BIULETYN INSTYTUTU AUTOMATYKI I ROBOTYKI
NR 28, 2010

Programowe zabezpieczenie plików przechowywanych na dyskach zewnętrznych

Jan CHUDZIKIEWICZ

Zakład Systemów Komputerowych, Instytut Teleinformatyki i Automatyki WAT,
ul. Gen. S. Kaliskiego 2, 00-908 Warszawa
jchudzikiewicz@wat.edu.pl

STRESZCZENIE: W artykule przedstawiono programowe rozwiązanie pozwalające na zabezpieczenie zawartości plików przechowywanych na dyskach zewnętrznych podłączanych do komputera poprzez interfejs USB. Przedstawione rozwiązanie wykorzystuje sterownik typu filter-driver dla systemu Windows® oraz algorytm szyfrowania blokowego. Istotą zaprezentowanego rozwiązania było zapewnienie zgodności z systemem plików FAT32 (możliwość zastosowania narzędzi do defragmentacji dysku) oraz spełnienie wymagań czasowych tak, aby przy odczycie danych nie występowały zauważalne opóźnienia czasowe (odczyt plików audio oraz video).

SŁOWA KLUCZOWE: sterowniki urządzeń, algorytmy szyfrowania danych, system plików

1. Wprowadzenie

Ostatnie lata rozwoju teleinformatyki pozwalają wysnuć tezę, iż producenci sprzętu oraz oprogramowania wzmogli wysiłki mające na celu zwiększenie zaufania do oferowanych produktów, głównie poprzez poprawianie stabilności rozwiązań sprzętowo-programowych jak i podnoszenie gwarantowanego przez nie poziomu bezpieczeństwa.

Nawet w najprostszych systemach i aplikacjach istnieje z reguły wiele miejsc, w których potencjalnie możliwy jest atak na zabezpieczenia, a ich liczba ograniczona jest jedynie inwencją napastnika. Uproszczając, wymienić można trzy podstawowe obszary, w których informacje narażone są na przechwycenie:

- ✓ podczas wprowadzania (np. z klawiatury),
- ✓ podczas przesyłania (np. poprzez sieć lokalną czy sieć Internet),
- ✓ podczas składowania (np. na dysku twardym lub taśmie magnetycznej).

W zakresie zainteresowania niniejszego artykułu leży trzeci

z przedstawionych obszarów, obejmujący zabezpieczenie danych składowanych na stałych oraz wymiennych nośnikach informacji. Obecnie szyfrowanie danych na nośnikach informacji stało się powszechne głównie dzięki dużej dostępności rozwiązań programowych, zarówno komercyjnych jak i opartych na otwartym kodzie (ang. *Open Source*), pozwalających zainteresowanym na zabezpieczenie istotnych dla nich informacji przed ich zinterpretowaniem przez osoby nieupoważnione. W efekcie istnieje możliwość realizacji procesu szyfrowania danych w oparciu o jedno z rozwiązań:

- ✓ wykorzystanie mechanizmów wbudowanych w system operacyjny [1],
- ✓ wykorzystanie oprogramowania firm zewnętrznych.

W przypadku pierwszego rozwiązania i najnowszych systemów operacyjnych, mechanizm szyfrowania jest integralnym elementem zabezpieczeń w systemie operacyjnym. Dostarcza silnej ochrony dla systemu i danych przechowywanych na dysku systemowym, gwarantując jednocześnie, że dane przechowywane w komputerze pozostają zakodowane nawet, gdy ktoś próbuje manipulować komputerem przy wyłączonym systemie operacyjnym. Pomaga to bronić się przed „atakami offline", atakami polegającymi na zablokowaniu lub ominięciu zainstalowanego systemu operacyjnego, lub fizycznym usunięciu dysku twardego, aby oddzielnie zaatakować dane. Aby wykorzystać wszystkie możliwości rozwiązań systemowych należy instalować system na komputerach wyposażonych w odpowiednie układy (np. mikrochip TPM - Trusted Platform Module, od wersji 1.2), jak również odpowiednie wersje BIOSU (BIOS spełniający standardy organizacji - Trusted Computing Group).

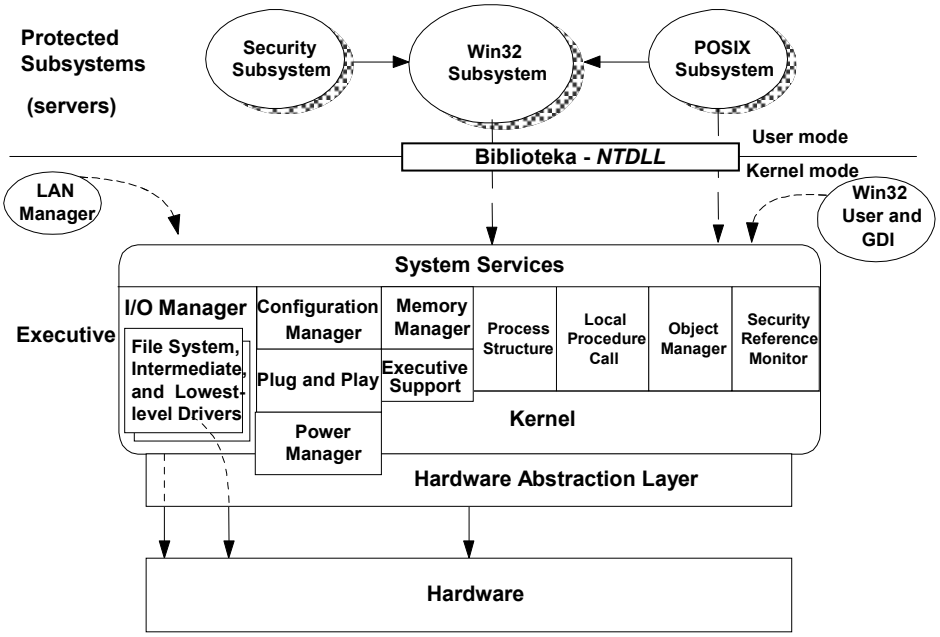
Rozwiązania firm zewnętrznych, zależnie od wersji, charakteryzują się różną funkcjonalnością. W większości przypadków opierają się na najnowszych rozwiązaniach algorytmów szyfrowania, jak [10][11][12]: AES, GOST, Blowfish, 3DES, CAST-128, SAFER, IDEA, RC5. Oprogramowanie tego typu najczęściej wykorzystuje dodatkowe zabezpieczenia w postaci: haseł użytkownika, tokenów kompatybilnych z PKCS#11 (kluczy elektronicznych) oraz plików wykorzystanych jako klucze szyfrujące.

Obydwa rozwiązania charakteryzują się zaletami, ale mają również wady, do których można zaliczyć: w przypadku mechanizmów wewnętrznych systemu najczęściej zabezpieczanie wydzielonego dysku (partycji) bez możliwości wyłączenia działania takiego mechanizmu, natomiast w przypadku narzędzi firm trzecich wiarygodność kodu. Biorąc powyższe pod uwagę wydaje się, że najlepszym rozwiązaniem będzie rozszerzenie funkcjonalności mechanizmów wewnętrznych systemu o dodatkowe mechanizmy zabezpieczeń takie, jak np. mechanizm podpisu elektronicznego. Można to uzyskać poprzez opracowanie odpowiednich sterowników, realizujących proces uwierzytelnienia użytkownika z jednoczesnym zapewnieniem przeźroczystości procesu szyfrowania dla

aplikacji. W rozdziale przedstawiono przykładowe rozwiązanie polegające na opracowaniu sterownika typu mini-filter driver [2][3][4][5][7][8], który będzie realizował szyfrowanie (deszyfrowanie) danych zapisywanych (odczytywanych) na (z) urządzeniu zewnętrznym podłączonym poprzez interfejs USB.

2. Mechanizmy systemu Windows

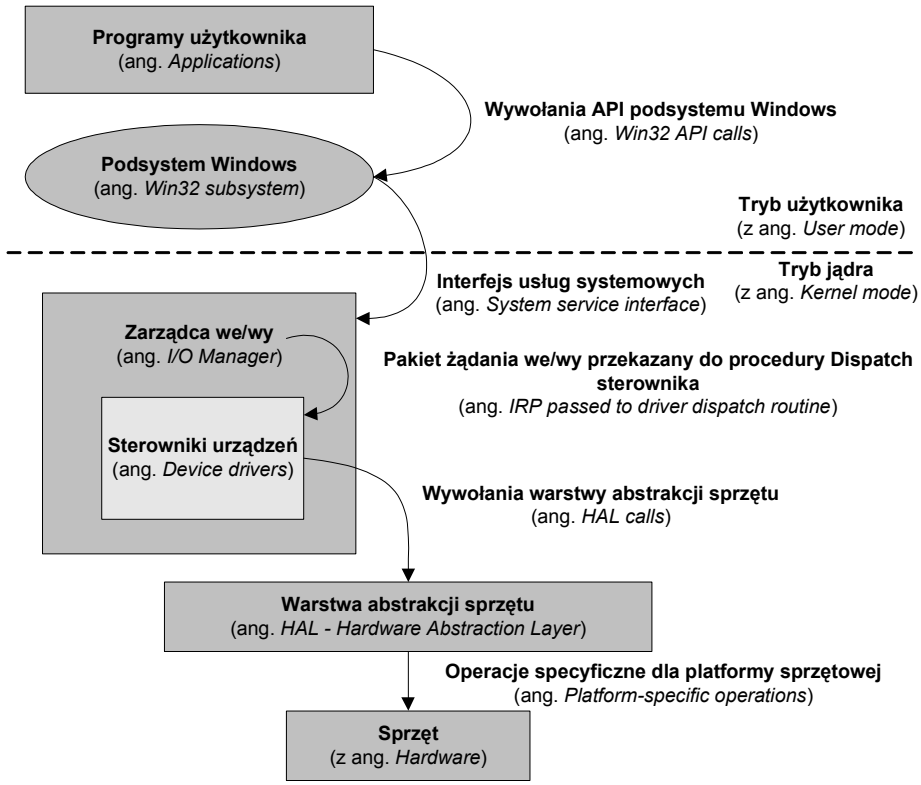
Budowa systemu operacyjnego Microsoftu rodziny Windows (Windows XP i kolejne), zakłada wykorzystanie dwóch poziomów uprzywilejowania, jak przedstawiono to na rysunku 1: trybu użytkownika oraz trybu jądra [7]. Poprzez modułową budowę (widoczna jest bardzo wyraźnie na przykładzie komponentów pracujących w trybie jądra) zrealizowano rozszerzalność systemu, natomiast zastosowanie warstwy abstrakcji sprzętu HAL (ang. *Hardware Abstraction Layer*) pozwoliło na dodanie możliwości przenoszalności pomiędzy różnymi architekturami sprzętowymi.



Rys. 1. Architektura systemów Windows NT [7]

W części obrazującej tryb użytkownika z perspektywy sterowników urządzeń, istotne są jedynie: aplikacje użytkownika, podsystemy środowiskowe (ang. *Subsystem*) (POSIX oraz Win32 - wykorzystywany w prezentowanym

rozwiązaniu) oraz biblioteka systemowa NTDLL - pośrednicząca w komunikacji z elementami trybu jądra. Podsystemy środowiskowe odpowiadają za emulację środowiska pracy dla uruchamianych w nich aplikacji. Tłumaczą ich odwołania do systemu oraz jego zasobów na pierwotne funkcje systemu Windows. Proces komunikacji pomiędzy aplikacją użytkownika a sterownikiem bazuje na przesyłaniu pakietów IRP (ang. *I/O Request Packet*). Pakiety IRP są tworzone przez zarządcę we/wy (ang. *I/O Manager*) na podstawie typu wygenerowanego żądania, co przedstawiono na rysunku 2.

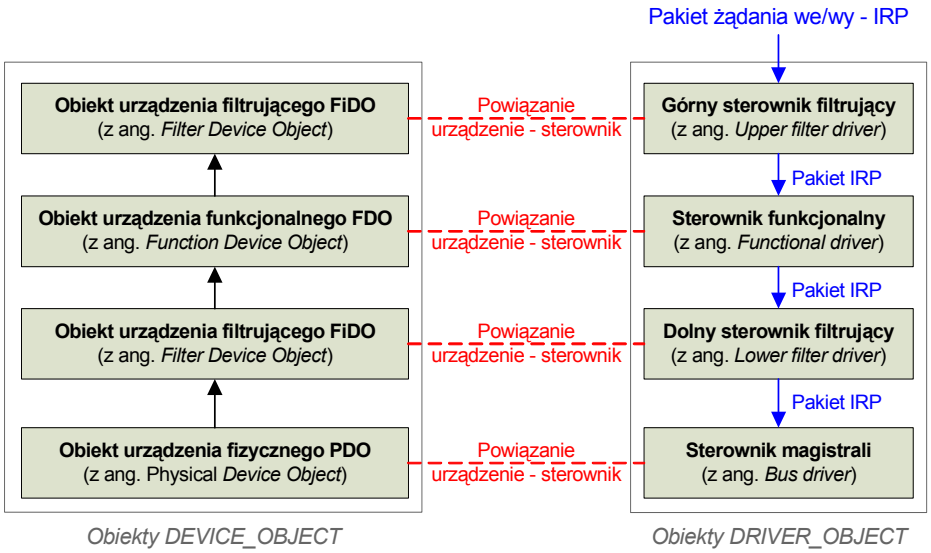


Rys. 2. Schematyczne przedstawienie procesu komunikacji pomiędzy aplikacją użytkownika a sterownikiem [7]

Aplikacje użytkownika komunikują się z komponentami trybu jądra wykorzystując interfejs wywołań usług systemowych, zdefiniowany w bibliotekach podsystemu środowiskowego, w którym pracują. Następnie podsystem środowiskowy, wykorzystując bibliotekę pośredniczącą NTDLL, wywołuje odpowiednie funkcje warstwy wykonawczej trybu jądra systemu (ang. *Executive*). W tym celu każdy komponent udostępnia formalny interfejs, umożliwiający dostęp oraz modyfikację struktur danych pozostających pod jego

kontrolą za pomocą przekazanych parametrów [9]. Udostępnienie przez komponenty jednolitego interfejsu – w większości przypadków udokumentowanego – pozwala na hermetyzację kodu odpowiedzialnego za ich funkcjonalność oraz, co najważniejsze, umożliwia ich wymianę. Dzięki temu pozostałe komponenty pracujące w trybie jądra, w tym sterowniki urządzeń, nie są zależne od implementacji funkcjonalności oferowanej przez poszczególne elementy systemu.

Przyjęty w systemie Windows model obsługi urządzeń zakłada, że urządzenia kontrolowane są poprzez stos współpracujących ze sobą sterowników, z których każdy odpowiedzialny jest za wybrany zestaw funkcji realizowanych przez dane urządzenie. Koncepcja stosu sterowników przedstawiona została na rysunku 3. Według tej koncepcji, sterownik magistrali znajdujący się na dole stosu współpracuje ze sterownikiem funkcjonalnym, zapewniając właściwą użyteczność obsługiwanego urządzenia. Dopuszczono też możliwość stosowania dodatkowych sterowników filtrujących, które, umieszczone na stosie, pozwalają na monitorowanie oraz, w miarę potrzeby, na modyfikowanie pakietów żądań we/wy kierowanych do urządzania leżącego u jego podstaw.



Rys. 3. Koncepcja stosu sterowników w systemie Windows [7]

DRIVER_*OBJECT* są to obiekty reprezentujące sterowniki i tworzone przez zarządcę we/wy w momencie ładowania sterownika. Sterowniki urządzeń odpowiedzialne są za powołanie do życia obiektów typu *DEVICE_OBJECT*,

reprezentujących w systemie urządzenia. Utworzenie obiektu tego typu następuje w chwili wywołania przez zarządcę we/wy procedury *AddDevice* sterownika. W zależności od roli pełnionej przez sterownik, powołany obiekt może reprezentować:

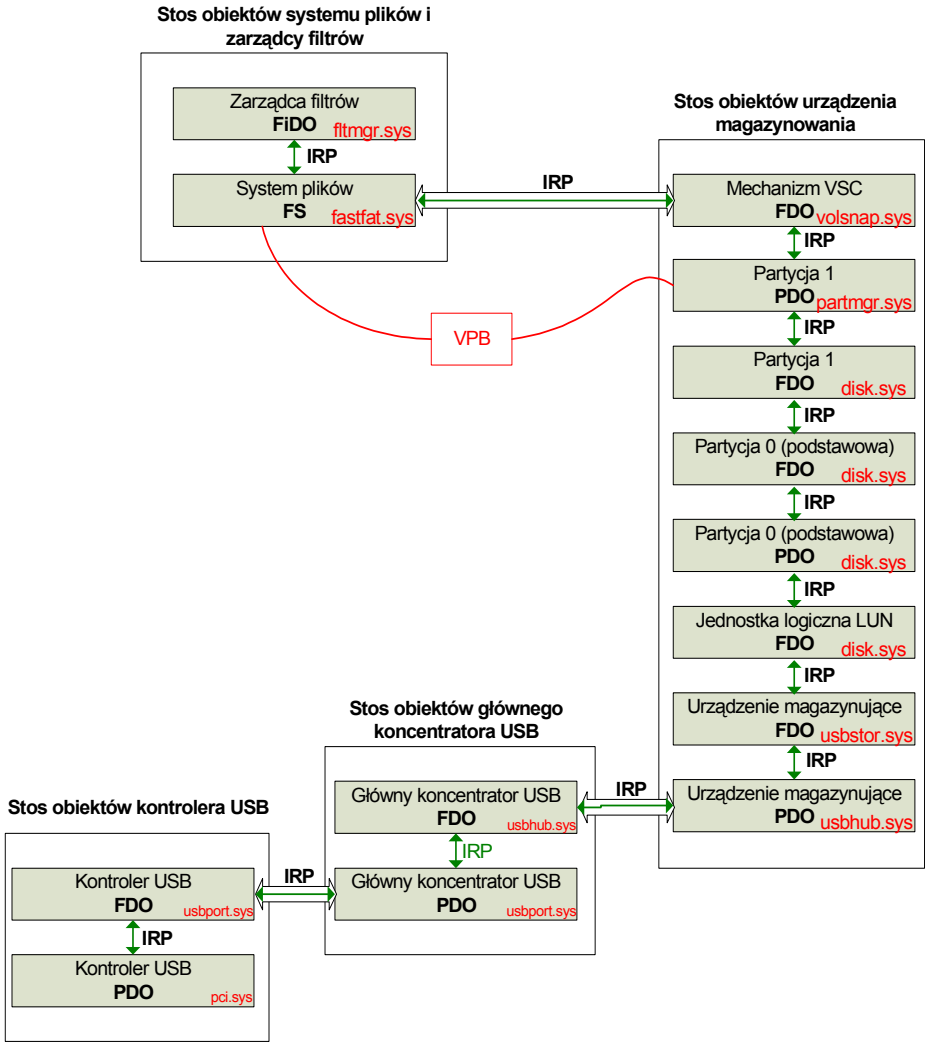
- ✓ obiekt typu PDO (ang. *Physical Device Object*) – reprezentujący połączenie pomiędzy urządzeniem a magistralą,
- ✓ obiekt typu FDO (ang. *Functional Device Object*) – wykorzystywany przez sterownik funkcjonalny do zapewnienia użyteczność urządzenia,
- ✓ obiekt typu FiDO (ang. *Filter Device Object*) – wykorzystywany przez sterownik filtrujący do przechowywania informacji związanych z filtrowaniem pakietów żądań we/wy.

Z chwilą, gdy wszystkie sterowniki ze stosu urządzenia powołają do życia obiekty typu *DEVICE_OBJECT* (rozpoczynając od sterownika znajdującego się najniżej w stosie), zarządca we/wy może rozpocząć przesyłanie pakietów żądań we/wy do urządzenia obsługiwanego przez dany zestaw sterowników.

Na rysunku 4 przedstawiono stos obiektów i skojarzonych z nimi sterowników dla urządzenia magazynowania podłączonego do systemu poprzez magistralę USB. Z przedstawionego schematu wynika, iż najwyżej w stosie sterowników znajduje się sterownik zarządcy filtrów, który zostanie omówiony w dalszej części artykułu. Tuż pod nim znajduje się sterownik systemu plików FAT32, oznaczony na schemacie jako obiekt typu FS (ang. *File system*). Należy w tym miejscu wspomnieć, że sterowniki systemu plików nie są umieszczone w stosie sterowników w sposób jawny. Skojarzenie pomiędzy sterownikiem systemu plików a obiektem reprezentującym partycję w urządzeniu magazynowania zrealizowane jest poprzez strukturę danych VPB (ang. *Volume Parameters Block*), co na rysunku 4 zostało pokazane jako połączenie pomiędzy modułami: PDO Partycja 1 a FS System plików.

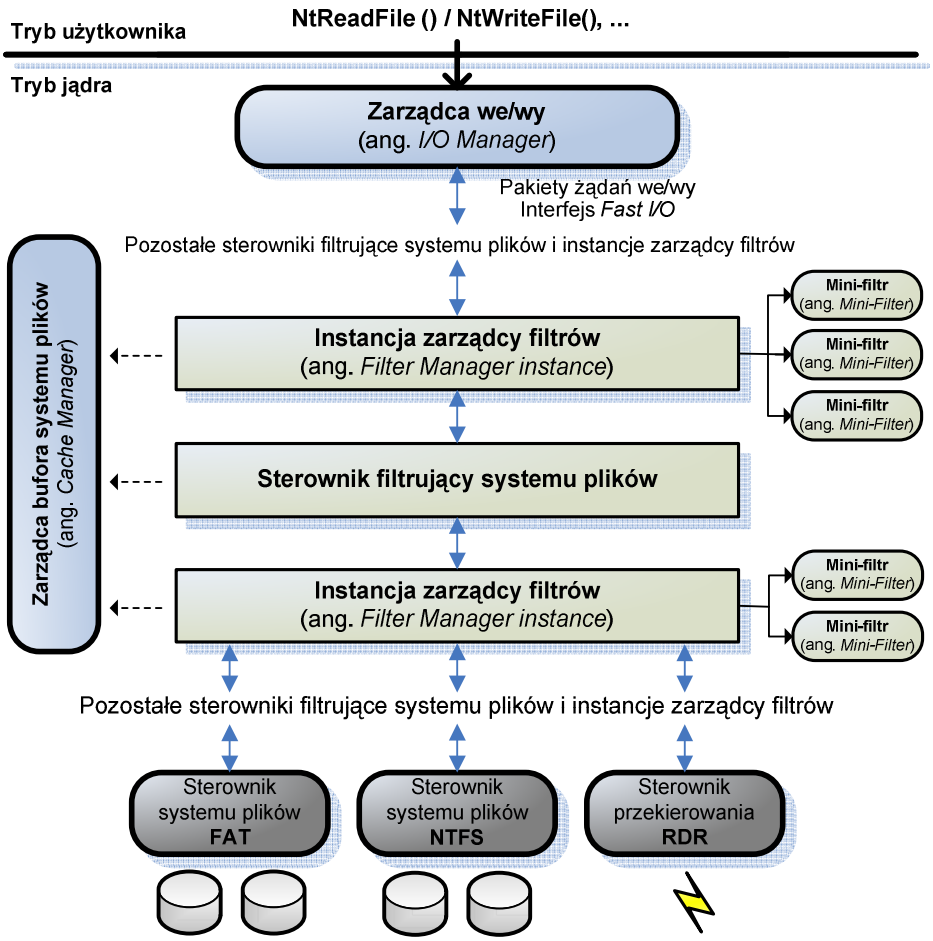
W przedstawionym rozwiązaniu wykorzystano komponent jądra zwany zarządcą filtrów (ang. *Filter Manager*), ułatwiający tworzenie sterowników filtrujących systemu plików. Komponent ten dostępny jest dla wszystkich systemów począwszy od Windows Server 2003, Windows XP z Service Pack 2 oraz Windows 2000 z Service Pack 4. Na rysunku 5 przedstawiono stos sterowników dla urządzeń magazynowania danych bazujący na mechanizmie zarządcy filtrów.

Sterowniki zgodne z nowym modelem określane są mianem mini-filtrów. Jedną z najważniejszych cech tego modelu sterowników jest fakt, że w procedurze rejestrującej je w systemie definiują, jakiego typu operacje we/wy będą realizować. Ponadto, mini-filtry mogą określić, czy monitorują ruch pakietów w dół, czy w górę stosu sterowników.



Rys. 4. Schemat stosu obiektów dla urządzenia magazynowania podłączonego do systemu za pomocą magistrali USB [3]

Mini-filtry rejestruje się u zarządcy filtrów z poziomu procedury *DriverEntry* (jest to punkt wejścia do sterownika – odpowiednik funkcji *main* w aplikacji), za pomocą funkcji *FltRegisterFilter*. W przypadku powodzenia, funkcja ta zwraca wskaźnik na obiekt mini-filtra zwany instancją (ang. *instances*) [8]. Komunikacja, ze sterownikiem *mini-filter* z poziomu trybu użytkownika, realizowana jest za pośrednictwem portu komunikacyjnego *mini-filtra* (ang. *mini-filter communication port*) tworzonego przez zarządcę filtrów.



Rys. 5. Model stosu sterowników dla urządzeń magazynowania w systemie Windows [6]

Port komunikacyjny powoływany jest z poziomu *mini-filtra* za pomocą funkcji *FltCreateCommunicationPort*. Parametry wywołania tej funkcji określają między innymi wskaźniki na punkty wywołań funkcji odpowiedzialnych za obsługę następujących zdarzeń [8]:

- ✓ nawiązania połączenia z trybu użytkownika (*ConnectNotifyCallback*),
- ✓ zakończenia połączenia z trybu użytkownika (*DisconnectNotifyCallback*),
- ✓ otrzymania wiadomości z trybu użytkownika (*MessageNotifyCallback*).

Aplikacja trybu użytkownika może komunikować się z *mini-filtrem*, wykorzystując funkcje *FilterSendMessage*, *FilterGetMessage* oraz *FilterReplyMessage*. Jeżeli port komunikacyjny nie jest dłużej potrzebny,

aplikacja trybu użytkownika powinna skasować uchwyt do obiektu reprezentującego nazwany port komunikacyjny za pomocą funkcji *ZwClose* lub *CloseHandle*. Przed wyładowaniem sterownika z systemu, *mini-filtr* powinien zamknąć port komunikacyjny za pomocą funkcji *FltCloseCommunicationPort* [8].

Zarządca filtrów udostępnia ponad sto pięćdziesiąt funkcji pomocniczych, dzięki czemu znacząco ułatwia tworzenie sterowników filtrujących systemu plików pod postacią *mini-filtrów*. Praktycznie każda operacja, której wykonanie w tradycyjnym modelu sterowników wymagało napisania specjalnej funkcji, z poziomu *mini-filtra* może być wykonana za pomocą jednego odwołania do funkcji pomocniczej.

3. Dobór algorytmu szyfrowania

Doboru algorytmu szyfrowania, zastosowanego w zaprezentowanym rozwiązaniu, dokonano uwzględniając następujące kryteria [3]:

- ✓ zapewnienie odpowiedniej szybkości działania.

  Zdecydowana większość urządzeń magazynowania danych podłączanych poprzez interfejs USB, oferuje transfery danych od kilkunastu (dla *PenDrive*'ów i kart pamięci) do kilkudziesięciu (dla zewnętrznych dysków twardych) megabajtów na sekundę,

- ✓ zapewnienie losowego dostępu do danych.

  Zarówno w przypadku implementacji szyfrowania na poziomie plików jak i woluminów, w kontekście szybkości należy rozważyć możliwość losowego dostępu do danych,

- ✓ zapewnienie możliwości szyfrowania wiadomości dowolnego rozmiaru (wyłączając oczywiście wiadomości o zerowej długości).

  Ma to szczególne znaczenie w przypadku szyfrowania na poziomie plików. Po pierwsze, zmiana rozmiaru plików jest skomplikowana pod względem implementacji programowej, po drugie, może mieć wpływ na zachowanie niektórych programów chroniących przed złośliwym kodem.

Biorąc pod uwagę powyższe kryteria do implementacji w przedstawionym rozwiązaniu wybrano algorytm Blowfish [12]. Za wyborem tego algorytmu, prócz przedstawionych powyżej kryteriów, przemawia jego prostota działania, gdyż wykorzystuje jedynie operacje dodawania i sumowania modulo 2 zmiennych 32-bitowych. Po drugie, na stronie autora algorytmu dostępne są kody źródłowe dla kilku popularnych języków programowania. Algorytm został

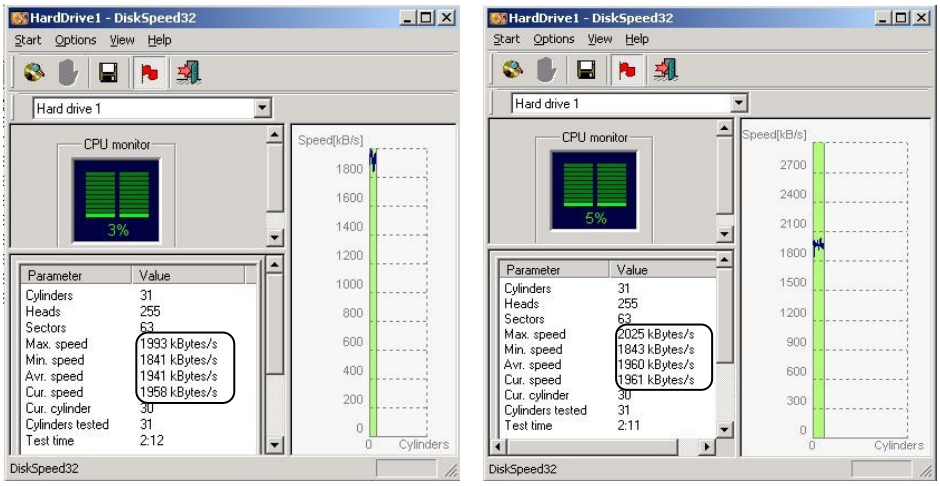
zaimplementowany do pracy w trybie licznikowym. Dzięki temu było możliwe zagwarantowanie:

- ✓ dużej szybkości szyfrowania i deszyfrowania danych,
- ✓ szyfrowania i deszyfrowania losowych obszarów pliku,
- ✓ niezmienności rozmiaru plików.

4. Podsumowanie

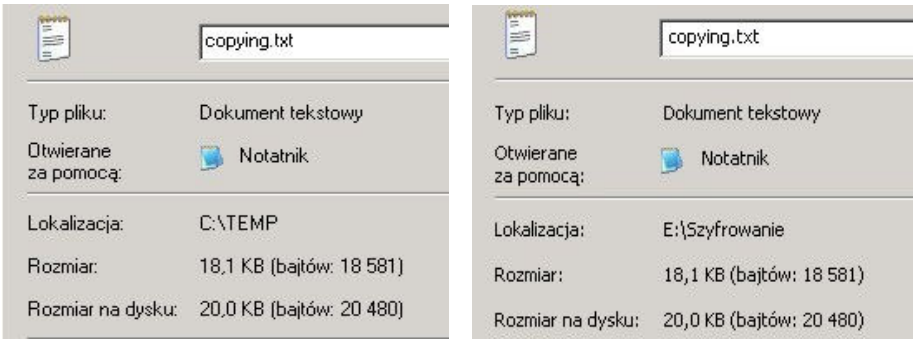
Zaprezentowane rozwiązanie pozwala na zabezpieczenie danych przechowywanych na dysku podłączonym do komputera poprzez interfejs USB. Mechanizm zabezpieczenia został zaimplementowany jako mini-filtr, czyli sterownik filtrujący systemu plików, zdefiniowany przez mechanizm zarządcy filtrów. W sterowniku wykorzystano kod źródłowy algorytmu Blowfish autorstwa Paula Kochera udostępniony na zasadzie licencji GNU LGPL (ang. *GNU Lesser General Public License*).

Opracowany sterownik przetestowano w środowisku wirtualnym utworzonym w oparciu VMware Sever ver. 1.0.10 build-203137 z zainstalowanym systemem Windows 2003 Server SP1. Uzyskane wyniki potwierdziły, że nie wpływa on na szybkość odczytu i zapisu danych (patrz ramka na rysunku 6) oraz rozmiar pliku (patrz rysunek 7).



a) b)

Rys. 6. Pomiar szybkości dostępu do dysku zewnętrznego
a) bez zainstalowanego sterownika b) z zainstalowanym sterownikiem



Rys. 7. Porównanie rozmiaru pliku umieszczonego a) na dysku bez szyfrowania b) na dysku szyfrowanym

Ponieważ sterownik jest w fazie testów i rozbudowy, w obecnym kształcie, niektóre jego parametry zostały jawnie określone w kodzie źródłowym (np. klucz szyfrowania) i w chwili obecnej nie ma możliwości ich zmiany bez ponownej kompilacji sterownika.

Literatura

[1] BROŻYNA M., *BitLocker - szyfrowanie dysków w Windows Vista*, MicrosoftTechNet http://www.microsoft.com/poland/technet/bazawiedzy/centrumrozwiazan/cr02801.mspx

[2] BURN D. D., *Getting Started with the Windows Driver Development Environment*, WinHEC 2005 Version, 18 kwietnia 2005.

[3] MALINOWSKI K., *Projekt i implementacja systemu bezpiecznej wymiany danych z wykorzystaniem portu USB*, Praca dyplomowa WAT 2006, kierownik Jan Chudzikiewicz.

[4] MSDN Library (October 2003 Release), Glossary: *Windows DDK*, Microsoft Corporation, Redmond, 2003.

[5] NAGAR R., OSR's Classic Reprints: *Windows NT File System Internals*, OSR Press, 2006.

[6] NAGAR R., *Filter Manager, Microsoft,* Redmond, 2003, http://download.microsoft.com/download/f/0/5/f05a42ce-575b-4c60-82d6-208d3754b2d6/Filter_Manager.ppt

[7] ONEY W., *Programming the Microsoft® Windows® Driver Model*, Microsoft Press, Redmond, 2003.

[8] PRACA ZBIOROWA, *Filter Driver Development Guide*, Microsoft, Redmond, 2005.

[9] RUSSINOVICH M. E., SOLOMON D. A., *Microsoft® Windows® Internals, Fourth Edition: Microsoft Windows Server™ 2003, Windows XP, and Windows 2000*, Microsoft Press, Redmond, 2005.

[10] SCHNEIER B., KELSEY J., WHITING D., WAGNER D., HALL C., FERGUSON N., *Performance Comparison of the AES Submissions*, Marzec 1999, http://www.schneier.com/paper-aes-performance.html

[11] SCHNEIER B., WHITING D.: *A Performance Comparison of the Five AES Finalists*, 2000, http://www.schneier.com/paper-aes-comparison.html

[12] SCHNEIER B., *The Blowfish Encryption Algorithm*, http://www.schneier.com/blowfish.html

Software Security for Files Stored on External Hard Drives

ABSTRACT: In the paper a software security solution for data stored on external USB hard drives is presented. The described solution takes advantages of a Filter-Driver for Windows and the Blowfish encrypted algorithm. An assumption for the presented solution was the compatibility with FAT32 (a possibility of applying tools for disk defragmentation) and to meet time requirements for data reading (audio and video files reading without delays).

KEY WORDS: device drivers, encryption algorithms, file system

Praca wpłynęła do redakcji: 08.10.2010.