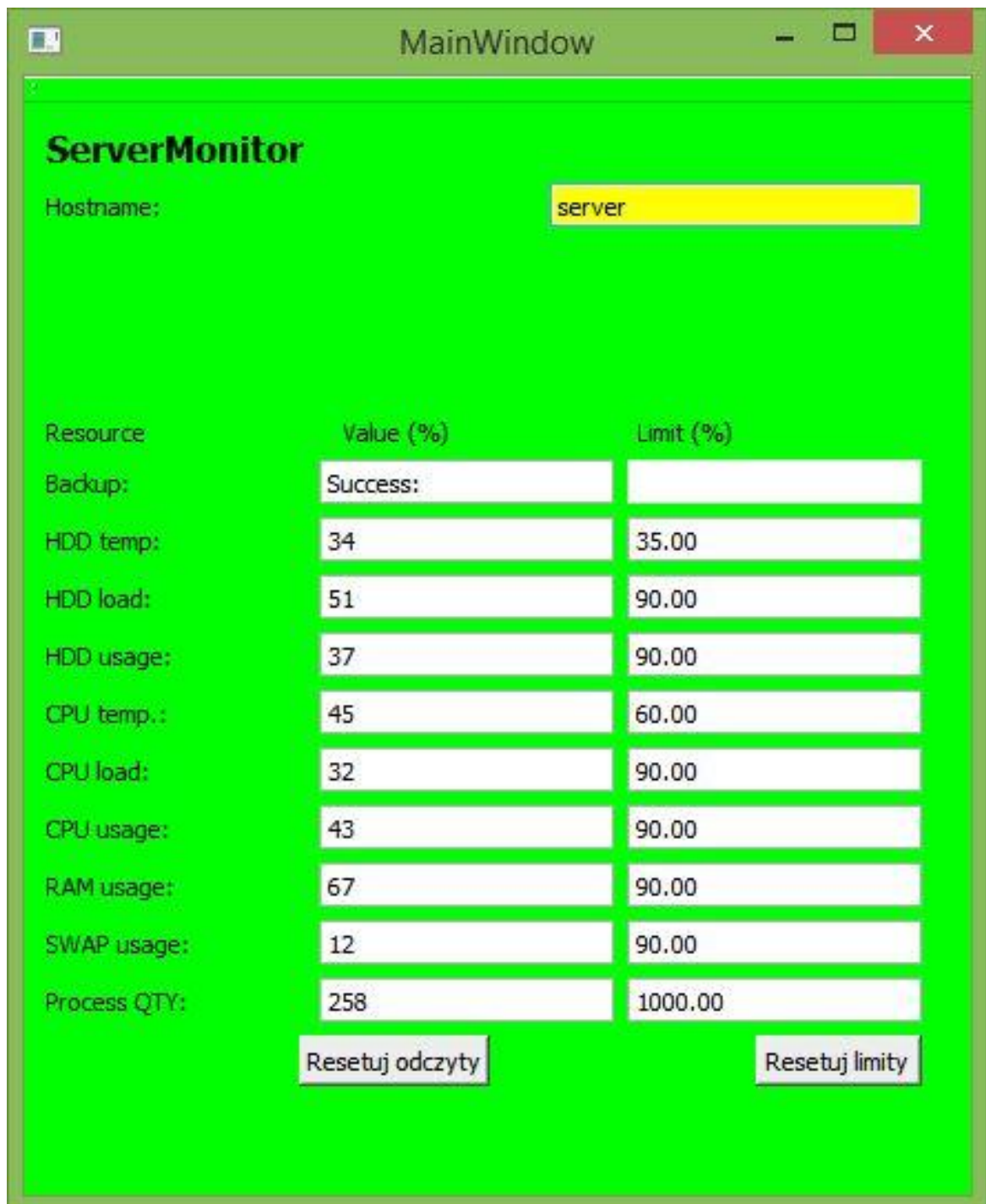
Aplikacja „ServerMonitor” została napisana w języku C++ z użyciem biblioteki QT [13]. Działa na zasadzie odczytywania wartości z plików tekstowych, które zostały pobrane z serwera poprzez rsync/ssh w celu wyświetlenia ich na stanowisku administratora. Dzięki takiemu podejściu wyeliminowana została konieczność tworzenia skomplikowanych systemów uwierzytelniających oraz ustanawiania szyfrowanych kanałów komunikacyjnych, gdyż aplikacje rsync/ssh w sposób natywny je wykorzystują i obsługują.

Najważniejszym elementem aplikacji „ServerMonitor” jest klasa „QFileSystemWatcher”, dzięki której może odbywać się monitoring plików w których zapisywane są aktualne informacje pobrane z serwera. Klasa dziedziczy z głównej klasy QObject a deklarowana jest poprzez `#include <QFileSystemWatcher>`. Dostępna jest w bibliotece Qt od wersji 4.2.

3.5. Projekt interfejsu graficznego

Praca z aplikacją nie wymaga uwierzytelniania ani logowania ze względu na wcześniej opisaną bezpieczną technologię rsync/ssh, za pomocą której następuje uwierzytelnienie do serwera i pobieranie danych dotyczących monitorowanych parametrów.

Wygląd okna aplikacji po stronie stanowiska administratora można zobaczyć na poniższym zrzucie ekranowym (Rysunek 3.2).



Resource	Value (%)	Limit (%)
Backup:	Success:	
HDD temp:	34	35.00
HDD load:	51	90.00
HDD usage:	37	90.00
CPU temp.:	45	60.00
CPU load:	32	90.00
CPU usage:	43	90.00
RAM usage:	67	90.00
SWAP usage:	12	90.00
Process QTY:	258	1000.00

Rysunek 3.2 – okno aplikacji "ServerMonitor".

3.5.1. Interfejs

W związku z tym, że aplikacja służy jedynie do monitorowania parametrów pracy serwera nie jest zbyt rozbudowana w części wizualnej. Najważniejsze jej elementy kryją się w kodzie skryptów sh/bash. Są nimi ustawienia konfiguracyjne i skrypty automatyzujące

wiele czynności, których ostatecznym celem jest powiadomienie administratora o stanie serwera, incydentach, anomaliach i awariach. Większość z tych skryptów wykorzystuje jako główną metodę powiadamiania zwykłą pocztę e-mail/sms oraz zapis do logów (dzienników systemowych) serwera. Jedna instancja aplikacji „ServerMonitor” ma za zadanie przedstawiać jedynie najważniejsze informacje o jednym serwerze. Możliwe jest zdefiniowanie limitów wartości (druga kolumna) przy, których włączą się „alarmy” lub powiadomienia.

Aplikacja „ServerMonitor” może jednocześnie monitorować wiele różnych serwerów (Rysunek 3.3), co osiąga się poprzez jej wielokrotne uruchomienie (wiele niezależnych instancji) z różnych niezależnych folderów, do których przesyłane są dane o bieżącym stanie monitorowanych serwerów.



Rysunek 3.3 – wiele instancji "ServerMonitor".

4. Realizacja systemu

4.1. Implementacja

Część serwerowa to skrypty, które monitorują usługi i zasoby serwera oraz podejmują różnego rodzaju działania w zależności od sytuacji. Mogą restartować usługę, blokować dostęp do usług, powiadamiać administratora poprzez email lub aplikację „ServerMonitor”.

Aby jednak umożliwić bezpieczną i bezproblemową komunikację pomiędzy stanowiskiem administratora a serwerem, w pierwszej kolejności należy zainstalować aplikacje SSH i Rsync oraz wygenerować klucze, dzięki którym aplikacja Rsync będzie łączyła się z serwerem poprzez protokół SSH/Scp za pomocą klucza. W przypadku środowiska Windows potrzebna będzie dodatkowe oprogramowanie Cygwin, które dostarcza kolekcję narzędzi niezbędnych do obsługi środowiska skryptowego i połączeń z serwerami Liux/Unix. Po zainstalowaniu środowiska Cygwin’a, warto uzupełnić je o oprogramowanie apt-cyg [14], dzięki, któremu można znacznie uprościć proces instalacji aplikacji Ssh, Rsync i innych, w środowisku Cygwin, przy pomocy poleceń:

```
apt-cyg install openssh  
apt-cyg install rsync
```

W środowisku Linux analogicznie należy zainstalować powyższe oprogramowanie. W dystrybucjach Debian/Ubuntu i pochodnych będą to polecenia:

```
apt-get install openssh-client  
apt-get install rsync
```

Po stronie serwera dodatkowo należy zainstalować aplikację openssh-server:

```
apt-get install openssh-client  
apt-get install openssh-server  
apt-get install rsync
```

Po wstępnym przygotowaniu środowiska, należy po stronie klienta wygenerować klucz SSH i umieścić go na serwerze. Operację przeprowadzamy przy pomocy komend przedstawionych na poniższym listingu (Listing 1):

Listing 1 – generowanie klucza ssh.

```
1 klient@IP:~> ssh-keygen -t rsa
2 Generating public/private rsa key pair.
3 Enter file in which to save the key (/home/a/.ssh/id_rsa):
4 Created directory '/home/a/.ssh'.
5 Enter passphrase (empty for no passphrase):
6 Enter same passphrase again:
7 Your identification has been saved in /home/user/.ssh/id_rsa.
8 Your public key has been saved in /home/user/.ssh/id_rsa.pub.
9 The key fingerprint is:
10 5e:4a:05:79:6a:9f:56:7c:ae:a9:68:37:bc:34:e4 klient@IP
```

Następnie należy utworzyć folder ~/.ssh na zdalnym serwerze:

```
klient@IP:~> ssh serwer@IP2 mkdir -p .ssh
serwer@IP2's password:
```

Ostatecznie należy wgrać wygenerowany klucz publiczny ssh na serwer:

```
klient@IP:~> cat .ssh/id_rsa.pub | ssh serwer@IP2 'cat \
>> .ssh/authorized_keys'
serwer@IP2's password:
```

Po tych czynnościach możliwym stanie się logowanie się do serwera poprzez Ssh/Rsync za pomocą klucza publicznego i bez użycia dodatkowego hasła:

```
klient@IP:~> ssh serwer@IP2
```

Skryptami, które w pierwszej kolejności skorzystają z powyższej metody logowania bez użycia hasła, będą skrypty do pobierania i tworzenia kopii zapasowej wszystkich istotnych plików serwera. Skrypty do tworzenia kopii bezpieczeństwa składają się z trzech plików, w których umieszczono sekwencje poleceń. Pierwszy z nich to skrypt inicjalizujący,

drugi to właściwy skrypt, a trzeci to lista wykluczeń folderów, których nie należy pobierać z serwera do kopii bezpieczeństwa.

Na listingu (Listing 2) przedstawiono skrypt inicjalizujący tworzenie kopii bezpieczeństwa, który jest kompatybilny tylko z systemami Unix/Linux (został przetestowany w dystrybucjach Debian Wheezy oraz Jessie pracujących w środowisku VPS). Inicjalizacja to utworzenie pierwszej kopii, z której to dla zaoszczędzenia miejsce będą w następnym skrypcie tworzone twarde dowiązania (hard link), dzięki czemu, dane które się nie zmieniają w kolejnych kopiach będą zajmowały na dysku jedną i tą samą przestrzeń (nie będą niepotrzebnie powielane do nowych plików). Takie podejście pozwoli w dłuższej perspektywie czasu zaoszczędzić znaczną ilość miejsca na dysku serwera tworzącego kopie bezpieczeństwa.

Poniżej (Listing 2) znajduje się kod źródłowy skryptu inicjalizującego wykonanie pierwszej kopii bezpieczeństwa. Skrypt nosi nazwę: `initial-backup-server.sh`, jest to skrypt zrealizowany w powłoce SH.

Listing 2 – initial-backup-server.sh.

```
1  #!/bin/sh
2
3  user=root
4  ip=192.168.1.1
5  exclude=exclude-list.txt
6
7  data=$(date +%Y-%m-%d)
8  path=/home/BACKUP/SERVER/backup-server-`${data}`
9  mkdir -p `${path}`
10
11 if [ -e "${path}.log" ]; then
12     /usr/bin/rsync -avzP --ipv4 --exclude-from `${exclude}` \
13     --delete-during `${user}@`${ip}`:/* `${path}`/ >> `${path}.log`
14 else
15     /usr/bin/rsync -avzP --ipv4 --exclude-from `${exclude}` \
16     --delete-during `${user}@`${ip}`:/* `${path}`/ > `${path}.log`
17 fi
```

Powyższa czynność jest niezbędna w celu utworzenia pierwszej kopii. Kopia ta będzie stanowiła podstawę linkowania plików dla następnej, która z kolei będzie podstawą dla następnych kopii i tak dalej. Każda następna kopia będzie tworzyła twarde linki do plików, które nie uległy zmianie w stosunku do poprzedniej kopii bezpieczeństwa, jako

nowe pliki będą pobierane i zapisywane tylko te, które uległy zmianie, od czasu wykonania poprzedniej kopii. W ten sposób uzyskamy bardzo wydajny i oszczędzający cenne zasoby dyskowe system tworzenia kopii bezpieczeństwa.

Ponadto w obu skryptach użyto komendy rsync z opcją „-z”, która oznacza, że przesyłane dane są kompresowane w trakcie przesyłania, co dodatkowo pozwala zmniejszyć zużycie przepustowości łącza. Może to być bardzo przydatne w przypadku połączeń pomiędzy zdalnymi niezależnymi lokalizacjami.

Skrypt `hardlink-backup-server.sh`, którego zawartość przedstawiono na poniższym listingu (Listing 3), odpowiada za tworzenie każdej następnej kopii bezpieczeństwa. Zastosowano w nim opcję programu Rsync o nazwie „--link-dest”, dzięki której wskazywana jest poprzednia kopia [15]. Jak już wcześniej zostało wspomniane tworzone są twarde dowiązania do poprzedniej kopii – właśnie dzięki tej dyrektywie. Skrypt został zaprojektowany do uruchamiania raz na dobę (oraz części) z demona CRON (mechanizm zegarowego uruchamiania programów w systemach Linux/Unix).

Listing 3 – hardlink-backup-server.sh.

```
1  #!/bin/sh
2
3  #nazwa backupu
4  name=server
5  #zdalny użytkownik ssh do backupu
6  user=root
7  #funkcja skryptu (okresla rozne nazwy)
8  func=backup
9  #ip zdalnego serwera z ktorego pobierane są dane do backupu
10 ip=192.168.1.1
11 #listy wykluczenia katalogów w osobnym pliku
12 exclude=exclude-list.txt
13 #ścieżka do katalogu a plikami dla server monitor'a
14 infopath=~/.info
15 #katalog przeznaczenia backupów
16 destination=/home
17 #wstaw date w formacie RRRR-MM-DD z dnia z ktorego istnieje
18 #ostatni a kopia
19 #jeśli backup z dnia poprzedniego nie istnieje pozostaw wartość
20 #zmiennej dataminus pusta np:
21 #dataminus=2015-03-31
22 dataminus=
```

```
23 #poniżej nie należy edytować
24 fulldatetime=$(/bin/date "+%Y-%m-%d %H:%M:%S")
25 server=$(hostname -s)
26 infofile=${func}-${name}
27 backupdir=${destination}/${(echo ${func} | tr '[:lower:]' \
28 '[:upper:]')}/${(echo ${name} | tr '[:lower:]' '[:upper:]')}
29 data=$(date +%Y-%m-%d)
30 [ -z "${dataminus}" ] && dataminus=$(date -d "-1 day" +%Y-%m-%d)
31 path=${backupdir}/${func}-${name}-${data}
32 previous=${backupdir}/${func}-${name}-${dataminus}
33 [ ! -e "${path}" ] && mkdir -p ${path}
34 [ ! -e "${infopath}" ] && mkdir -p ${infopath}
35
36 if [ -e "${previous}" ]; then
37     if [ -e "${path}.log" ]; then
38         /usr/bin/rsync -avzP --ipv4 --exclude-from ${exclude} \
39 --delete-during --delete-excluded --link-dest=${previous} \
40 ${user}@${ip}:/ * ${path}/ >> ${path}.log
41     else
42         /usr/bin/rsync -avzP --ipv4 --exclude-from ${exclude} \
43 --delete-during --delete-excluded --link-dest=${previous} \
44 ${user}@${ip}:/ * ${path}/ > ${path}.log
45     fi
46     echo "${fulldatetime} Backup różnicowy ${name} wykonano na \
47 ${server}..." | tee ${infopath}/${infofile}
48 else
49 echo "Backup z dnia poprzedniego ${name} nie istnieje" | tee \
    ${infopath}/${infofile}.txt
50 fi
```

Plik `exclude-list.txt` przedstawiony na listingu (Listing 4), jest niezbędny do prawidłowej pracy powyższych skryptów. Eliminuje on tworzenie kopii z tymczasowych, systemowych i wirtualnych systemów plików, które to pliki są tworzone przez system automatycznie i nie są niezbędne w kopii do przywrócenia działania serwera po jego awarii. Niektóre z plików są reprezentacjami urządzeń, w systemie, więc ich zabezpieczenie do kopii bezpieczeństwa nie jest potrzebne [7].

Listing 4 – `exclude-list.txt`.

```
1 /dev/
2 /proc/
```

```
3 /sys/
4 /tmp/
5 /run/
6 /mnt/
7 /media/
8 /lost+found/
```

Jak już zostało wcześniej wspomniane skrypt należy uruchomić z demona cron, poprzez wpis w pliku `/etc/crontab`, dzięki czemu kopia będzie wykonywana każdego dnia o godzinie np.: 1:00, a informacja o wykonanej kopii zostanie automatycznie przesłana do administratora:

```
0 1 * * *      root /root/hardlink-backup-server.sh | \
mail -s "Backup log" email@administratora.pl
```

Następny ważny skrypt, chroniący serwer to skrypt `firewall_ipv4.sh` (zapora sieciowa - Listing 5) [16]. Skrypt uruchamiany jest z pliku `/etc/rc.local` podczas startu systemu, co zostanie zaprezentowane poniżej. Skrypt `firewall_ipv4.sh` (oraz `firewall_ipv6.sh`) ustawia najważniejsze parametry związane z zaporą sieciową, jak ochrona przez atakami, skanowaniem, syn-flood (DoS), dozwolone i zabronione listy adresów IP:

Listing 5 – firewall – zapora sieciowa ipv4.

```
1 #!/bin/bash
2 hostname=$(/bin/hostname)
3 server_main=serwer
4 server_sec=serwer2
5
6 path_ipv4_local=/root/scripts/firewall/IPV4/BLACK_LISTS_LOCAL
7 path_ipv6_local=/root/scripts/firewall/IPV6/BLACK_LISTS_LOCAL
8 path_ipv4_world=/root/scripts/firewall/IPV4/BLACK_LISTS_WORLD
9 path_ipv6_world=/root/scripts/firewall/IPV6/BLACK_LISTS_WORLD
10 hand=bruteforce_hand_ip_list.txt
11 full=bruteforce_full_ip_list.txt
12 if [ "${hostname}" = "${server_main}" ]; then
13     auto=bruteforce_fail2ban_ip_list.txt
14 else
15     auto=bruteforce_fail2ban_ip_list_other.txt
```



```
16 fi
17
18 # Czyszczenie regul firewalla
19 /sbin/iptables -F
20 /sbin/iptables -X
21 /sbin/iptables -F -t nat
22 /sbin/iptables -X -t nat
23 /sbin/iptables -F -t filter
24 /sbin/iptables -X -t filter
25
26 #echo 1 > /proc/sys/net/ipv4/ip_forward > /dev/null 2>&1
27 #echo "1" > /proc/sys/net/ipv4/icmp_echo_ignore_all
28 echo "1" > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts \
29 > /dev/null 2>&1
30 echo "1" > /proc/sys/net/ipv4/icmp_ignore_bogus_error_responses \
31 > /dev/null 2>&1
32 echo "1" > /proc/sys/net/ipv4/conf/all/log_martians \
33 > /dev/null 2>&1
34 echo "0" > /proc/sys/net/ipv4/conf/all/accept_source_route \
35 > /dev/null 2>&1
36 echo "0" /proc/sys/net/ipv4/conf/all/accept_redirects \
37 > /dev/null 2>&1
38 echo "1" /proc/sys/net/ipv4/conf/all/rp_filter > /dev/null 2>&1
39 echo "1" > /proc/sys/net/ipv4/tcp_syncookies > /dev/null 2>&1
40 echo "1" > /proc/sys/net/ipv4/conf/all/log_martians \
41 > /dev/null 2>&1
42
43 if [ "$1" = "stop" ]
44 then
45     /sbin/iptables -P INPUT ACCEPT
46     /sbin/iptables -P FORWARD ACCEPT
47     /sbin/iptables -P OUTPUT ACCEPT
48 echo "0" > /proc/sys/net/ipv4/icmp_echo_ignore_all \
49 > /dev/null 2>&1
50 echo "0" > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts \
51 > /dev/null 2>&1
52 echo "0" > /proc/sys/net/ipv4/icmp_ignore_bogus_error_responses \
53 > /dev/null 2>&1
54 echo "0" > /proc/sys/net/ipv4/conf/all/log_martians \
55 > /dev/null 2>&1
56 echo "1" > /proc/sys/net/ipv4/conf/all/accept_source_route \
57 > /dev/null 2>&1
```

```
58 echo "1" > /proc/sys/net/ipv4/conf/all/accept_redirects \
59 > /dev/null 2>&1
60 echo "0" > /proc/sys/net/ipv4/conf/all/rp_filter \
61 > /dev/null 2>&1
62 echo "1" > /proc/sys/net/ipv4/tcp_syncookies \
63 > /dev/null 2>&1
64 echo "0" > /proc/sys/net/ipv4/conf/all/log_martians \
65 > /dev/null 2>&1
66     exit
67 fi
68
69 /sbin/iptables -P INPUT DROP
70 /sbin/iptables -P FORWARD DROP
71 /sbin/iptables -P OUTPUT ACCEPT
72
73 /sbin/iptables -A INPUT -i lo -j ACCEPT
74 /sbin/iptables -A INPUT -s 1.1.1.1/24 -j ACCEPT
75
76 if [ "${hostname}" = "${server_main}" ]; then
77 /sbin/iptables -A INPUT -s 2.2.2.2 -j ACCEPT
78 fi
79
80 if [ "${hostname}" = "${server_sec}" ]; then
81 /sbin/iptables -A INPUT -s 3.3.3.3 -j ACCEPT
82 fi
83
84 /sbin/iptables -A INPUT -p icmp -s 37.187.62.250 -j ACCEPT
85
86 /sbin/iptables -A INPUT -p tcp --dport 113 -j REJECT \
87 --reject-with icmp-port-unreachable
88
89 # Ochrona przed atakami
90 /sbin/iptables -A INPUT -p icmp --icmp-type echo-request \
91 -m limit --limit 2/s -j ACCEPT # Ping of death
92 /sbin/iptables -A INPUT -p icmp --icmp-type echo-request \
93 -j REJECT --reject-with icmp-host-unreachable
94 /sbin/iptables -A INPUT -m conntrack --ctstate NEW \
95 -p tcp --tcp-flags SYN,RST,ACK,FIN,URG,PSH ACK -j DROP
96 /sbin/iptables -A INPUT -m conntrack --ctstate NEW \
97 -p tcp --tcp-flags SYN,RST,ACK,FIN,URG,PSH FIN -j DROP \
98 /sbin/iptables -A INPUT -m conntrack --ctstate NEW \
99 -p tcp --tcp-flags SYN,RST,ACK,FIN,URG,PSH FIN,URG,PSH \
```

```
100 -j DROP
101 /sbin/iptables -A INPUT -m conntrack --ctstate INVALID \
102 -p tcp ! --tcp-flags SYN,RST,ACK,FIN,PSH,URG \
103 SYN,RST,ACK,FIN,PSH,URG -j DROP
104
105 /sbin/iptables -N syn-flood
106 /sbin/iptables -A INPUT -p tcp --syn -j syn-flood
107 /sbin/iptables -A syn-flood -m limit --limit 100/s \
108 --limit-burst 150 -j RETURN
109 /sbin/iptables -A syn-flood -m limit --limit 100/s \
110 --limit-burst 150 -j LOG --log-prefix "SYN-flood: "
111 /sbin/iptables -A syn-flood -j DROP
112
113 /sbin/iptables -A INPUT -m state --state ESTABLISHED,RELATED \
114 -j ACCEPT
115
116 #IPSET - masowo blokowane adresy IPV4
117 if [ "${hostname}" = "${server_main}" ]; then
118 /sbin/iptables -I INPUT -m set --match-set denyset_ipv4 src -j DROP
119 fi
120
121 if [ "${hostname}" = "${server_sec}" ]; then
122 for ipaddress in $(egrep -h -v -E "^#|^$"
    ${path_ipv4_local}/${full}); do
123 /sbin/iptables -I INPUT -s ${ipaddress} -j DROP
124 done
125 fi
126
127 #Dozwolony ruch (porty)
128 if [ "${hostname}" = "${server_main}" ]; then
129 /sbin/iptables -A INPUT -m state --state NEW -m multiport -p tcp \
130 --dports 20,21,22,25,53,80,110,143,443,465,587,993,995,2222 \
131 -j ACCEPT
132 /sbin/iptables -A INPUT -p udp --dport 53 -j ACCEPT
133 fi
134
135 if [ "${hostname}" = "${server_sec}" ]; then
136 /sbin/iptables -A INPUT -m state --state NEW -m multiport -p tcp \
137 --dports 22,53 -j ACCEPT
138 /sbin/iptables -A INPUT -p udp --dport 53 -j ACCEPT
139 fi
```

Istnieje również analogiczny skrypt zapory sieciowej dla ochrony w warstwie protokołu IP wersja 6 (ipv6). Jest on załączony na płycie CD do niniejszej pracy, jednakże ze względu na bardzo duże podobieństwo i niewielkie różnice nie będzie osobno omawiany.

W skrypcie zapory sieciowej ipv4 oraz ipv6 (Listing 5) kluczowe jest polecenie:

```
„iptables -A INPUT -m state --state NEW...”
```

Definiuje ono listę otwartych portów serwera z działającymi usługami, do których dostęp jest otwarty. Dostęp ten jednak mimo wszystko nie powinien być otwarty dla adresów IP, z których przeprowadzano w przeszłości ataki na serwer. O blokadzie takich adresów decyduje wiersz:

```
„iptables -I INPUT -m set --match-set denyset_ipv4 src -j DROP”.
```

Reguły „denyset” tworzone są w skrypcie ipset.sh, który zostanie omówiony dalej.

Plik /etc/rc.local – jest głównym plikiem rozruchowym, z którego jednorazowo podczas startu można uruchamiać różnego typu programy i usługi, których włączenie nie jest zdefiniowane w innych miejscach systemu jak np.: „init” lub „systemd”.

Zawartość pliku /etc/rc.local przedstawia poniższy listing (Listing 6):

Listing 6 – skrypty startowe.

```
1 echo "Serwer serwea został zrestartowany" | mail -s \  
2 "Serwer - nastąpił restart" admin@domenaadmina.pl  
3 ipset create denyset hash:net family inet -quiet  
4 ipset create deny6set hash:net family inet6 -quiet  
5 /root/scripts/firewall/ipset.sh  
6 exit 0
```

W powyższym skrypcie (Listing 6) tworzona jest lista „denyset” (oraz deny6set dla ipv6), która posłuży do przechowywania zabronionych adresów IP. Adresy te nie będą miały dostępu do serwera. Lista tych adresów zostanie użyta w następnym skrypcie ipset.sh przedstawionym na listingu Listing 9.

Część usług oraz skryptów uruchamiana jest z CRON – uniksowego demona zajmującego się okresowym wywoływaniem innych programów. Plik konfiguracyjny usługi CRON znajduje się w /etc/crontab. Oto jego zawartość (Listing 7):

Listing 7 – crontab.

```
1 #firewall  
2 0 * * * * root /root/scripts/firewall/autofail2ban.sh  
3 5 * * * * root /root/scripts/firewall/ipset.sh  
4 1 * * * * root /root/scripts/rsync_firewall_secondary.sh
```

```
5
6 #sprawdzajace
7 0,30 * * * * root /root/scripts/named-checkconf.sh
8 0/5 * * * * root /root/scripts/pingtest-secondary.sh
9 0/5 * * * * root /root/scripts/pingtest-ipv6_home_router.sh
10 0/5 * * * * root /root/scripts/pingtest-ipv6_home_net.sh
11 * * * * * root /root/scripts/onekiller.sh
```

Na powyższym listingu (Listing 7), przedstawiono wpisy określające w jaki sposób z demona CRON uruchamiane mają być skrypty modyfikujące pracę zapory sieciowej oraz skrypty testujące. Oprócz tego uruchamiany jest skrypt pilnujący limitów procesora dla poszczególnych użytkowników skryptu „onekiller.sh” (Rysunek 4.2).

Skrypt autofail2ban.sh – odpowiada za automatyczne tworzenie „czarnych” list adresów IP potrzebnych skryptom zapory sieciowej do automatycznego blokowanie adresów IP, z których wcześniej dokonywano ataków na serwer lub prób włamania do usług serwera. Listing 8 przedstawia skrypt i jego zasadę działania:

Listing 8 – autofail2ban.sh.

```
1 #!/bin/sh
2
3 hostname=$(/bin/hostname)
4 local_ipv4_ips="(1.1.1.1|2.2.2.2) "
5 local_ipv6_ips="(2000:45d0:32:b00::abc|2000:55d0:33:300::abc) "
6
7 path_ipv4_local=/root/scripts/firewall/IPV4/BLACK_LISTS_LOCAL
8 path_ipv6_local=/root/scripts/firewall/IPV6/BLACK_LISTS_LOCAL
9 path_ipv4_world=/root/scripts/firewall/IPV4/BLACK_LISTS_WORLD
10 path_ipv6_world=/root/scripts/firewall/IPV6/BLACK_LISTS_WORLD
11 hand=bruteforce_hand_ip_list.txt
12 full=bruteforce_full_ip_list.txt
13 if [ "${hostname}" = "servea.pl" ]; then
14     auto=bruteforce_fail2ban_ip_list.txt
15 else
16     auto=bruteforce_fail2ban_ip_list_other.txt
17 fi
18
19 #IPV4
20 #Fail2ban - zbieranie IP (ipv4)
21 /bin/cat /var/log/fail2ban.log /var/log/fail2ban.log.1 \
```

```
22         | /bin/egrep -E "(\ Ban\ )" \
23         | /usr/bin/awk '{print $7}' \
24         | /usr/bin/sort | /usr/bin/uniq \
25         | /bin/egrep -v -E "${local_ipv4_ips}" \
26         > ${path_ipv4_local}/${auto}
27
28 /bin/zcat /var/log/fail2ban.log.*.gz \
29         | /bin/egrep -E "(\ Ban\ )" \
30         | /usr/bin/awk '{print $7}' \
31         | /usr/bin/sort \
32         | /usr/bin/uniq \
33         | /bin/egrep -v -E "${local_ipv4_ips}" \
34         >> ${path_ipv4_local}/${auto}
35
36 #Exim reject log
37 if [ -e /var/log/exim ]; then
38 #echo "exim istnieje"
39 /bin/cat /var/log/exim/rejectlog /var/log/exim/rejectlog.1 \
40         | /bin/grep "login authenticator failed" \
41         | /bin/grep "535 Incorrect authentication data" \
42         | /bin/egrep -o '[[[:digit:]]{1,3}\. [[[:digit:]]{1,3}\
43 \. [[[:digit:]]{1,3}\. [[[:digit:]]{1,3}' \
44         | /usr/bin/sort \
45         | /usr/bin/uniq -c \
46         | /usr/bin/awk '{if ($1 > 99) print $2}' \
47         | /bin/egrep -v -E "${local_ipv4_ips}" \
48         | /usr/bin/sort \
49         >> ${path_ipv4_local}/${auto}
50
51 /bin/zcat /var/log/exim/rejectlog.*.gz \
52         | /bin/grep " login authenticator failed" \
53         | /bin/grep "535 Incorrect authentication data" \
54         | /bin/egrep -o '[[[:digit:]]{1,3}\. [[[:digit:]]{1,3}\
55 \. [[[:digit:]]{1,3}\. [[[:digit:]]{1,3}' \
56         | /usr/bin/sort \
57         | /usr/bin/uniq -c \
58         | /usr/bin/awk '{if ($1 > 99) print $2}' \
59         | /bin/egrep -v -E "${local_ipv4_ips}" \
60         | /usr/bin/sort \
61         >> ${path_ipv4_local}/${auto}
62 fi
63
```

```
64 #Dovecot
65 /bin/zcat /var/log/syslog.*.gz \
66     | /bin/grep dovecot \
67     | /bin/grep login \
68     | /bin/egrep "(no auth attempts|auth failed)" \
69     | /bin/egrep -o 'rip=[[[:digit:]]{1,3}\.[:digit:]]{1,3}\
70 \.[:digit:]]{1,3}\.[:digit:]]{1,3}' \
71     | /usr/bin/cut -f2 -d= \
72     | /usr/bin/sort \
73     | /usr/bin/uniq -c \
74     | /usr/bin/awk '{if ($1 > 99) print $2}' \
75     | /bin/egrep -v -E "${local_ipv4_ips}" \
76     | /usr/bin/sort \
77     >> ${path_ipv4_local}/${auto}
78
79 #Pure-ftp
80 /bin/zcat /var/log/syslog.*.gz \
81     | /bin/grep "Authentication failed for user" \
82     | /bin/egrep -o '[[[:digit:]]{1,3}\.[:digit:]]{1,3}\
83 \.[:digit:]]{1,3}\.[:digit:]]{1,3}' \
84     | /usr/bin/sort \
85     | /usr/bin/uniq -c \
86     | /usr/bin/awk '{if ($1 > 99) print $2}' \
87     | /bin/egrep -v -E "${local_ipv4_ips}" \
88     | /usr/bin/sort \
89     >> ${path_ipv4_local}/${auto}
90
91 #Proftpd
92 if [ -e /var/log/proftpd ]; then
93 #echo "proftpd istnieje"
94 /bin/cat /var/log/proftpd/auth.log \
95     | /bin/grep "\"PASS (hidden)\" 530" \
96     | /usr/bin/awk '{print $3}' \
97     | /usr/bin/sort \
98     | /usr/bin/uniq -c \
99     | /usr/bin/awk '{if ($1 > 99) print $2}' \
100    | /bin/egrep -v -E "${local_ipv4_ips}" \
101    | /usr/bin/sort \
102    >> ${path_ipv4_local}/${auto}
103
104 /bin/zcat /var/log/proftpd/auth.log.*.gz \
105     | /bin/grep "\"PASS (hidden)\" 530" \
```

```
106      | /usr/bin/awk '{print $3}' \
107      | /usr/bin/sort \
108      | /usr/bin/uniq -c \
109      | /usr/bin/awk '{if ($1 > 99) print $2}' \
110      | /bin/egrep -v -E "${local_ipv4_ips}" \
111      | /usr/bin/sort \
112      >> ${path_ipv4_local}/${auto}
113 fi
114
115 #Polaczenie IP v4 wprowadzana ręcznie
116 /bin/cat ${path_ipv4_local}/${auto} ${path_ipv4_local}/${hand} \
117      | /bin/egrep -v -E "${local_ipv4_ips}" \
118      | /bin/sed '/^$/d' \
119      | /usr/bin/sort \
120      | /usr/bin/uniq \
121      > ${path_ipv4_local}/${full}
122
123 #IPV6
124 #Polaczenie IP v6 wprowadzana ręcznie
125 /bin/cat ${path_ipv6_local}/${auto} ${path_ipv6_local}/${hand} \
126      | /bin/egrep -v -E "${local_ipv6_ips}" \
127      | /bin/sed '/^$/d' \
128      | /usr/bin/sort \
129      | /usr/bin/uniq \
130      > ${path_ipv6_local}/${full}
```

Jak widać na powyższym listingu (Listing 8), skrypt wymaga do działania usługi o nazwie fail2ban dostępnej w systemie Linux (Debian). Skrypt korzysta z logów tej usługi oraz z innych logów i dzienników systemowych, do pozyskiwania adresów IP, które dokonywały ataków na usługi serwera.

Następnym ważnym skryptem niezbędnym do prawidłowej pracy zapory sieciowej jest skrypt ipset.sh (Listing 9). Jest to skrypt do automatycznego blokowania bardzo dużych list adresów IP, z którymi zaporą sieciową Iptables/Ip6tables nie byłaby sobie w stanie poradzić bezpośrednio. Skrypt ten jest bardzo istotny dla serwera, ponieważ ogranicza dostęp do serwera z niedozwolonych adresów IP. Jego główną zaletą jest rozmiar listy, która dzięki mechanizmowi tablicy haszującej może być ogromna i dzięki temu nie wpływa znacząco na obciążenie serwera a co za tym idzie na skuteczność działania zapory sieciowej, która w innym wypadku była by słaba (mało wydajna i znacznie obciążająca procesor serwera):

Listing 9 – *ipset.sh*.

```
1  #!/bin/sh
2  hostname=$(/bin/hostname)
3
4  path_ipv4_local=/root/scripts/firewall/IPV4/BLACK_LISTS_LOCAL
5  path_ipv6_local=/root/scripts/firewall/IPV6/BLACK_LISTS_LOCAL
6  path_ipv4_world=/root/scripts/firewall/IPV4/BLACK_LISTS_WORLD
7  path_ipv6_world=/root/scripts/firewall/IPV6/BLACK_LISTS_WORLD
8  hand=bruteforce_hand_ip_list.txt
9  full=bruteforce_full_ip_list.txt
10 if [ "${hostname}" = "servea.pl" ]; then
11     auto=bruteforce_fail2ban_ip_list.txt
12 else
13     auto=bruteforce_fail2ban_ip_list_other.txt
14 fi
15
16 if [ "${hostname}" = "servea.pl" ]; then
17 #ipset destroy denyset...
18 #echo > /etc/ipset.conf
19
20 #Takze w /etc/rc.local
21 ipset create denyset_ipv4 hash:net family inet -quiet
22 ipset create denyset_ipv6 hash:net family inet6 -quiet
23
24 # Dodanie adresow IPV4 do zestawu blokowanych:
25 for ipaddress in $(egrep -h -v -E "^#|^$" \
26 ${path_ipv4_world}/*.txt); do ipset add denyset_ipv4 \
27 $ipaddress -quiet; done
28 for ipaddress in $(egrep -h -v -E "^#|^$" \
29 ${path_ipv4_local}/${full}); do ipset add denyset_ipv4 \
30 $ipaddress -quiet; done
31
32 # Dodanie adresow IPV6 do zestawu blokowanych:
33 for ipaddress in $(egrep -h -v -E "^#|^$" \
34 ${path_ipv6_world}/*.txt); do ipset add denyset_ipv6 \
35 $ipaddress -quiet; done
36 for ipaddress in $(egrep -h -v -E "^#|^$" \
37 ${path_ipv6_local}/${full}); do ipset add denyset_ipv6 \
38 $ipaddress -quiet; done
39
```

```
40 #Zapisanie zestawow blokowanych sieci na stale
41 ipset save > /etc/ipset.conf
42 fi
43
44 #Restart firewalla
45
46 /etc/init.d/firewall_ipv4 stop
47 /etc/init.d/firewall_ipv4 start
48 /etc/init.d/firewall_ipv6 stop
49 /etc/init.d/firewall_ipv6 start
```

Skrypt ten odpowiedzialny jest za umieszczenie na czarnej liście adresów IP, które mają być blokowane przez zaporę sieciową. Czarne listy zostały stworzone poprzez skrypt `autofail2ban.sh` (Listing 8), a następnie użyte przez reguły skryptu zapory sieciowej (Listing 5), i o którym już zostało wcześniej powiedziane.

Skrypt do sprawdzania stref DNS „`named-checkconf.sh`” (Listing 10), to bardzo prosty skrypt ale jakże nieoceniony w hostingu. W przypadku dodawania domen przez użytkowników Panelu serwera „DirectAdmin” może dojść do błędów. Celem tego skryptu jest sprawdzenie poprawności pliku konfiguracyjnego odpowiadającego za konfigurację stref DNS. Jeśli zostanie wykryty problem (błąd w strefie DNS), automatycznie zostanie wysłany e-mail do administratora z powiadomieniem o zaistniałym błędzie. Dzięki czemu można ustrzec się przed poważną awarią funkcjonowania wielu usług internetowych korzystających z określonego serwera DNS:

Listing 10 – sprawdzanie stref DNS.

```
1  #!/bin/sh
2
3  func=named-checkconf
4  infopath=/root/info
5  emails=email@admina.pl
6  host=$(/bin/hostname)
7  nerror=$(/usr/sbin/${func})
8  ncheck=$(echo $? )
9
10 if [ "${ncheck}" -ne "0" ]; then
11     echo "${nerror}" | mail -s "Na serwerze ${host} \
12 wykryto błąd konfiguracji: ${func}" ${emails}
13     mkdir -p ${infopath}; echo "${nerror}" \
```

```
14 > ${infopath}/${func}
15 fi
```

Poniżej (Listing 11) przedstawiono skrypt do zrzucania informacji o bieżącym obciążeniu CPU do plików. Informacje, które można uzyskać dzięki temu skryptowi są bardzo ważne do pełnego zdiagnozowania zdarzeń, które już minęły, a dokonały zaburzeń w pracy serwera podczas nieobecności administratora. Skrypt należy uruchamiać z CRON, co minutę:

Listing 11 – topdump.sh.

```
1  #!/bin/sh
2
3  path=/root/TOPDUMP
4  mkdir -p ${path}
5
6  for i in $(seq 1 5); do
7    data=$(/bin/date +%Y-%m-%d-%H-%M-%S)
8    /usr/bin/top -n 1 > ${path}/top_${data}.txt
9    sleep 10
10 done
```

Listing 12 przedstawia wpis w pliku `/etc/crontab`, który uruchamia skrypt przedstawiony wcześniej (Listing 11), za pomocą demona CRON:

Listing 12 – topdump.sh w /etc/crontab.

```
1  * * * * * root /root/scripts/topdump.sh > /dev/null 2>&1
2  0 0 * * * root /usr/bin/find /root/TOPDUMP/ -type f \
3  -mtime +3 -delete > /dev/null 2>&1
```

Drugi wiersz w powyższym wpisie (Listing 12) to polecenie niezbędne do usuwania starych danych o historycznym obciążeniu serwera, które nie są już potrzebne, w celu zwolnienia miejsca na dysku serwera.

Następnym przydatnym skryptem, jest przedstawiony poniżej (Listing 13) służący do pilnowania działania usług (może być stosowany do pilnowania działania takich usług jak: Apache, Exim, Bind, Dovecot i innych). W przypadku gdyby usługa serwera przestała działać skrypt ten wystartuje ją ponownie. Skrypt musi być uruchamiany raz na minutę

z demona CRON. Za każdym razem sprawdza on, czy usługa odpowiada na zapytania po protokole TCP/IP, w przeciwnym razie restartuje ją:

Listing 13 – check-service.sh.

```
1  #!/bin/sh
2
3  IP=127.0.0.1
4  port=25
5  service=exim4
6
7  nc -z -w5 ${IP} ${port}
8  if [ "$?" -ne "0" ]; then
9    service ${service} restart
10 echo "Zrestartowano usługę ${service} ponieważ nie odpowiadała"
11 fi
```

Powyższy skrypt po zmianie nazwy usługi oraz numeru portu może być dostosowany do monitorowania większości usług serwera, które działają w oparciu o protokół TCP/IP.

Następny skrypt `check_hdd_usage.sh` (Listing 14) służy do pomiaru obciążenia dysku lub macierzy dyskowej w systemie. Skrypt bada bieżące uśrednione obciążenie dysku z np.: 5 sekund, a następnie wysyła informację o tym do pliku, który będzie monitorowany przez aplikację „ServerMonitor”. W przypadku przekroczenia limitu określonego w zmiennej „threshold”, dodatkowo wysłane zostanie powiadomienie przy pomocy wiadomości e-mail do administratora serwera.

Listing 14 – check_hdd_usage.sh.

```
1  #!/bin/sh
2
3  emails=email@admina.pl
4  hdd=sda
5  func=hdd-usage
6  infopath=/root/info
7  threshold=70
8  interval=1
9
10 host=$(/bin/hostname)
11 my_pid=$(echo $$)
12 script_name=$(echo $0 | grep -o '^[^/]*$')
```

```
13
14 if /bin/ps ax | /bin/grep ${script_name} | /bin/grep -v grep \
15 | /bin/grep -v ${my_pid} ; then
16     exit
17 else
18     for i in $(seq 1 11); do #co 5 sekund
19         hdd_usage=$(/usr/bin/iostat -d -x ${hdd} 5 2 | \
20 /bin/grep ${hdd} | /usr/bin/tail -n 1 | /usr/bin/awk '{print $14}')
21         /bin/echo "${hdd_usage}" > ${infopath}/${func}.txt
22         int_hdd_usage=$(/bin/echo ${hdd_usage} \
23 | /usr/bin/cut -f1 -d,)
24         if [ "${int_hdd_usage}" -gt "${threshold}" ]; then
25             data=$(/bin/date "+%Y-%m-%d %H:%M:%S")
26             /bin/echo "${data} Avg ${hdd} usage \
27 > ${threshold}" | /usr/bin/mail -s "${data} Avg ${hdd} usage \
28 > ${threshold}" ${emails}
29
30         fi
31         #sleep ${interval}
```

Podobnie do powyższego działa następny skrypt `check_all_snmp.sh` (Listing 15). Zapisuje on dane o średnim obciążeniu CPU (np.: z 1, 5 i 15 minut), CPU IDLE, użyciu RAM, itp. do plików, tekstowych, które to będą następnie pobierane i monitorowane przez aplikację „ServerMonitor”. Skrypt do pracy wymaga skonfigurowanej i uruchomionej usługi SNMP na monitorowanym serwerze [17]. Dla bezpieczeństwa usługę SNMP należy postawić w środowisku izolowanym, np. w chroot [18].

Listing 15 – `check_all_snmp.sh`.

```
1  #!/bin/sh
2
3  infopath=/root/info
4  interval=5
5  host=$(/bin/hostname)
6
7  my_pid=$(echo $$)
8  script_name=$(echo $0 | grep -o '^[^/]*$')
9  /bin/echo "${host}" > ${infopath}/hostname.txt
10 if /bin/ps ax | /bin/grep ${script_name} | /bin/grep -v grep |
    /bin/grep -v ${my_pid} ; then
11     exit
```

```
12 else
13     for i in $(seq 1 11); do #co 5 sekund
14         cpu_load_01m=$(/usr/bin/snmpwalk -v1 -cpublic \
15 localhost .1.3.6.1.4.1.2021.10.1.3.1 | cut -f2 -d'')
16         echo $cpu_load_01m > ${infopath}/cpu-load.txt
17         cpu_load_05m=$(/usr/bin/snmpwalk -v1 -cpublic \
18 localhost .1.3.6.1.4.1.2021.10.1.3.2 | cut -f2 -d'')
19         echo $cpu_load_05m > ${infopath}/cpu-load_05m.txt
20         cpu_load_15m=$(/usr/bin/snmpwalk -v1 -cpublic \
21 localhost .1.3.6.1.4.1.2021.10.1.3.3 | cut -f2 -d'')
22         echo $cpu_load_15m > ${infopath}/cpu-load_15m.txt
23         cpu_idle=$(/usr/bin/snmpwalk -v1 -cpublic \
24 localhost .1.3.6.1.4.1.2021.11.11.0 | cut -f4 -d" ")
25         echo $cpu_idle > ${infopath}/cpu-idle.txt
26         echo $cpu_idle > ${infopath}/cpu-usage.txt
27         ram_used=$(/usr/bin/snmpwalk -v1 -cpublic \
28 localhost .1.3.6.1.4.1.2021.4.6.0 | cut -f4 -d" ")
29         echo $ram_used > ${infopath}/ram-usage.txt
30         sleep ${interval}
31     done
32 fi
```

Powyższy skrypt z łatwością można rozszerzyć o monitorowanie dodatkowych parametrów serwera, gdyż skrypt pobiera je z aplikacji SNMP dla serwera Linux Debian.

Następny skrypt „onekiller.sh” (Rysunek 4.1, Rysunek 4.3, Rysunek 4.3) służy do pilnowania limitów wykorzystania procesora (CPU) poprzez każdego użytkownika indywidualnie. W przypadku kiedy użytkownik przekracza przydzielony mu limit (lub limit domyślny), to poszczególne procesy PHP danego użytkownika będą zakańczane przez skrypt (w systemie poprzez polecenie „kill”). Wcześniej jednak zostanie obniżony ich priorytet systemowy, aby zminimalizować obciążenie i dać procesom PHP szansę działania z mniejszą szkodą, poprzez wyeliminowanie możliwości zawłaszczenia całej mocy obliczeniowej serwera przez dany proces. Jeśli w tym czasie obciążenie generowane przez poszczególne procesy php-fpm (lub php-cgi) spadnie poniżej limitu, to nie zostaną one zatrzymane. W przeciwnym razie skrypt zatrzyma działanie takiego nadmiernie obciążającego procesu.

Skrypt przystosowany jest do pracy z interpretatorem PHP skompilowanym jako fastcgi (fcgid) lub php-fpm, gdyż w tych trybach każdy wątek PHP uruchamiany jest jako osobny proces z dodatkowymi ograniczeniami podwyższającymi bezpieczeństwo.

Zasada działania skryptu polega na jego cyklicznym uruchamianiu z demona CRON raz na minutę w pętli (5 razy co 10 sekund) i sprawdzaniu komendą „ps” lub „top”, które dokonują pomiarów użycia CPU poprzez poszczególnych użytkowników (najbardziej obciążających procesor). Następnie dla każdego z procesów użytkowników przekraczających limit uruchamiana jest funkcja killing(), która co określony czas bada zgodność procesu z jego limitem. W przypadku przekroczenia limitu proces jest zakańczany przez polecenie systemowe „kill”.

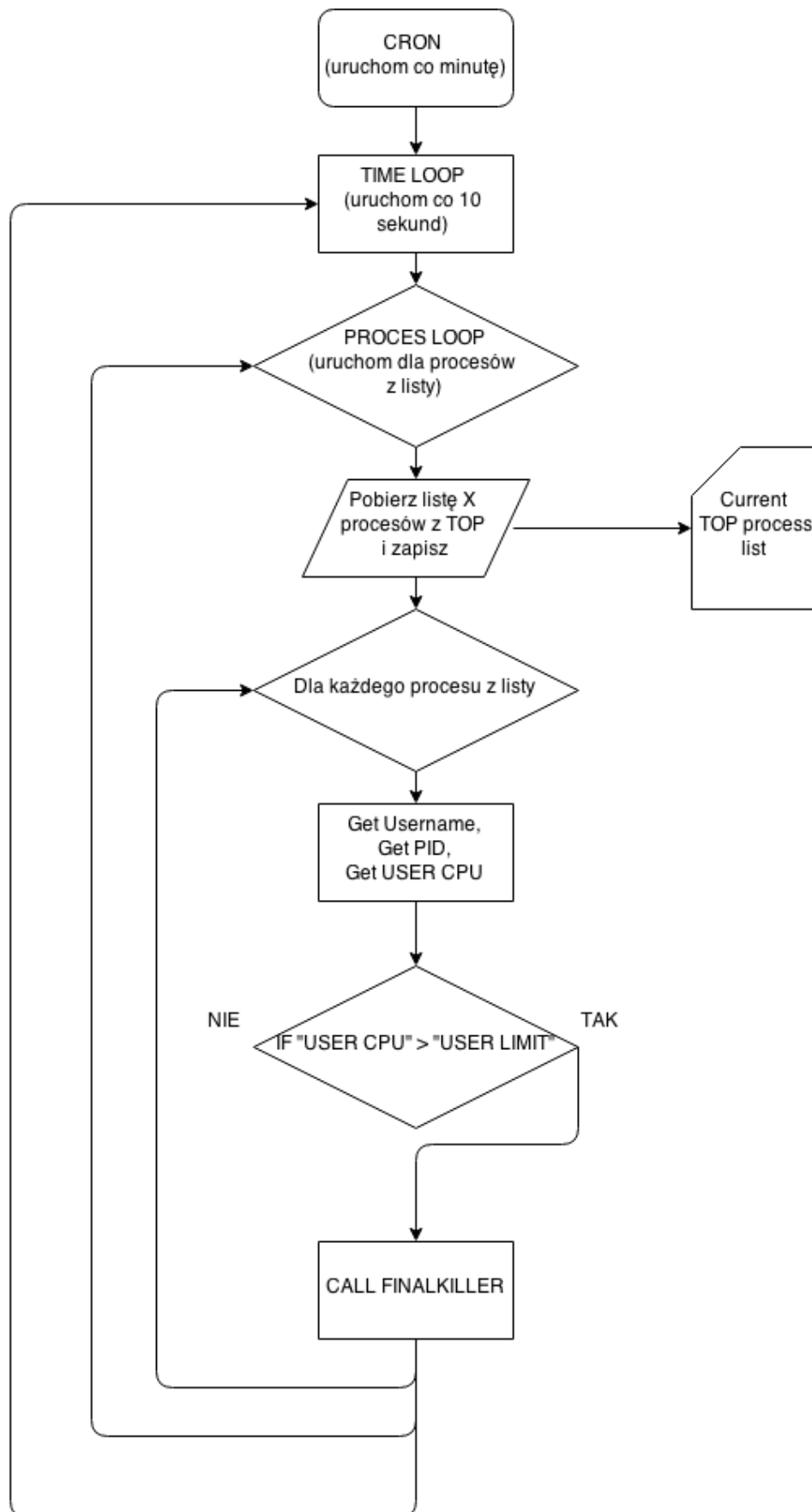
Algorytm działania skryptu przedstawiono na rysunku (Rysunek 4.2). Nie zostały jeszcze zaimplementowane funkcje dla uprzywilejowanych procesów użytkowników (każdy użytkownik, będzie miał prawo do jednego uprzywilejowanego procesu, który będzie mógł chwilowo skorzystać ze 100% mocy obliczeniowej CPU). Jednakże takie dodatkowe funkcje pojawią się na późniejszym etapie rozwoju tego skryptu. Algorytm przedstawia aktualną uproszczoną wersję skryptu onekiller.sh [19].

```

1  #!/bin/bash
2  #uruchamiac z cron'a co minute
3  #np: * * * * root /var/scripts/root/shellkiller.sh > /dev/null 2>&1
4  file1=/tmp/onekiller_1.tmp
5  file2=/tmp/onekiller_2.tmp
6  procesy="ps -l"
7  log=/var/log/onekiller.log
8  lfolder=/root/userscpulimit
9  path=/root/scripts
10 default_cpu_limit=25
11
12 function checking() {
13   for proces in $(ps auxx | awk '{print $3, $2, $1, $11}' | grep $proces) | head -n 35 | awk '{gsub (/\/\./,"",$1); if ($1 > 5) print $1,$3@"$2"}' > $file1;
14   if [ -e $file1 ]; then
15     file=$(cat $file1)
16     ilosc=$(echo "$file" | awk 'NF > 0' | wc -l | awk '{print $1}')
17     if [ $ilosc -ne 0 ]; then
18       for i in $(seq 1 $ilosc); do
19         actual_user_name=$(echo "$file" | awk '{print $2}' | cut -f 1 -d @ | head -n $(i) | tail -n 1)
20         actual_user_pid=$(echo "$file" | awk '{print $2}' | cut -f 2 -d @ | head -n $(i) | tail -n 1)
21         actual_user_cpu=$(echo "$file" | awk '{print $11}' | head -n $(i) | tail -n 1 | cut -f 1 -d '.')
22         if [ -e $lfolder/$actual_user_name ]; then
23           conf_user_cpu_limit=$(cat $(lfolder)/$(actual_user_name))
24         else
25           conf_user_cpu_limit=$default_cpu_limit
26         fi
27         if [ $actual_user_cpu -gt $conf_user_cpu_limit ]; then
28           renice -n +19 -p $(actual_user_pid) > /dev/null 2>&1
29           actual_user_name_pid=$(echo "$file" | awk '{print $2}')
30           data=$(date +%Y-%m-%d %H:%M:%S)
31           $0 $(actual_user_name_pid) &
32           echo "${data} PASSED: $(proces) o PID: $(actual_user_pid) USERNAME: $(actual_user_name) BECAUSE USED:
33             $(actual_user_cpu) % CPU" >> $log
34         fi
35       done
36     fi
37   done
38 }
39
40

```

Rysunek 4.1 – skrypt onekiller.sh – funkcja checking().

Rysunek 4.2 – docelowy algorytm skryptu `onelikker.sh`.


```

8  lfolder=/root/userscpulimit
9  path=/root/scripts
10 default_cpu_limit=25
11
12 function checking() {
13
14 }
15
16 function killing() {
17     for proces in $(ps aux | awk '{print $3, $2, $1, $11}' | grep $proces) | head -n 35 | awk '{gsub (/\/\./, "", $1); if ($1 > 5) print
18         $1,$3"@#$2}"; } > $file2;
19         if [ -e $file2 ]; then
20             file=$(cat $file2)
21             ilosc=$(echo "${file}" | awk 'NF > 0' | wc -l | awk '{print $1}')
22             if [ $(ilosc) -ne 0 ]; then
23                 for i in $(seq 1 $(ilosc)); do
24                     actual_user_name=$(echo "${file}" | awk '{print $2}' | cut -f 1 -d @ | head -n ${i} | tail -n 1)
25                     actual_user_pid=$(echo "${file}" | awk '{print $2}' | cut -f 2 -d @ | head -n ${i} | tail -n 1)
26                     actual_user_cpu=$(echo "${file}" | awk '{print $1}' | head -n ${i} | tail -n 1 | cut -f 1 -d '.')
27                     old_user_name=$(echo $S1 | cut -f 1 -d @)
28                     if [ "${old_user_name}" = "${actual_user_name}" ]; then
29                         old_user_pid=$(echo $S1 | cut -f 2 -d @)
30                         if [ $(old_user_pid) -eq $(actual_user_pid) ]; then
31                             if [ -e $(lfolder)/$(actual_user_name) ]; then
32                                 conf_user_cpu_limit=$(cat $(lfolder)/$(actual_user_name))
33                             else
34                                 conf_user_cpu_limit=$default_cpu_limit
35                             fi
36                             if [ $(actual_user_cpu) -gt $(conf_user_cpu_limit) ]; then
37                                 data=$(date +%Y-%m-%d %H:%M:%S)
38                                 kill -9 $(old_user_pid)
39                                 echo "${data} KILLED: ${proces} o PID: ${old_user_pid} USERNAME:
40                                     ${old_user_name} BECAUSE USED: ${actual_user_cpu} % CPU" >> $log
41                             fi
42                         fi
43                     fi
44                 done
45             fi
46         done
47     fi
48 }
49
50 if [ -z "$1" ]; then
51     for ((j=1; j<=5; j++)); do
52         checking
53         sleep 10
54     done
55 else
56     sleep 9
57     S1=$1
58     killing
59 fi
60

```

Rysunek 4.3 – skrypt onekiller.sh – funkcja killing().

Rysunek 4.3 przedstawia funkcję killing() ze skryptu onekiller.sh, odpowiedzialną za zakańczanie procesów php-fpm, które przekraczają dopuszczalne obciążenie CPU.

Działanie skryptu zostało zweryfikowane za pomocą prostego skryptu testującego (Listing 16) uruchamianego z poziomu interpretera PHP [20] serwera:

Listing 16 – test.php.

```

1  <?php
2  for($i = 0; $i < 1000000000; $i++) {
3      $a += $i;
4  }
5  ?>

```

Skrypt onekiller.sh Przedstawiony (Rysunek 4.3) działa w oparciu i polecenie systemowe „Ps”, za pomocą którego procesy są monitorowane – mierzone jest ich indywidualne użycie CPU:


```
7  infopath=/root/info
8  emails=email@admina.pl
9  host=$(/bin/hostname)
10 nerror=$(/bin/${func} -c 3 ${ip})
11 ncheck=$(echo $? )
12
13 if [ "${ncheck}" -ne "0" ]; then
14     echo "${nerror}" | mail -s "Na serwerze ${host} wykryto \
15 błąd: ${func} ${ip} ${dst}" ${emails}
16     mkdir -p ${infopath}; echo "${nerror}" > \
17 ${infopath}/${func}-${dst}
18 fi
```

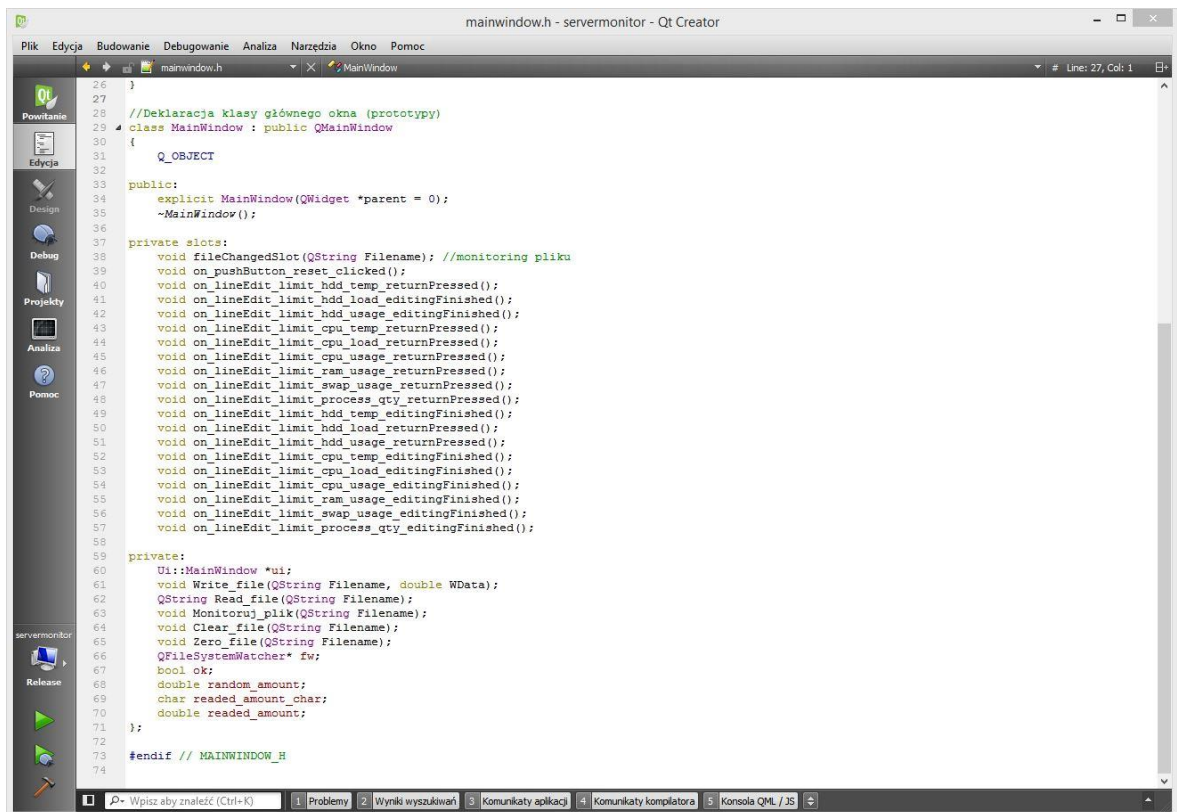
W skład systemu wchodzi inne skrypty. Niektóre są bardzo podobne do wyżej przedstawionych, z drobnymi modyfikacjami, więc ich omawianie nie będzie potrzebne.

Aplikacja Server Monitor – służąca do graficznej prezentacji podstawowych danych o serwerze zbudowana jest w oparciu o jedną dużą klasę opartą o kilka głównych metod przeznaczonych do monitorowania zmian w plikach z danymi o stanie serwera oraz prezentacji tych danych w czasie rzeczywistym.

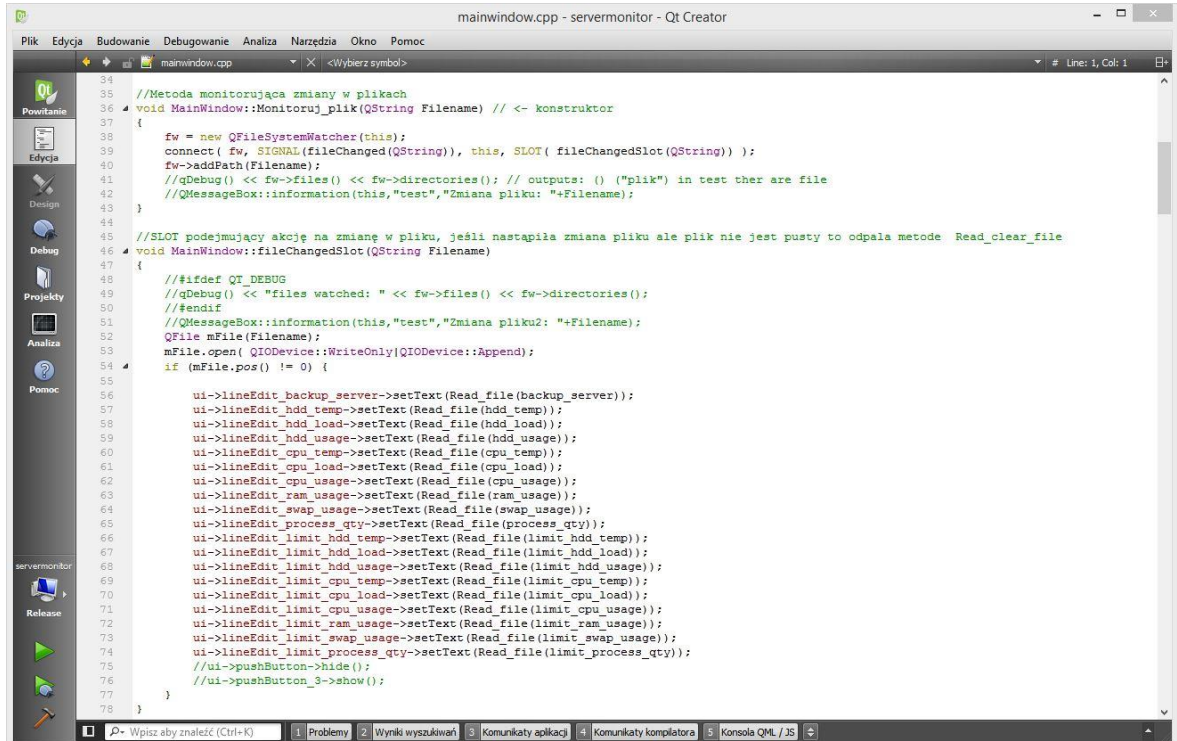
Rysunek 4.4 przedstawia deklarację klasy głównego okna MainWindow, która to zawiera deklaracje metod użytych w klasie. Obok metod występują tzw. sloty – są to metody specjalnego przeznaczenia. Zawsze są typu void i muszą być zadeklarowane w klasie pod słowem kluczowym „slots:”. Sloty mogą posiadać argumenty. W przypadku kiedy zostaną użyte do przekazywania wartości, wartość ta musi zostać podana w nawiasie. Sloty nie posiadają swojego ciała gdyż służą jedynie do przekazywania informacji.

Oprócz deklaracji kilkunastu slotów, klasa główna zawiera zestaw kilku metod służących do monitoringu i przetwarzania (odczytywania i zapisywania) danych w plikach tekstowych. Dzięki czemu dane pobierane z serwera do plików tekstowych mogą być natychmiast przetwarzane, porównywane z limitami i wyświetlane w odpowiedni sposób na pulpicie administratora.

Kluczową klasą QT, niezbędną do działania aplikacji jest QFileSystemWatcher, która dostarcza interfejs przeznaczony do monitorowania plików i katalogów w związku z ich modyfikacjami. Dzięki tej klasie, możliwym jest automatyczne odświeżanie wyświetlanych danych z plików, jeśli te się zmieniły.



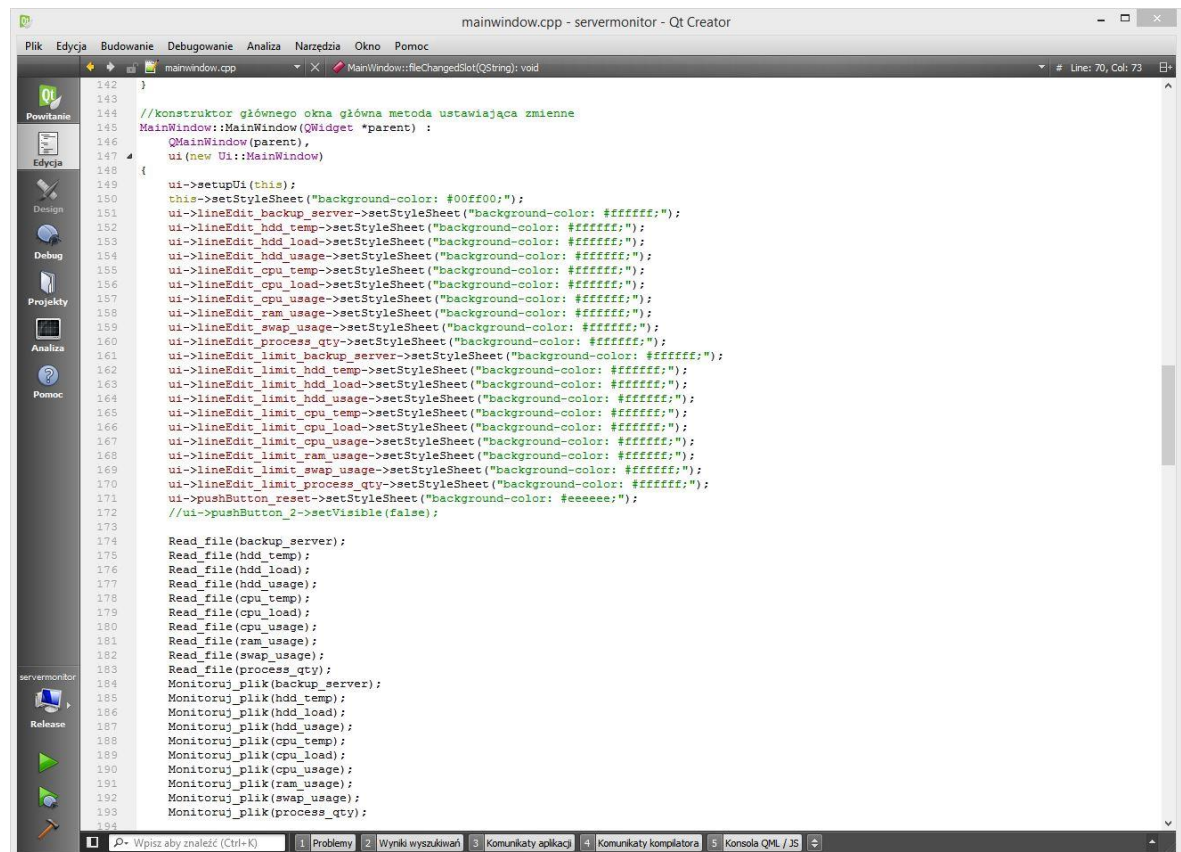
Rysunek 4.4 – mainwindow.h.



Rysunek 4.5 – mainwindow.cpp.

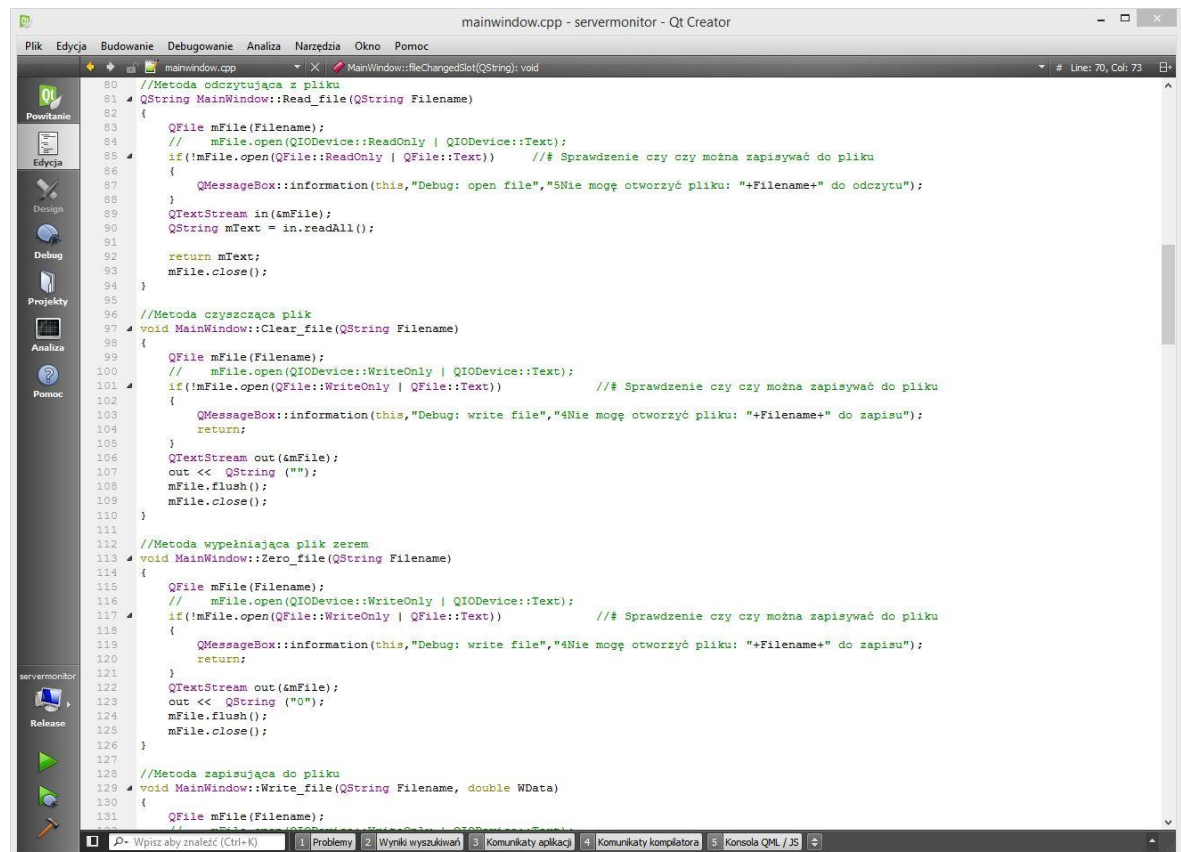
Rysunek 4.5 – przedstawia implementację metod odpowiedzialnych za monitorowanie zmian w plikach, która w momencie wykrycia tych zmian, ustawia właściwe dla nich wartości w widget LineEdit.

Rysunek 4.6 przedstawia implementację konstruktora w klasie MainWindow. Konstruktor ten służy do zainicjowania obiektu okna głównego. Odpowiedzialny jest za ustawienie kolorystyki oraz wstępną inicjalizację procesu odczytywania i monitorowania zmian w plikach.



Rysunek 4.6 – konstruktor.

Rysunek 4.7 przedstawia z kolei implementację metod służących do odczytywania, zapisywania i czyszczenia plików z danymi w klasie MainWindow. Metody te są szczegółowo udokumentowane w komentarzach do kodu źródłowego. Na zwrócenie uwagi zasługuje fakt, że metody te posiadają wbudowane zabezpieczenie, na wypadek, gdyby plik, źródłowy nie istniał lub nie można byłoby go utworzyć, lub zapisywać z powodu braku odpowiednich uprawnień w systemie plików (np. tylko do odczytu). Odpowiedzialną za wyświetlenie komunikatu ostrzegawczego będzie klasa QMessageBox, dzięki której na ekranie pojawi się właściwy komunikat o problemie. Metody te napisane są w bardzo uniwersalny sposób, dzięki czemu wykorzystywane są przez inne metody do obsługi wielu różnych operacji, na różnych plikach i danych w zależności od podanych argumentów.



Rysunek 4.7 – implementacja metod.

4.2. Testowanie programu

W celu przetestowania przeprowadzono szereg testów praktycznych na systemie produkcyjnym na serwerze VPS, pracującym pod kontrolą Debian Gnu/Linux (Wheezy) z zainstalowanym oprogramowaniem DirectAdmin. Zbadano poprawność działania usług (Apache, Dovecot, Exim, Bind9, Proftpd, MySQL, Php-fpm 5.3 i 5.4) pod obciążeniem wygenerowanym sztucznie oraz podczas faktycznych ataków dokonywanych na serwer przez różnego rodzaju boty.

Oprócz tego, do przetestowania systemu użyto skryptów BASH (Listing 18, Listing 19). Skrypty te służą do generowania sztucznego obciążenia CPU, w celu weryfikacji poprawności działania systemu monitoringu oraz skryptu onekiller.sh (Rysunek 4.3).

Listing 18 – count_pi.sh.

```

1  #!/bin/bash
2
3  if [ $# -eq 1 ]
4  then
5      Liczba=$1

```

```
6  else
7      echo -n "Wprowadź ile razy mam wykonać obliczenia: "
8      read Liczba
9  fi
10
11  for((i=0;i<Liczba;i++)); do
12      # scale=50 - precyzja wyświetlania, parametr dla bc
13      # seq x x 100 - precyzja obliczeń, im większe tym większa
    precyzja. Wartość 200 zwiększa czas i obciążenie CPU
14      (echo -n "scale=50;" && (seq 1 2 100 | xargs -I{} echo
    '(16*(1/5)^(1/2)-4*(1/239)^(1/3))' | paste -sd+ | bc -l
15  done
```

Listing 19 – count_fi.sh.

```
1  #!/bin/bash
2
3  if [ $# -eq 1 ]
4  then
5      Liczba=$1
6  else
7      echo -n "Wprowadź ile razy mam wykonać obliczenia: "
8      read Liczba
9  fi
10
11  for((j=0;j<Liczba;j++)); do
12      fib_a=0
13      fib_b=1
14      for((i=0;i<=90;i++)); do
15          fib_n=$((fib_a+fib_b))
16          fib_a=${fib_b}
17          fib_b=${fib_n}
18      done
19      echo -n "Liczba fi wynosi: "
20      echo "scale=40; ${fib_b}/${fib_a}" | bc -l
21  done
```

4.3. Instrukcja instalacji programu

Instalacja systemu ServerMonitor na stanowisku administratora przy uwzględnieniu posiadania skonfigurowanego serwera VPS (lub wielu serwerów VPS) z zainstalowanym oprogramowaniem serwerowym Debian (wraz z niezbędnymi bibliotekami i programami narzędziowymi) z udostępnionymi dowolnymi usługami (Np. Apache, MySQL, PHP) oraz z uruchomionymi skryptami monitorującymi, ogranicza się do następujących kroków:

1. Wygenerowanie kluczy uwierzytelniających SSH na desktopie administratora w systemie Linux lub w konsoli Cygwin (Windows) do logowania samym kluczem (bez hasła). Proces ten został omówiony w rozdziale 4.1 (Listing 1).
2. Umieszczenie skompilowanej aplikacji wraz z całym katalogiem o nazwie „exe” w katalogu domowym dla właściwego środowiska Cygwin (może być ich kilka) np.: C:\cygwin64\home\Konrad\exe i uruchomienie programu z poziomu środowiska Windows.
3. Uruchomienie skryptów `./check_all_snmp.sh` oraz `./check_hdd_usage.sh` z demona CRON na serwerze (na przykład co minutę lub co dowolny czas, w zależności od potrzeb).
4. Uruchomienie z demona CRON (co minutę) na komputerze administratora polecenia (Listing 20):

Listing 20 – crontab klienta.

```
1 * * * * * root for i in $(seq 1 11); do \  
2 rsync -avP root@adresserwera.pl:/root/info/ /home/Konrad/exe/;\   
3 sleep 5; done
```

Dla każdego dodatkowego monitorowanego serwera operacje przedstawione w powyższych punktach należy powtórzyć. Nazwy katalogów „exe” można zastąpić nazwami monitorowanych serwerów.

W zależności od serwera i potrzeb można rozszerzyć bądź zaważyć monitorowane zasoby poprzez odpowiednią modyfikację skryptu `check_all_snmp.sh` (Listing 15).

4.4. Bezpieczeństwo systemu

Głównym atutem systemu ServerMonitor, jest zapewnienie wysokiego poziomu bezpieczeństwa, poprzez zastosowanie oprogramowania `ssh/rsync` do przesyłania danych

między stacją administratora a serwerami. Dzięki temu rozwiązaniu wykluczona została w znacznej mierze, możliwość stworzenia błędu w systemie, który zagrażałby bezpieczeństwu, na etapie komunikacji skryptów z aplikacją. Dzięki zastosowaniu bezpiecznych standardów wymiany danych oraz dzięki użyciu powiadomienia przy pomocy standardowej poczty e-mail, przerzucono ciężar z samodzielnego nadzorowania aplikacji/skryptów pod względem bezpieczeństwa komunikacji, na zewnętrzne zespoły nadzorujące użyte aplikacje (twórców aplikacji rsync/ssh, ssl, deweloperów Debiana oraz niezależnych programistów) [21].

Każdy system jest na tyle bezpieczny, na ile bezpieczne jest jego najsłabsze ogniwo. W tym przypadku dzięki zastosowaniu sprawdzonych i bezpiecznych rozwiązań, największe zagrożenie, czyli możliwość skompromitowania systemu poprzez złamanie zabezpieczeń protokołu komunikacyjnego niemal całkowicie została wyeliminowana. Z drugiej jednak strony uzyskano bardzo wygodny dostęp, do monitorowanych serwerów i monitorujących aplikacji, bez konieczności podawania haseł, co jest dodatkowym atutem.

5. Wnioski

Podsumowując – cel został osiągnięty. Stworzony na potrzeby niniejszej pracy system, już funkcjonuje produkcyjnie i już powstały dla niego pewne specyficzne modyfikacje i rozszerzenia, związane specyfiką ataków jakie miały miejsce. System zaczął żyć własnym życiem i ewoluować. Został uruchomiony na dwóch serwerach VPS, pracujących pod kontrolą Debian Wheezy 8.1 wraz z komercyjnym oprogramowaniem DirectAdmin (tzw. Panel Serwera).

Skrypty do tworzenia kopii bezpieczeństwa owych dwóch serwerów spisują się bardzo dobrze ponieważ na maszynie VPS wyposażonej w 100GB przestrzeń dyskową, znajduje kilkanaście pełnych kopii obu maszyn VPS, z których każda osobno posiada 25GB oraz 10GB danych, do zabezpieczenia. Uzyskano tak dużą gęstość danych dzięki zastosowaniu mechanizmu twardych linków i zapisywania tych samych danych na dysku z wielu różnych kopii tylko raz w jednej i tej samej przestrzeni (bez ich powielania dla każdej kolejnej kopii bezpieczeństwa – jeśli plik linkowany nie uległ modyfikacji).

Skrypty wysyłające wiadomości e-mail z powiadomieniami o zdarzeniach sprawdzają się w praktyce. Wiele razy przyczyniły się do szybkiego rozwiązania awarii związanych z pojawiającym się błędem w konfiguracji usług lub z przeciążeniem serwera.

Zapora sieciowa i skrypt pilnujący limitów CPU skutecznie zabezpieczyły serwery przed wieloma atakami na utrzymywane na nich strony internetowe oraz konta systemowe i skrzynki pocztowe. Skrypt zapory sieciowej z powodzeniem zablokował w czasie trzech miesięcy pracy serwera około 10 tys atakujących adresów IP, bez widocznego wzrostu obciążenia CPU. Jest to spore osiągnięcie, które udało się uzyskać dzięki tzw. tablicom haszującym, opisanym w niniejszej pracy.

W przyszłości system ServerMonitor, zarówno skrypty jak i sama aplikacja będą rozbudowywane o nowe funkcje monitorujące. Na pewno nie jest to wszystko, co można zrobić w temacie monitoringu serwerów, ale raczej niniejsza praca będzie stanowiła podwaliny pod dalszą rozbudowę systemu monitoringu serwerów.

6. Bibliografia

- [1] N. Zitek, „20000 Academy,” 2014. [Online]. Available: <http://www.20000academy.com/blog/2014/06/17/itil-reactive-proactive-problem-management-two-sides-coin/>.
- [2] O. T. J.C. Mackin, Egzamin 70-412 – Konfigurowanie zaawansowanych usług Windows Server 2012 R2, przekład: Krzysztof Kapustka, Warszawa: APN PROMISE, 2014.
- [3] Wikipedia, „Power Usage Effectiveness,” [Online]. Available: http://en.wikipedia.org/wiki/Power_usage_effectiveness.
- [4] C. Hunt, Serwery sieciowe Linuksa, przekład: Jacek Sokulski, Warszawa: MIKOM, 2000.
- [5] C. A. Bruce Eckel, Thinking in C++, Tom 2, Helion, 2004.
- [6] M. G. Sobell, Praktyczny przewodnik LINUX. Programowanie w powłoce. Wydanie III., Gliwice: Helion, 2014.
- [7] M. G. Sobell, Linux. Programowanie w powłoce, Wydanie III, Helion, 2014 r..
- [8] A. Frisch, UNIX Administracja systemu, READ ME, 1997.
- [9] J. Grębosz, Symfonia C++, Tom 1 i 2, Kraków: Edition 2000, 2008.
- [10] „QtCreator,” [Online]. Available: <https://www.qt.io/>.
- [11] „DirectAdmin,” [Online]. Available: <http://www.directadmin.com/>.
- [12] „Draw.IO,” [Online]. Available: <https://www.draw.io/>.
- [13] P. E. Alan Ezust, C++ i QT, Wprowadzenie do wzorców projektowych, Wydanie II, Helion, 2014.
- [14] A.-c. p. G. Code. [Online]. Available: <https://code.google.com/p/apt-cyg/>.
- [15] M. Jakl, „Time Machine for every Unix out there,” [Online]. Available: https://blog.interlinked.org/tutorials/rsync_time_machine.html.
- [16] C. Hunt, TCP/IP Administracja sieci, O'Reilly, READ ME , 1998.
- [17] D. Community, „SNMP,” [Online]. Available: <https://wiki.debian.org/SNMP>.
- [18] D. Community, „Chroot,” [Online]. Available: <https://wiki.debian.org/chroot>.

- [19] N. Wirth, Algorytmy + struktury danych = programy, Warszawa: Wydawnictwa Naukowo-Techniczne, 2004.
- [20] J. W. Craig Hilton, PHP3. Internetowe aplikacje bazodanowe, Helion, 2000.
- [21] G. S. S. Garfinkel, Bezpieczeństwo w UNIXie i Internecie, READ ME, 1997.
- [22] B. Eckel, Thinking in C++ Tom 1, Helion, 2002.
- [23] „Advanced Bash-Scripting Guide,” [Online]. Available:
<http://www.tldp.org/LDP/abs/html/>.

7. Spis rysunków

Rysunek 1.1 – system typu ticket – HelpSpot.	3
Rysunek 1.2 – system wykresów – Observium.	4
Rysunek 1.3 – monitoring serwera www – Apache Server Status	7
Rysunek 1.4 – monitoring obciążenia CPU – Htop.....	8
Rysunek 1.5 – konsolowy monitoring wielu zasobów/serwerów jednocześnie – Tmux.	9
Rysunek 2.1 – panel zarządzania – DirectAdmin.....	17
Rysunek 3.1 – diagram kontekstowy.....	21
Rysunek 3.2 – okno aplikacji "ServerMonitor".....	24
Rysunek 3.3 – wiele instancji "ServerMonitor".	25
Rysunek 4.1 – skrypt onekiller.sh – funkcja checking().....	46
Rysunek 4.2 – docelowy algorytm skryptu onelikker.sh.....	47
Rysunek 4.3 – skrypt onekiller.sh – funkcja killing().....	48
Rysunek 4.4 – mainwindow.h.....	51
Rysunek 4.5 – mainwindow.cpp.....	51
Rysunek 4.6 – konstruktor.....	52
Rysunek 4.7 – implementacja metod.....	53

8. Spis listingów

Listing 1 – generowanie klucza ssh.	27
Listing 2 – initial-backup-server.sh.	28
Listing 3 – hardlink-backup-server.sh.	29
Listing 4 – exclude-list.txt.	30
Listing 5 – firewall – zaporą sieciową ipv4.	31
Listing 6 – skrypty startowe.	35
Listing 7 – crontab.	35
Listing 8 – autofail2ban.sh.	36
Listing 9 – ipset.sh.	40
Listing 10 – sprawdzanie stref DNS.	41
Listing 11 – topdump.sh.	42
Listing 12 – topdump.sh w /etc/crontab.	42
Listing 13 – check-service.sh.	43
Listing 14 – check_hdd_usage.sh.	43
Listing 15 – check_all_snmp.sh.	44
Listing 16 – test.php.	48
Listing 17 – pingtest.sh.	49
Listing 18 – count_pi.sh.	53
Listing 19 – count_fi.sh.	54
Listing 20 – crontab klienta.	55