

Bezpieczeństwo systemu FreeBSD

Przemysław Frasunek

21 maja 2003 roku

1 Wstęp

FreeBSD jako system „z pudełka” jest domyślnie zainstalowany w sposób dość bezpieczny. Nie oznacza to oczywiście, że w takim stanie powinien pracować w roli ważnego serwera. Większość administratorów przed podłączeniem serwera do sieci i założeniem pierwszych kont, czy uruchomieniem pierwszych serwisów wprowadza pewne zmiany w konfiguracji systemu. Właśnie te zmiany, określone inaczej mianem „zabezpieczania systemu” chciałbym opisać w poniższym artykule, który skierowany jest przede wszystkim do osób posiadających podstawową wiedzę o administracji systemami z rodziny BSD.

1.1 Krótka charakterystyka zagrażających bezpieczeństwu błędów w oprogramowaniu

1.1.1 Przepelnienie bufora

Historia błędów tego typu sięga roku 1988, kiedy pojawił się pierwszy internetowy robak, Morris Worm, atakujący na całym świecie serwery Unixowe na platformie VAX i Sun, działające pod kontrolą systemu BSD. Robak wykorzystywał m.in. błąd w usłudze *finger* w celu nadpisania bufora, zawierającego nazwę pliku lokalnego klienta *finger*.

Najczęstszym celem ataku są funkcje nie sprawdzające długości wprowadzanych danych (`strcpy()`, `strcat()`, `sprintf()`, `sscanf()`), a także niewłaściwie zapisane warunki pętli `for` i `while`. Atak polega na dostarczeniu dłuższego łańcucha znaków, niż mieści się w buforze docelowym. W efekcie, nadpisaniu ulegają dane znajdujące się „za” tablicą. Najprostsze do wykorzystania są błędy, gdzie docelowy bufor jest zmienną automatyczną i znajduje się na stosie. Przykładowy kod:

```
void funkcja(char *p) {
    char buf[1024];
    strcpy(buf, p);
    return;
}
```

W powyższym przykładzie, tuż za buforem znajduje się zapisana wartość `%ebp`, a następnie zapisana wartość `%eip`. Wystarczy więc nadpisać `%eip`, tak aby przy powrocie, funkcja „wróciła” do kawałka kodu, który np. uruchomi powłokę. Łańcuch będzie składał się więc z 1028 bajtów „śmieci” oraz 4 bajtów adresu, pod którym umieszczamy kod do wywołania:

```
(gdb) break funkcja
Breakpoint 1 at 0x8048501: file test.c, line 4.
(gdb) set args 'perl -e 'print "A"x1028 . "\xad\xde\xef\xbe"'
(gdb) run
Starting program: /home/venglin/./t 'perl -e 'print "A"x1028
. "\xad\xde\xef\xbe"'
Breakpoint 1, funkcja (p=0xbfbff3ce 'A' <repeats 200 times>...)
at test.c:4
4          strcpy(buf, p);
(gdb) n
```

```

5             return;
(gdb) info frame
Stack level 0, frame at 0xbfbfff1f4:
   eip = 0x8048517 in funkcja (test.c:5); saved eip 0xdeadbeef
   called by frame at 0x41414141

```

Sporo ciekawszą odmianą tego typu błędów są błędy wykorzystujące fakt błędnego obliczenia wielkości bufora, w efekcie czego znak zerowej terminacji trafia o jeden bajt za daleko. Wbrew pozorom, takie błędy są również proste w wykorzystaniu, czego przykładem był daemon *ftpd* na OpenBSD 2.8.

Nadpisanie jednego bajta poza buforem, wartością 0x0 powoduje wyzerowanie najmłodszego bajta zachowanej wartości rejestru `%ebp`. Pozwala to na „oszukanie” systemu, który „myśli”, że stos został przesunięty do niższych adresów pamięci (rejestr `%ebp` jest rejestrem pomocniczym, przechowującym adres początku stosu). Zachowana wartość `%eip` zostaje wczytana spod niewłaściwego adresu, będącego pod kontrolą napastnika. Jak widać, w większości przypadków, umożliwia to bezproblemowe przejście kontroli nad wykonywanym procesem.

Przykład niewłaściwie napisanej funkcji pochodzi prosto z *ftpd* OpenBSD 2.8:

```

void
replydirname(name, message)
const char *name, *message;
{
    char npath[MAXPATHLEN];
    int i;

    for (i = 0; *name != '\0' && i < sizeof(npath) - 1; i++, name++) {
        npath[i] = *name;
        if (*name == '"')
            npath[++i] = '"';
    }
    npath[i] = '\0';
    reply(257, "\"%s\" \"%s\"", npath, message);
}

```

Jak widać na powyższym przykładzie, problematyczne jest rozwijanie znaków `'"` w pętli `for`, a faktyczne przepełnienie bufora następuje w linijce terminującej łańcuch wartością zerową.

Nieco trudniejsze do wykorzystania są ataki polegające na nadpisaniu bufora znajdujące się na sterpie. Przyjrzyjmy się przykładowi:

```

void funkcja(char *p) {
    char *q, *r;
    q = (char *)malloc(1024);
    r = strdup("/usr/bin/finger");
    strcpy(q, p);
    execl(r, r, NULL);
}

```

Tuż za buforem znajduje się łańcuch `„/usr/bin/finger”`. Nadpisując bufor 1024 bajtami „śmieci”, możemy zmienić ścieżkę do programu, który za chwilę zostanie uruchomiony.

Należy podkreślić, że klasyczne ataki wykorzystujące nadpisanie struktur kontrolnych dystrybutora pamięci `malloc()`, które znane są z systemów Linux, Solaris, Windows i Cisco IOS nie mogą zostać wykorzystane przeciw systemom z rodziny BSD, ze względu na oddzielenie struktur kontrolnych `malloc()` od alokowanych bloków pamięci. W przypadku podatnych systemów, alokowane bloki pamięci stanowią dwukierunkową listę, gdzie wskaźniki do poprzedniego i następnego elementu, a także wielkość elementu znajdują się tuż przed/za obszarem dostępnym dla aplikacji.

Istnieje wiele innych technik wykorzystywania błędów nadpisania bufora. Wszystkie jednak opierają się na modyfikacji fragmentu pamięci, do którego normalnie użytkownik nie ma dostępu. Zabezpieczyć się

przed tego typu atakami można wykorzystując bezpieczne funkcje (uwaga na tzw. problem terminującego znaku zerowego!) oraz sprawdzając długość łańcuchów dostarczonych przez użytkownika.

Obecnie klasyczne błędy pozwalające na ataki typu *buffer overflow* popełniane są już coraz rzadziej, głównie dzięki rosnącej świadomości programistów. Dodatkowo, zastosowanie ciekawych mechanizmów ochrony, takich jak ProPolice¹ czy WxorX², używanych obecnie domyślnie w systemie OpenBSD od wersji 3.3 znakomicie utrudnia wykorzystanie tego typu błędów.

1.1.2 Formatting bugs

Historia błędów tego typu sięga roku 2000, kiedy autor artykułu wraz z grupą specjalistów zajmujących się bezpieczeństwem opublikowali pierwsze³ programy wykorzystujące tego typu błąd w aplikacji *wu-ftpd 2.6.0*.

Atak ten polega na wykorzystywaniu funkcji łańcuchowych z rodziny `printf`, bez jawnego użycia łańcucha formatującego tekst. Umożliwia to wykorzystanie funkcjonalności tychże funkcji do nadpisania dowolnego obszaru w pamięci.

Są dwa główne sposoby wykorzystywania takich błędów. Pierwszy z nich jest ściśle powiązany z funkcją `sprintf()` i jej odmianami. W tym wypadku łańcuch formatujący jest wykorzystywany jedynie do wypełnienia bufora, a nadpisanie zapisanych wartości `%ebp` i `%eip` przeprowadzane jest już w normalny sposób. Przykładowa funkcja:

```
void funkcja(char *p)
{
    char buf[100];

    if (strlen(p) < sizeof(buf))
        sprintf(buf, p);
}
```

Przykład nadpisywania adresu powrotu:

```
Breakpoint 1 at 0x8048562: file test.c, line 5.
(gdb) set args "%100dabcdabcd"
(gdb) run
Starting program: /home/venglin/./t "%100dabcdabcd"

Breakpoint 1, funkcja (p=0xbfbff7be "%100dabcdabcd") at test.c:5
5             if (strlen(p) < sizeof(buf))
(gdb) n
7                 sprintf(buf, p);
(gdb) n
8                 puts(buf);
(gdb) info frame
Stack level 0, frame at 0xbfbff5e0:
    eip = 0x804858b in funkcja (test.c:8); saved eip 0x64636261
    called by frame at 0x64636261
```

Taki sposób ma nieco ograniczone zastosowanie, gdyż w większości przypadków błędna funkcja pochodzi z rodziny `snprintf()`, tak więc ma limitowaną długość łańcucha. Wówczas należy skorzystać z formatu `%,%n'`, w połączeniu z innymi formatami. Pozwala to na „zjęcie” ze stosu określonej ilości bajtów oraz nadpisanie dowolnej komórki pamięci wartością, będącą długością rozwiniętego łańcucha formatującego. Ogólny schemat łańcucha formatującego dla tego rodzaju błędu będzie wyglądał w sposób następujący: `|RRRR|P|W|%n`, gdzie `RRRR` jest wskaźnikiem do nadpisywanej wartości, a `P` formatem zdejmującym odpowiednią ilość bajtów ze stosu, aż do osiągnięcia umieszczonego wcześniej wskaźnika. W jest

¹<http://www.trl.ibm.com>

²<http://www.deadly.org/article.php3?sid=20030126143902>

³<http://downloads.securityfocus.com/vulnerabilities/exploits/wuftpd-2.6.0-exp2.c>
<http://downloads.securityfocus.com/vulnerabilities/exploits/wuftpd2600.c>

„wypełniaczem”, pozwalającym na osiągnięcie odpowiedniej długości rozwinięcia łańcucha formatującego, będącego docelową wartością nadpisania, np.: `abcd%.f%.f%.f%.f%.f%.f%123123d%n`

W przypadku tego rodzaju błędów, wymagana jest znajomość dokładnego adresu nadpisywanej wartości, a także jej położenia na stosie. Czyni to więc ten rodzaj błędów nieznacznie trudniejszymi do wykorzystywania.

1.1.3 Błędy związane z obsługą systemu plików

Niegdyś często spotykanym błędem były tzw. sytuacje wyścigu (ang. *race conditions*), niewłaściwe otwieranie plików tymczasowych oraz przewidywalność ich nazw. Ataki te raczej dotyczą aplikacji uruchamianych lokalnie i polegają na wykorzystaniu wykonywanych przez program operacji na plikach. Oto przykładowy kod:

```
void funkcja(char *p) {
    FILE *fp;

    fp = fopen("/tmp/hehe", "w");
    fprintf(fp, "%s", p);
    fclose(fp);
    return;
}
```

W przykładzie tym, program nie sprawdza, czy otwierany plik już istnieje. Napastnik może stworzyć dowiązanie symboliczne do pliku `/etc/master.passwd` i zmienić w ten sposób jego zawartość.

1.2 Historia bezpieczeństwa FreeBSD

System FreeBSD w domyślnej dystrybucji można uznać za dalece bardziej bezpieczny niż wiele innych systemów z rodziny UNIX. Od roku 1996 opublikowano 251⁴ raportów dotyczących odnalezionych błędów, z czego 126 dotyczy samego systemu lub oprogramowania towarzyszącego, natomiast reszta — portów i pakietów. Ze statystyk dotyczących błędów odnalezionych w roku 2002 wynika, że tylko 12 błędów dotyczyło kodu specyficznego dla FreeBSD, natomiast 32 błędy dotyczyły kodu pochodzącego z innych systemów operacyjnych oraz 95 błędów dotyczyło aplikacji dostarczanych w pakietach i portach.

W konkurującym pod względem bezpieczeństwa systemie operacyjnym OpenBSD w krótszym okresie czasu (1997–2003) odnaleziono 200⁵ błędów w samym systemie i aplikacjach towarzyszących, bez uwzględnienia błędów w portach i pakietach.

Oczywiście, porównania ilości raportów o błędach nie należy uważać za miarodajną ocenę obu systemów operacyjnych. Intencją autora było jedynie zaznaczenie, że marketing systemu OpenBSD, jako najbezpieczniejszego na świecie nie znajduje uzasadnienia w danych statystycznych, gdyż zarówno NetBSD, jak i FreeBSD mogą się pochwalić równie dobrą historią bezpieczeństwa, a wymiana kodu między systemami pozwala udoskonalić je o wcześniej nieobecnej funkcjonalności⁶.

Znakomita większość błędów w obu systemach operacyjnych dotyczyła powszechnych aplikacji, będących rozprowadzanych jako część wielu systemów operacyjnych. W szczególności są to: *BIND*, *Sendmail*, *Taylor UUCP*, *lpd*, *Kerberos*, *tcpdump*. Błędy takie — z punktu widzenia programistów systemowych — są najtrudniejsze do wykrywania, bowiem analizowanie cudzego kodu jest trudniejsze niż tworzenie własnego. Wiele z powyższych aplikacji zostało stworzonych wiele lat temu przez programistów nie mających żadnego doświadczenia w bezpiecznym programowaniu.

⁴wszystkie dane pochodzą z analizy archiwum raportów dotyczących bezpieczeństwa, znajdującego się pod adresem <ftp://ftp.freebsd.org/pub/CERT/advisories>

⁵dane pochodzą z analizy archiwum błędów znajdującego się na stronie <http://www.openbsd.org/errata.html>, autor uwzględnił tylko błędy z kategorii SECURITY, pomimo że wiele błędów z kategorii RELIABILITY pozwalało na zdalną lub lokalną destabilizację pracy systemu, a takie błędy według nomenklatury FreeBSD są traktowane na równi z błędami pozwalającymi na nieuprawniony dostęp do systemu

⁶przykładem może być przeniesienie stosu IPsec z OpenBSD na FreeBSD, dzięki czemu ten drugi posiada on dwie niezależne implementacje IPsec

2 Analiza bezpieczeństwa systemu FreeBSD

2.1 Domyślnie uruchomione usługi

Podobnie jak system OpenBSD, również FreeBSD posiada bardzo niewielką ilość działających domyślnie usług. W przypadku zdalnych usług są to:

- *OpenSSH*
- *inetd* (wraz z serwisem *time*)
- *Sendmail*

Profile bezpieczeństwa dostępne jeszcze na poziomie instalacji pozwalają dodatkowo okroić lub rozszerzyć listę dostępnych domyślnie serwisów. Wśród lokalnych usług działających domyślnie należy wymienić:

- *syslogd*
- *cron*
- *usbd*

2.2 System plików

Domyślna instalacja systemu w wersji 4.8-RELEASE zawiera 61 plików z ustawionymi bitami SUID lub SGID, z czego 38 to aplikacje wykonujące się z prawami superużytkownika. W przypadku OpenBSD 3.3, 56 plików posiada bity SUID lub SGID, z czego tylko 28 to aplikacje wykonujące się z prawami superużytkownika. Separacja kodu, który nie musi być wykonywany z prawami superużytkownika od kodu, który tego wymaga wydaje się być trafionym pomysłem, który niestety nie jest jeszcze powszechnie realizowany przez programistów FreeBSD.

W systemie FreeBSD nie istnieją żadne wbudowane mechanizmy, pozwalające na ograniczenie możliwości wykonywania własnych aplikacji przez użytkowników, aczkolwiek ochrona taka może zostać zaimplementowana za pomocą modułów *rexec*⁷ lub *Cerber*.

2.3 Ochrona przed atakami

W domyślnej instalacji FreeBSD brakuje także jakichkolwiek mechanizmów utrudniających wykorzystanie błędów typu *buffer overflow*. Pomimo tego, że ich skuteczność nie jest pełna, znacznie podwyższają one poziom trudności, który musi pokonać potencjalny napastnik. Przykładem mechanizmu, który może być zaimplementowany na systemie FreeBSD jest opisany wcześniej ProPolice; autorzy przetestowali go w środowisku FreeBSD 4.4-RELEASE, jednakże dobrym posunięciem byłoby zintegrowanie go z systemem, tak jak uczynili programiści OpenBSD.

3 Aktywne bezpieczeństwo

3.1 Podstawy

Pierwszym i niezwykle ważnym elementem bezpieczeństwa każdego systemu jest dostęp administratora i użytkowników. Wiąże się to ze stosowaniem bezpiecznego hasła i/lub innych mechanizmów kontroli dostępu. Poniższe wskazówki powinny być wzięte pod uwagę podczas projektowania mechanizmu dostępu do zdalnego systemu:

- hasła użytkowników i administratora nie powinny być łatwe do zgadnięcia, idealne są całkowicie losowe hasła
- w miarę możliwości należy zdeaktywować serwisy pozwalające na autoryzację czystym tekstem — *telnet*, *rlogin*, *rsh*, *ftp*, *pop3* i wiele innych; zamiast nich należy zastosować szyfrowane rozwiązania alternatywne lub skorzystać z tuneli SSL⁸.

⁷<http://cerber.sourceforge.net>

⁸np. stunnel — <http://www.stunnel.org>

- w miarę możliwości powinny być stosowane hasła jednorazowego użytku, najlepiej generowane przez token, ewentualnie przygotowywane przy pomocy mechanizmu S/Key⁹ lub OTP¹⁰.
- znacznie lepiej używać mechnizmu autoryzacji z wykorzystaniem klucza publicznego zabezpieczonego hasłem i całkowicie zablokować możliwość tradycyjnego logowania na konto
- mechanizmy Kerberos i NIS/YP ułatwiają zarządzanie dostępem do sieci składającej się z wielu serwerów, jednakże serwer centralny jest w takim wypadku zawsze słabym punktem i powinien być szczególnie chroniony
- osoba posiadająca fizyczny dostęp do serwera może na wiele sposobów uzyskać nieuprawniony dostęp do danych na nim zgromadzonych

3.2 Zainstalowane usługi i aplikacje

Autor stosuje i poleca powszechną metodologię polegającą na ograniczaniu ilości pracujących usług oraz zainstalowanych aplikacji, w szczególności SUID i SGID. Po uruchomieniu nowego serwera, przed jego podłączeniem do sieci zalecane są następujące kroki:

- deaktywacja nieużywanych usług (np. *sendmail*)
- deaktywacja nieużywanych daemonów
- usunięcie bitów SUID i SGID z aplikacji, które nie będą używane¹¹
- usunięcie nieużywanych części systemu, w szczególności nieużywanych binariów
- odmontowanie nieużywanych systemów plików (np. *procfs*)

Zakres działających usług i dostępnych binariów powinien być każdorazowo określony dla poszczególnych serwerów.

Usługi pracujące na danym serwerze powinny być w miarę możliwości podzielone pod względem funkcjonalnym oraz odseparowane od siebie przy pomocy mechanizmu *jail*¹², przy czym każda z wirtualnych maszyn powinna być zainstalowana według podanych już wytycznych.

3.3 Firewall

FreeBSD posiada wbudowane dwa niezależne mechanizmy filtra pakietów — *ipfw* oraz *ipfilter*; pierwszy z nich jest specyficzny dla systemu FreeBSD, natomiast drugi dostępny na wiele platform i zintegrowany także z innymi systemami. Funkcjonalność obu jest bardzo zbliżona i dokładnie opisana w stosownych manualach¹³. Również niezmiennie dla obu są podstawowe zasady konfiguracji:

- jako politykę stosuj domyślne blokowanie wszystkiego i selektywne zezwalanie
- utrzymuj archiwum logów z odrzuconych pakietów, ewentualnie także z nawiązywanych połączeń TCP
- korzystaj z mechanizmu utrzymywania stanów na firewallu (*state filtering*)
- na interfejsie zewnętrznym odfiltruj podsieci, które nie są używane w internecie (10.0.0.0/8, 127.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16, 224.0.0.0/3, 255.0.0.0/8)
- na interfejsie zewnętrznym odfiltruj przychodzące pakiety, które posiadają adres źródłowy z Twojej sieci — takie pakiety nie mogą się pojawić w naturalnych warunkach i oznaczają próby spoofingu
- na interfejsie zewnętrznym odfiltruj wychodzące pakiety z adresem źródłowym innym niż używane w Twojej sieci

⁹[key\(1\)](#)

¹⁰[otp\(1\)](#)

¹¹na niektórych maszynach wystarczy aby tylko `/usr/bin/passwd` oraz `/usr/bin/su` posiadały bity SUID

¹²[jail\(8\)](#)

¹³[ipfw\(8\)](#), [ipf\(5\)](#)

- odfiltruj pakiety posiadające nietypowe kombinacje flag TCP (np. URG, PSH, FIN)
- w miarę możliwości stosuj limit pasma dla pakietów SYN i ICMP
- stosuj normalizację pakietów wchodzących do sieci, w szczególności dotyczy to składania fragmentów
- regeneruj pole sekwencji TCP dla pakietów wychodzących
- odrzucaj pakiety ICMP redirect
- weź pod uwagę możliwość limitowania filtrowania interfejsu lo0, jest to szczególnie interesujące w przypadku posiadania dużej ilości środowisk *jail*

Firewall powinien być zlokalizowany zarówno w punkcie styku sieci wewnętrznej z internetem, jak i na każdym z serwerów świadczących usługi. Pamiętajmy, że duża część ataków pochodzi z wnętrza sieci lokalnej!

Gdy budujemy rozproszoną sieć korporacyjną, warto zadbać o to, żeby poszczególne jej części były ze sobą połączone przy pomocy tuneli IPsec. Zarówno OpenBSD, jak i FreeBSD posiadają natywne wsparcie dla protokołów ESP, AH i IPComp w trybach *transport* i *tunnel*.

3.4 Ograniczanie uprawnień

Ograniczanie dostępu do poszczególnych elementów systemu jest kluczowe zarówno dla serwerów posiadających lokalnych użytkowników, jak i dla serwerów świadczących jedynie zdalne usługi. Pod hasłem „ograniczania uprawnień” rozumieć należy zarówno proste zmiany o charakterze „bezpieczeństwo przez oczywistość”, jak i skomplikowane mechanizmy, które można nazwać *systemowymi firewallami*.

Do prostych mechanizmów można zaliczyć:

- odpowiednie ustawienie praw dostępu do plików systemowych
- stosowanie mechanizmu *securelevel*¹⁴, wraz ze stosownie ustawionymi flagami¹⁵ na wszystkich binariach i niektórych plikach konfiguracyjnych
- ograniczenie zasobów, które może użyć aplikacja lub użytkownik¹⁶

Bardziej złożone definicje zasad dostępu do systemu przez każdą aplikację możliwe są do wprowadzenia za pomocą modułu *Cerber*, który m.in. pozwala na dokładną specyfikację zakresu wywołań systemowych i ich argumentów, do których dostęp będzie posiadać aplikacja.

Inny moduł jądra — *revec* również może się przydać do ograniczania dostępu do systemu. Oferuje on następującą funkcjonalność:

- uniemożliwienie wybranym użytkownikom uruchamiania ich programów
- usuwanie zmiennych środowiskowych LD_
- limitowanie ilości argumentów przekazywanych do wywoływanej aplikacji SUID/SGID
- limitowanie długości argumentów przekazywanych do wywoływanej aplikacji SUID/SGID
- filtrowanie niedozwolonych znaków z argumentów przekazywanych do wywoływanej aplikacji SUID/SGID
- przekazywanie tylko określonych zmiennych środowiskowych do wywoływanej aplikacji SUID/SGID
- limitowanie długości zmiennych środowiskowych przekazywanych do wywoływanej aplikacji SUID/SGID
- filtrowanie niedozwolonych znaków z argumentów przekazywanych do wywoływanej aplikacji SUID/SGID

¹⁴`init(8)`

¹⁵`chflags(1)`

¹⁶`login.conf(5)`, `limits(1)`

3.5 Wykrywanie ataków

Wykrywanie niektórych zdalnych ataków jest ułatwione, gdy zastosujemy aplikację z rodziny NIDS (*network intrusion detection system*). Przykładem może być tu *Snort*¹⁷, który przy pomocy aktualnej i obszernej bazy sygnatur potrafi wykryć wiele ataków.

Podstawowym problemem występującym przy próbie wykrycia udanego włamania jest fakt, że napastnik prawie zawsze usuwa lub zniekształca pozostawione logi systemowe. Jednym z ciekawych rozwiązań tego problemu jest wysyłanie logów na innego hosta, a jeszcze lepiej — na konsolę szeregową, która posiada możliwość archiwizowania przychodzącej zawartości. Interesującym rozwiązaniem wydaje się także wysyłanie logów do bazy SQL i archiwizowanie ich tam, co w przyszłości może pozwolić na łatwiejsze wyszukiwanie. Autor przygotował prostego *wrappera*¹⁸, który odbiera wiadomości od aplikacji *syslogd* i dodaje je do bazy MySQL.

Opisane powyżej aplikacje *rexec* i *Cerber* pozwalają na logowanie wywołań systemowych `execve()`¹⁹, co pozwala na śledzenie czynności wykonywanych przez lokalnych użytkowników oraz nieświadomych napastników.

Warto także przechowywać w bezpiecznym miejscu sumy kontrolne MD5 wszystkich binariów systemowych, bibliotek i ważniejszych plików konfiguracyjnych. Można to zaimplementować zarówno pisząc prosty skrypt, jak i korzystając z gotowych „kombajnów”, takich jak *Tripwire*²⁰.

4 Na zakończenie

Bezpieczeństwo nie jest produktem, jest procesem. Do zadań administratora powinno należeć stale monitorowanie odpowiednich list dyskusyjnych, na których raportowane są błędy w odnajdywanych aplikacjach. Pamiętajmy również, że stosowanie różnych, niekonwencjonalnych rozwiązań potrafi w wielu wypadkach utrudnić życie niedoświadczonemu napastnikowi.

¹⁷usr/ports/security/snort

¹⁸<http://www.frasunek.com/sources/security/sqlsyslogd/>

¹⁹*Cerber* pozwala na logowanie prawie wszystkich wywołań systemowych

²⁰usr/ports/security/tripwire