KONRAD ZUWAŁA

Rootkity ACPI

Stopień trudności

Rootkity od zawsze były zmorą administratorów sieci. Jednak wraz z rozwojem technologii ukrywania złośliwego kodu w systemie, rozwijały się także narzędzia, które go wykrywały. Przed ich twórcami stanęło teraz nowe zadanie: rootkity, które ukrywają się w BIOSie, znane pod nazwą rootkitów ACPI.

Odkąd powstały komputery, zaczęto sobie zadawać pytanie o ekologię takiego urządzenia. Także koszty jego działania odgrywały dużą rolę. W związku z tym już w 1989 roku firma Intel przedstawiła swoją pierwszą specyfikację standardu zarządzania energią. Jednak prawdziwy przełom nastąpił w 1992 roku, wraz z utworzeniem organizacji Energy Star i idącego z tym standardu APM (ang. *Advanced Power Management* – Zaawansowane Zarządzanie Energią). Umożliwiał on niższy pobór mocy przez urządzenia dzięki częściowemu lub całkowitemu wyłączaniu niektórych z nich wtedy, kiedy nie były w użyciu.

Standard APM był jednak niedoskonały. Z założenia był on wbudowany bezpośrednio w pamięć płyty głównej, więc poprawne zarządzanie energią oraz bezpieczeństwo były niezależne od systemu operacyjnego i opierały się na kodzie, którego działanie było nie do końca sprawdzone – kodzie, który zawierał liczne błędy. System operacyjny miał niewielki (jeśli nie zerowy) wpływ na rozdysponowywanie energii na poszczególne urządzenia.

Odpowiedzią na niedoskonałości APM było opublikowanie specyfikacji ACPI (ang. *Advanced Configuration and Power Interface* – Zaawansowany Interfejs Konfiguracji Energii) w roku 1996. Standard ten poprawił większość niedoskonałości APM. Najważniejszym usprawnieniem było przesunięcie wszystkich funkcji związanych z zarządzaniem energią na poziom systemu operacyjnego, co poprawiało bezpieczeństwo i uniezależniało zachowanie ACPI od konkretnej wersji płyty głównej czy określonego producenta. Dodatkowo wprowadzono język maszynowy i skryptowy (AML i ASL), które umożliwiają definiowanie funkcji przydatnych z punktu widzenia ACPI. Co więcej, skrypty ACPI uruchamiane są z uprawnieniami najwyższego poziomu, mają one więc pełen dostęp do zasobów komputera, mogą modyfikować pamięć i w razie potrzeby być wywołane z poziomu systemu operacyjnego. Jednak ACPI ciągle było krytykowane za zbyt obszerną, liczącą kilkaset stron specyfikację oraz za podejście do kwestii bezpieczeństwa związanych z dostępem ACPI do zasobów komputera.

Specyfikacja ACPI – jak to działa?

Pierwszą specyfikację standardu ACPI opublikowano w grudniu 1996 roku. Do dnia dzisiejszego ulegała ona wielokrotnym zmianom, obecnie obowiązuje już wersja trzecia tegoż standardu. Jednak ogólne zasady nie uległy zmianie i obowiązują ciągle od pierwszych wersji standardu.

Głównym usprawnieniem w stosunku do APM było przekazanie zarządzania poborem energii do systemu operacyjnego. Poprzednio to BIOS zarządzał całym procesem. Kolejną nowinką było wprowadzenie zaczerpniętej z komputerów osobistych technologii, dopuszczającej przeniesienie komputera w stan bardzo niskiego

Z ARTYKUŁU DOWIESZ SIĘ

- czym jest ACPI,
- poznasz zasady działania ACPI,
- poznasz podstawy języka skryptowego ACPI,
- jak działa rootkit ukrywany w BIOSie,
- jak bronić się przed tego typu zagrożeniami.

CO POWINIENEŚ WIEDZIEĆ

- znać podstawy działania komputera,
- czym są przerwania i na jakiej zasadzie one działają,
- czym jest rootkit.

poboru mocy poprzez odłączenie napięcia od podstawowych podzespołów – takich, jak pamięć RAM czy dyski twarde. Umożliwiono również włączanie komputera za pomocą klawiszy na klawiaturze, poprzez specjalny klawisz lub wpisanie odpowiedniej sekwencji klawiszy, ustawianej w BIOSie komputera.

ACPI wprowadził tzw. stany działania komputera. Stany te definiują sposób poboru mocy przez sprzęt. Dzielą się one na podstany, coraz to bardziej szczegółowo określając ilość niezbędnej energii. Podstawowymi stanami są:

- G0 – pracuje – komputer znajduje się pod zasilaniem i jest w trakcie pracy,
- G1 – uśpiony – stan hibernacji lub wstrzymania znany np. z komputerów przenośnych, dzieli się na kolejne podstany określające jak *bardzo* komputer jest uśpiony i jaki jest jego aktualny pobór napięcia,
- G2 – stan podobny do G3, gdzie komputer jest wyłączony – z tą różnicą, że część komponentów może znajdować się pod napięciem w celu umożliwienia np. uruchomienia komputera za pomocą klawisza spacji, w stanie tym zasilanie musi być fizycznie podłączone do komputera,
- G3 – stan wyłączenia, brak zasilania, stan równoważny fizycznemu odłączeniu zasilania od komputera.

Z punktu widzenia rootkita najważniejszą częścią standardu ACPI jest dostępność języków AML i ASL, które to umożliwiają modyfikację pamięci komputera (tablic ACPI), co może prowadzić np. do nadpisania sterownika urządzenia sieciowego – tak, aby to wysyłało pakiety na określony adres MAC/IP (w zależności, czy działamy w sieci lokalnej, czy w sieci Internet).

Istotnym elementem ACPI w kontekście działania rootkita są tak zwane tablice (czy też tabele) ACPI, które zawierają informacje o różnych urządzeniach zainstalowanych w komputerze i sposobie ich obsługi. Możliwość nadpisania tych tablic pozwala na zaalokowanie kodu rootkita do odpowiedniego miejsca w pamięci.

Skoro tablice te są istotnym elementem wymaganym do poprawnego działania rootkita, należałoby poznać ich rodzaje i funkcje. Wyróżniamy ponad 10 typów takich tabel, począwszy od głównej tablicy urządzeń, wskaźników, tablic rozszerzonych, firmware. Więcej na ich temat można znaleźć pod adresami podanymi w ramce *W Sieci*. Dla nas jednak tablice te nie będą istotne, albowiem użyjemy innej metody – bezpośrednio nadpiszemy pamięć – tak, aby uzyskać pełne przywileje dla naszego *backdoora*. Zostanie przedstawiona zarówno technika dla systemu Windows, korzystająca z funkcji dlań specyficznych, jak również sposób na umieszczenie takiego BIOSowego *backdoora* w systemie z rodziny GNU/Linux, dzięki czemu większość stacji serwerowych i desktopów stanie się podatna na nasz atak.

Założenia rootkita ukrytego w BIOSie – wady i zalety

Zanim przystąpimy do tworzenia właściwego rootkita, który następnie zostanie ukryty w BIOSie, musimy zadać sobie podstawowe pytanie – dlaczego wybieramy akurat taką metodę? Przecież można w o wiele łatwiejszy sposób przygotować działającego *backdoora*, dodatkowo ukryć go w istniejącym systemie plików systemu operacyjnego i mieć problem z głowy, nie zawracając jej sobie niepotrzebnymi trudnościami. Co więc takiego daje nam rootkit ukryty w BIOSie?

Cóż, niewątpliwą zaletą takiego rozwiązania jest jego trwałość. Niezależnie od warunków, w jakich sprzęt będzie działał, niezależnie od liczby jego rebootów, awarii zasilania czy wreszcie nawet przeinstalowania systemu operacyjnego, nasz kod ciągle pozostanie w komputerze. Nie musimy zajmować się więc dodaniem procedur uruchomieniowych rootkita do procesów startowych zaatakowanego systemu operacyjnego, nie musimy obawiać się jego usunięcia wraz z pojawieniem się świeżej wersji systemu na maszynie ofiary (oczywiście, jeśli system będzie zgodny z tym zainstalowanym poprzednio – rootkit dla Linuksa nie zadziała na Windows).

Jedną z największych korzyści jest trudność wykrycia takiego oprogramowania. Przecież klasyczne programy antywirusowe czy wykrywacze rootkitów skanują tylko system plików i pamięć RAM, nie sprawdzają jednak BIOSu i jego pamięci flash. Trudno jest więc pozbyć się takiego szkodnika, jeśli uda mu się zainfekować nasz komputer. Jest to już zupełnie inną kwestią, opisywaną zresztą nieco dalej.

Listing 1. *Prototyp funkcji OperationRegion*

```
OperationRegion(Nazwa, Przestrzeń, Offset, Długość);
```

Listing 2. *Prototyp funkcji SeAccessCheck*

```
BOOLEAN SeAccessCheck(
            IN PSECURITY_DESCRIPTOR SecurityDescriptor,
            IN PSECURITY_SUBJECT_CONTEXT SubjectSecurityContext,
            IN BOOLEAN SubjectContextLocked,
            IN ACCESS_MASK DesiredAccess,
            IN ACCESS_MASK PreviouslyGrantedAccess,
            OUT P PRIVILEGE_SET *Privileges OPTIONAL,
            IN PGENERIC_MAPPING GenericMapping,
            IN KPROCESSOR_MODE AccessMode,
            OUT PACCESS_MASK GrantedAccess,
            OUT PNTSTATUS AccessStatus
            );
```

Listing 3. *Nadpisanie funkcji systemu Windows, przedstawione przez Johna Heasmana na konferencji BlackHat*

```
OperationRegion(SEAC, SystemMemory, 0xC04048, 0x1)
Field(SEAC, AnyAcc, NoLock, Preserve)
{
   FLD1, 0x8
}
Store(0x0, FLD1)
```

Nie ma jednak rzeczy idealnych. Mimo niewątpliwych plusów, rootkity ukrywane w BIOSie mają pewne zasadnicze wady. Przede wszystkim nasuwa się pytanie o możliwość umieszczenia takiego oprogramowania w BIOSie komputera. Przecież można zablokować możliwość *flashowania* BIOSu, a więc zapisywania czegokolwiek w jego pamięci flash. Często jest to blokada sprzętowa, polegająca na ustawieniu jakiejś zworki na płycie głównej. Sam BIOS nierzadko pozwala na regulację uprawnień zapisu w pamięci flash. Aby operacja umieszczania rootkita zakończyła się sukcesem, musimy więc mieć pewność, że uda nam się go zapisać, że płyta główna jest odpowiednio skonfigurowana i wreszcie, że sam BIOS na to pozwala. Nadpisywanie BIOSu też nie jest operacją prostą, musimy być bowiem wyposażeni w odpowiednie oprogramowanie i odrobinę wiedzy o konkretnym układzie.

Sam fakt stworzenia takiego oprogramowania nasuwa pewne trudności. Musimy mieć bowiem pojęcie o niskopoziomowych funkcjach ACPI, o działaniu samego BIOSu, o strukturze pamięci atakowanego systemu operacyjnego i mechanizmach zabezpieczających, których on używa. Wreszcie, musimy mieć pomysł na wklejenie naszego kodu do systemu – tak, aby za każdym razem się poprawnie uruchamiał.

Znając już wady i zalety takiego rozwiązania, możemy z czystym sumieniem przystąpić do tworzenia rootkita – o ile oczywiście uznamy, że nakład pracy, który należy włożyć w jego opracowanie będzie odpowiedni do uzyskanego wyniku.

AML – klucz do bram systemu

AML, czyli język maszynowy ACPI, jest językiem programowania w pełnym tego słowa znaczeniu. Posiada on konstrukcje logiczne, możliwość kontroli przepływu danych, implementuje operacje bitowe, możliwość wykonywania działań arytmetycznych czy synchronizacji wykonywanych operacji. Dostępne są także różne typy danych – zarówno do odczytu, jak i zapisu. Daje on więc całkiem spore możliwości potencjalnemu intruzowi, który chciałby wykorzystać go w celach innych niż pierwotne przeznaczenie.

Użyjemy w zasadzie tylko jednej funkcji, którą doskonale opisuje John Heasman w swojej prezentacji z konferencji BlackHat. Prototyp funkcji przedstawiono na Listingu 1.

Funkcja ta została zaprojektowania w celu udostępnienia prostej metody do tworzenia interfejsów dla urządzeń. Prościej mówiąc, jest ona niezwykle przydatna, gdy chcemy w prosty sposób obsłużyć zachowanie określonego urządzenia, związane najczęściej z pobieraniem przez nie mocy. Funkcja ta przyjmuje cztery parametry:

- Nazwa – ogólna nazwa obszaru w pamięci, do którego się odwołujemy; określamy, jakiego urządzenia jest to pamięć,
- Przestrzeń – przestrzeń w pamięci Nazwa, której dotyczy nasze odwołanie. Poprawnymi polami przestrzeni pamięci operacyjnej (czyli tej, do której będziemy się odwoływać) są SystemIO, SystemMemory, CMOS, PCI_Config, SMBus,
- Offset – jak sama nazwa wskazuje, jest to przesunięcie względem początku pamięci,
- Długość – ilość zapisywanych/czytanych bajtów.

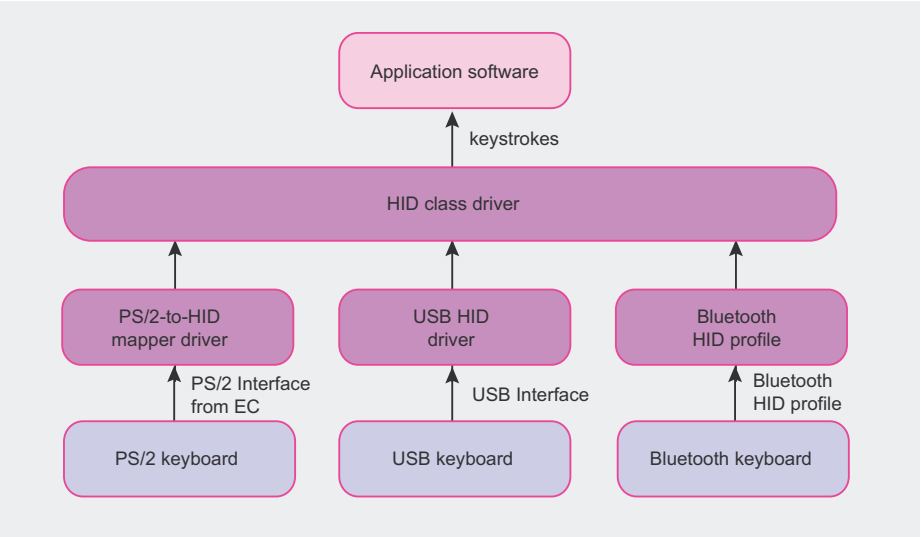
Warto wspomnieć, że funkcja ta może służyć zarówno do zapisu informacji, jak i do ich odczytu. Dodatkowo, nazwy i przestrzenie, zwane zgodnie z nazwą funkcji regionami, można dzielić na mniejsze jednostki zwane polami (ang. *Field*).

Co w praktyce daje nam ta funkcja? Cóż, dosłownie nieograniczone możliwości modyfikowania pamięci. Dzieje się tak dlatego, że sterownik ACPI działa z nieograniczonymi przywilejami, może więc modyfikować dowolne miejsce w pamięci. Oczywiście nie pozostanie to niezauważone, jednak o tym wspomnimy w części artykułu poświęconej wykrywaniu tego typu ataków.

Nasz rootkit wykorzysta jedną z koncepcji Johna Heasmana z konferencji BlackHat, gdzie pokazał on, w jaki sposób możemy utworzyć takiego właśnie *backdoora*. Oczywiście jest to tylko jedna z wielu dostępnych metod, jednak opisywanie każdej z nich nie miałoby większego sensu ze względu na dużą rozległość tematu.

W Sieci

- *http://www.acpi.info* – strona domowa ACPI,
- *http://www.blackhat.com/presentations/bh-federal-06/BH-Fed-06-Heasman.pdf* – prezentacja Johna Heasmana na konferencji BlackHat,
- *http://www.osronline.com/DDKx/kmarch/k110_42wi.htm* – opis funkcji SeAccessCheck,
- *http://pl.wikipedia.org/wiki/ACPI* – strona Wikipedii dotycząca ACPI. Godna lektury.



Rysunek 1. *Struktura ACPI na komputerze*

Wiemy już, jak stworzymy rootkita, pozostaje jeszcze dobór odpowiednich narzędzi. Niezbędny będzie kompilator ASL/AML, który można bezpłatnie pobrać ze stron firmy Microsoft, jak również z oficjalnej strony opisującej standard ACPI. Linki te podane są w ramce *W Sieci*. Niezbędny będzie również dobry asembler i dezasembler. Dlaczego? Otóż, aby umieścić coś w pamięci flash BIOSu, musimy go w jakiś sposób nadpisać. Oznacza to, że należy dołączyć nasz kod do kodu BIOSu producenta płyty głównej, czyli mówiąc wprost – niezbędna jest jego dezasemblacja i ponowna kompilacja wraz z dołączonym kodem. Warto dodać, iż sam BIOS zawiera interpreter niezbędny do wykonania naszego kodu ASL/AML. Doskonałym dezasemblerem jest IDA Pro Free, którą można pobrać ze strony producenta, również wyszczególnionej w ramce *W Sieci*. Zaopatrzeni w odpowiednie narzędzia, możemy przystąpić do demonstracji przykładowego kodu *backdoora*, wraz z opisem jego działania. Będziemy się ciągle opierać na badaniach Johna Heasmana, zaprezentowanych podczas konferencji BlackHat.

Budujemy rootkita

Pierwszą i zasadniczą kwestią, która decyduje o kształcie rootkita, jest system operacyjny, który ma paść jego ofiarą. Różne będą bowiem metody obejścia zabezpieczeń Windows, różne – Linuksa. Chodzi generalnie o włączenie naszego rootkita do pamięci – tak, aby kod w nim zapisany zaczął się wykonywać.

Zacznijmy od systemu Windows – jako, że jest on chyba nieco bardziej popularny od darmowego Linuksa. Zgodnie z tym, co przedstawił Heasman, w systemach z rodziny NT mamy do czynienia z funkcją `SeAccessCheck`, której zadaniem jest sprawdzenie, czy aktualnie działający proces ma pełne, czy ograniczone prawa dostępu do zasobów systemowych. Prototyp tejże funkcji przedstawiony jest na Listingu 2.

Jak widać na Listingu 2., funkcja przyjmuje dużą ilość parametrów. Większości z nich nie trzeba opisywać, albowiem ich nazwy są na tyle intuicyjne, że nawet mało wprawny użytkownik będzie wiedział, jakie jest ich zastosowanie i znaczenie.

Dla nas interesujący jest parametr `AccessMode`, który odpowiada za ustalenie, czy wywołanie jest dokonywane z przestrzeni jądra systemu, otrzymując tym samym pełne uprawnienia, czy też z poziomu przestrzeni użytkownika, czyli ze sporymi ograniczeniami w swobodzie wykonywania kodu. Jasnym jest więc, że musimy w jakiś sposób ustawić dla naszego rootkita odpowiednią wartość tego parametru. Użyjemy do tego celu skryptowego języka ACPI, tak, jak zaprezentował Heasman na konferencji BlackHat. Odpowiednie wywołanie funkcji zaprezentowane jest na Listingu 3.

Jak widać na listingu, funkcja nadpisuje adres w pamięci, w którym rezyduje

Listing 4. *Nadpisanie funkcji obsługi nieużywanego przerwania – sys_ni_syscall()*

```
OperationRegion(NISC, SystemMemory, 0x12BAE0, 0x40)
{
   NICD, 0x40
}
Store(Buffer () {0xFF, 0xD3, 0xC3, 0x90, 0x90, 0x90, 0x90, 0x90}, NICD)
```

Listing 5. *Uruchomienie backdoora*

```
#define UNUSED_INT 0x11 // nieużywane przerwanie
#include <syscall.h>
int backdoor() {
   // tutaj jest kod naszego backdoora
   return -ENOSYS; // kod błędu związany z obsługą nieistniejącego przerwania
}
int main() {
   return (syscall(UNUSED_INT, &backdoor));
}
```

Windowsowe sprawdzenie przestrzeni wykonywania rozkazu. Nadpisując konkretne pole funkcji odpowiednią wartością, wymusimy traktowanie kodu naszego rootkita jako uprzywilejowanego, wywoływanego przez kernel. Stąd uzyskujemy pełne prawa pracy na systemie Windows, tak, jakbyśmy dysponowali rootkitem przestrzeni kernela. Jednak zaletą naszego rozwiązania jest fakt, że rootkit będzie bardzo trudny do wykrycia i usunięcia z poziomu systemu Windows.

Widać jednak, że skonstruowanie takiego rootkita wiąże się z dużym wysiłkiem i nakładem pracy. Dodatkowo, gdyby nie prezentacja wielokrotnie cytowanego i wspominanego przeze mnie Johna Heasmana, trudnym byłoby dojście do tego wszystkiego i samodzielne odkrycie tajników rootkita ACPI.

W systemie Linux Heasman zaproponował inny rodzaj exploitacji mającej na celu przejęcie uprawnień jądra systemu. Metoda opiera się na wykorzystaniu specjalnej funkcji systemu Linux, której zadaniem jest obsługa nieużywanych przerwań. Musimy więc odwołać się do nieużywanego przerwania systemowego, aby wywołać tę właśnie funkcję. Oczywiście, zostanie ona nadpisana, prowadząc do uruchomienia naszego rootkita zamiast funkcji obsługi. Kod przedstawiony jest na Listingach 4. i 5.

Jak widać na Listingu 4. nadpisujemy adres funkcji tak, by zamiast niej wywołana została funkcja, której adres rezyduje w rejestrze EBX – dane zapisane do bufora to nic innego, jak asemblerowe opcody instrukcji `call ebx; retn; nop; nop; nop; nop; nop`. Ciało funkcji zostało więc nadpisane – tak, by wywołała ona zawartość rejestru EBX – to właśnie on jest używany przy przekazywaniu argumentów do funkcji przerwań przez funkcję `syscall()`. Odpowiednie jej wywołanie zaprezentowane jest na Listingu 5.

Kod zawarty na Listingu 5 łączy to, co zrobiliśmy za pomocą ASL, z systemem operacyjnym – obsługa nieużywanego przerwania jest przekazywana do naszej funkcji, która jest tak naprawdę tylną furtką – możemy tam wstawić kod otwierający np. powłokę na wskazanym porcie, bądź jakąś procedurę logowania zdarzeń wykonywanych na komputerze użytkownika. Jedynym ograniczeniem, jakie mamy, jest nasza wyobraźnia.

Ta sekcja artykułu zaprezentowała sposób, w jaki można przygotować przykładowego rootkita. Techniki te i wiele innych pokazuje John Heasman w swoim wystąpieniu z konferencji BlackHat, do którego odnośniki znajdują się w ramce *W Sieci*. Ostatnia już część niniejszego artykułu będzie poświęcona kwestii, która może być bardziej istotna w codziennej pracy – jak wykryć takiego rootkita i jak się go pozbyć. Nikt przecież nie chciałby być ofiarą podobnego ataku, w dodatku ofiarą niezorientowaną, co można zrobić w takiej nieciekawej sytuacji.

Prewencja i wykrywanie – zabezpieczenie komputera przed rootkitem

Gdy wiemy już, jak działa rootkit, zostaje nam do omówienie kwestia jego wykrycia i, co ważniejsze, zabezpieczenia się przed nim. Zaczniemy więc od metod zabezpieczenia się przed rootkitem w BIOSie.

Podstawową metodą uniemożliwiającą instalacje takiego rootkita jest zabezpieczenie BIOSu przed *flashowaniem*. Nie mogąc nadpisać BIOSu, nie mamy jak umieścić rootkita w jego pamięci, czyli *de facto* jesteśmy bezpieczni. Jest to najprostsza metoda, wiąże się ona jednak z pewnymi niedogodnościami, albowiem przy każdej próbie aktualizacji BIOSu musimy przestawić odpowiednią zworkę na płycie głównej, bądź – jeśli BIOS na to pozwala – zmienić ustawienia w jego programie konfiguracyjnym.

Drugim, choć bardziej radykalnym i niekomfortowym w życiu codziennym, rozwiązaniem jest po prostu wyłączenie obsługi ACPI. Polepszenie bezpieczeństwa jest oczywiste, albowiem wszystkie funkcje odpowiadające za ACPI zostają wyłączone, programy napisane w ASL/AML nie działają – jesteśmy stuprocentowo bezpieczni. Jednak jest to rozwiązanie o tyle niekomfortowe, że tracimy niemałą funkcjonalność – czyż nie każdy z nas przyzwyczaił się do tego, że po naciśnięciu *Wyłącz system* komputer sam się wyłącza? A bez ACPI musielibyśmy w tym celu ręcznie wciskać przycisk Power, oczywiście po uprzednim ukazaniu się komunikatu, znanego np. z Windows 98 - *Możesz teraz bezpiecznie wyłączyć komputer.*

Heasman przedstawił jeszcze jeden sposób na zabezpieczenie. Są to BIOSy podpisywane cyfrowo, które umożliwiają kontrolę nad tym, czy coś niechcianego nie zostało zapisane do pamięci flash. Dzięki temu unikniemy sytuacji, w której niepożądany kod zagości na naszym komputerze.

Pozostaje jeszcze kwestia wykrywania takich niechcianych rzeczy na naszej maszynie. Jak już wspominałem, działania rootkita nie pozostaną do końca niezauważone. W systemie Windows otrzymamy komunikat o błędzie, mówiący o tym, że ACPI próbuje odwołać się do obszaru pamięci zarezerwowanego dla systemu Windows. Będzie to jednak tylko powiadomienie, albowiem Windows zezwoli na taką operację.

Na Linuksie możemy posłużyć się poleceniem dmesg. Za jego pomocą wypiszemy zawartość buforów ACPI oraz odczytamy ewentualne kody błędów – podobnie, jak w Windows, dowiemy się, że *coś* próbuje odwołać się do adresu w pamięci zarezerwowanego dla systemu operacyjnego, bądź – że bufory ACPI *dziwnie* wyglądają.

Detekcja rootkita ACPI nie jest więc zadaniem pracochłonnym dla doświadczonego administratora. Jednak mniej wprawna osoba może nie uznać tych błędów za coś groźnego, tłumacząc sobie to jakimś wyjątkiem lub zwyczajnie – błędem BIOSu.

Podsumowanie

Rootkity ACPI wydają się być ciekawym zjawiskiem we współczesnej informatyce. Jednak ich skuteczność oraz łatwość ukrycia przeciwstawione są ogromnym trudnościom w ich implementacji i doskonaleniu, do ich poprawnego działania niezbędna jest bowiem duża wiedza twórcy. Miejmy nadzieję, że Czytelnik nie padnie ofiarą tak spreparowanego oprogramowania i nie będzie musiał martwić się o bezpieczeństwo własnego BIOSu, nie tracąc przy tym jego funkcjonalności.

Konrad Zuwała
Autor zajmuje się bezpieczeństwem aplikacji internetowych oraz szeroko rozumianą ochroną systemów komputerowych. W wolnych chwilach programuje (głównie C/C++, PHP) oraz zarządza portalem internetowym.
Kontakt z autorem: *kzuwala@poczta.onet.pl*