УДК 681.3.06

ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ІНТЕГРАЦІЇ ПРОГРАМНИХ ЗАСОБІВ ЗАХИСТУ З ОС WINDOWS XP/2000/NT НА РІЗНИХ РІВНЯХ МЕРЕЖНОЇ АРХІТЕКТУРИ

Сергій Гладиш
Одеська національна академія зв'язку ім. О. С. Попова, Проектно-кошторисний відділ Кримської філії ВАТ «Укртелеком»

Анотація: **Проаналізовано мережну архітектуру ОС Windows XP/2000/NT. Визначено можливості інтеграції (встроювання) програмних засобів захисту на різних рівнях мережної архітектури. Порівняно ефективність варіантів їх реалізації за допомогою механізмів, які надає ОС Windows XP/2000/NT. Наведено вимоги до програмного засобу захисту та рекомендовані рівні його реалізації. Запропоновано структуру та варіант реалізації програмного засобу захисту.**
Summary: **Windows XP/2000/NT network architecture is analyzed. The opportunities of security software integration (embedding) at different network architecture levels are determined. The comparison of the effectiveness of their realization variants with the help of different mechanisms which gives Windows XP/2000/NT is done. The requirements to security software and recommended levels of their realization are given. The structure and a variant of security software realization is offered**
Ключові слова: **Програмний засіб захисту, ОС Windows XP/2000/NT, мережна архітектура, драйвер.**

I Вступ

Операційні системи сімейства Windows XP/2000/NT завдяки своїй універсальності, мережній архітектурі, можливостям розширення власної функціональності, іншим властивостям і зовнішнім факторам на даний момент є найбільш розповсюдженими. Ці та інші обставини зумовлюють зростаючу актуальність проблеми реалізації програмних засобів захисту, які могли б бути інтегрованими з ОС Windows XP/2000/NT, розширюючи її можливості щодо захисту інформації та найбільш ефективно використовуючи штатні механізми та сервіси безпеки, що надає ОС.

Аналізуючи стан наукових досліджень і публікацій, що стосуються проблемної області інформаційної безпеки, можна відмітити, що, практичної літератури професійного рівня з розробки інтегрованих програмних засобів захисту, драйверів безпеки, глибокого та докладного аналізу мережних інтерфейсів та архітектури ОС Windows XP/2000/NT з точки зору інформаційної безпеки, виданої українською чи російською мовами є дуже небагато. Основним, але не завжди повним, джерелом практичної інформації є англомовні довідкові джерела [2] – [5]. В багатьох випадках більш актуальним джерелом виявляються хакерські та крекерські сайти в мережі Internet. Знання та досвід хакерів-професіоналів завжди можна з користю застосувати при розробці системи захисту інформації.

Щодо тем даної статті – дослідження можливостей та ефективності інтеграції програмних засобів захисту інформації з ОС Windows XP/2000/NT на різних рівнях її мережної архітектури – то відповідної спеціалізованої літератури та наукових публікацій ані вітчизняних, ані зарубіжних авторів немає, або ця інформація не є відкритою. Окремо треба відмітити джерело [1], в якому було започатковано розв'язання даної проблеми і на яке спирався автор при проведенні власного дослідження.

II Постановка завдання

Мета даної роботи – проаналізувати можливості реалізації програмних засобів захисту інформації на різних рівнях мережної архітектури ОС Windows XP/2000/NT, починаючи від прикладного рівня і закінчуючи рівнем драйверів пристроїв. Аналіз проведений з наступних позицій:

- існування можливості інтеграції програмного засобу на тому чи іншому рівні мережної архітектури;
- складність реалізації програмного засобу з позицій недокументованості певних інтерфейсів ОС;
- прозорість механізмів захисту;
- відповідність програмного засобу структурі та об'єму інформації, яку він буде захищати;
- рівень можливостей, стандартних сервісів, функцій та механізмів, які може надати ОС програмному засобу захисту інформації.

При вирішенні цього завдання було використано програмний продукт компанії NuMega – «***SoftIce***» – засіб відладки, який працює нижче рівня ядра операційної системи і дозволяє відстежувати такі компоненти ядра, як драйвери.

III Мережна архітектура ОС Windows XP/2000/NT

Дослідження мережної архітектури ОС є необхідною процедурою для визначення можливостей інтеграції (встроювання) програмних засобів захисту, а також для порівняння ефективності варіантів їх реалізації за допомогою механізмів, які надає ОС. Таке дослідження має встановити, як і куди можна інтегрувати засіб захисту, яким чином цьому засобу надати максимальні можливості з боку ОС, оскільки від цього може залежати вирішення конкретних практичних завдань захисту, які він зможе реалізувати.

Мережну архітектуру ОС Windows XP/2000/NT схематично в узагальненому вигляді можна представити так, як це наведено на рис. 1.

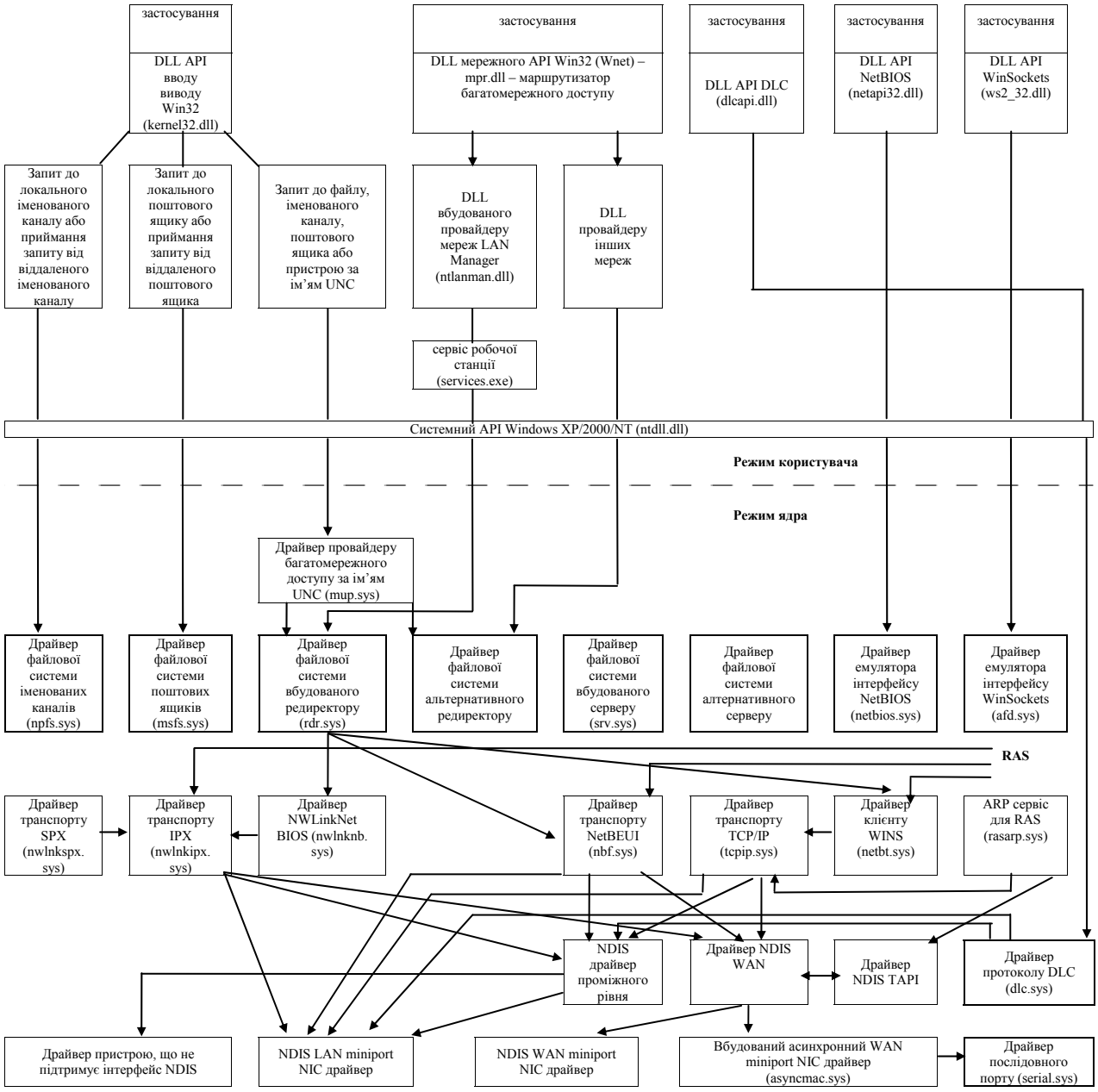

Рисунок 1 – Узагальнена схема мережної архітектури ОС Windows XP/2000/NT

ОС Windows XP/2000/NT на всіх рівнях мережної архітектури надає можливості інтеграції додаткових програмних модулів, які можуть контролювати запити мережних програмних компонентів відповідних рівнів і реалізовувати функції захисту інформації. Але в багатьох випадках виникає необхідність використовувати недостатньо документовані компанією Microsoft методи.

IV Порівняльний аналіз варіантів реалізації програмних засобів захисту

Розглянемо особливості реалізації програмних засобів захисту інформації на різних рівнях мережної архітектури ОС Windows XP/2000/NT. Під реалізацією захисту на будь-якому рівні будемо розуміти: чи можливо за допомогою розробки компонентів цього рівня реалізувати функції захисту, та чи можливо реалізувати програмний засіб захисту, який міг би впливати будь яким чином на вже існуючі компоненти даного рівня і виконувати функції захисту.

Реалізація захисту на рівні застосувань і власних DLL

Захист на рівні застосування не є прозорим. При запуску застосування створюється новий процес, адресний простір якого захищено від інших процесів. Тому застосування може захищати лише свої дані. Для того, щоб дані інших застосувань могли оброблятись застосуванням, яке реалізує захист цих даних, необхідно використовувати спеціальні механізми міжпроцесорної комунікації: копіювання даних з одного адресного простору в інший, або створення області, яка була б видимою з обох адресних просторів.

Реалізація захисту на рівні власної DLL також не забезпечує прозорого захисту. Для того, щоб захист даних застосувань виконувала бібліотека DLL, необхідно, щоб застосування викликало функції цієї DLL. Після того, як виконуваний код DLL буде спроектовано на адресний простір процесу, що викликає код і дані DLL стануть частиною процесу застосування.

Рівень застосувань і власних DLL не має всіх можливостей щодо реалізації функцій захисту мережних ресурсів. Тільки у взаємодії з власним драйвером застосування може стати ефективним засобом захисту.

Реалізація захисту на рівні системних DLL

До цих DLL відносяться бібліотеки, які надають застосуванням мережні інтерфейси:

- Netapi32.dll, Ws2_32.dll, Wsock32.dll, Mpr.dll, Kernel32.dll, Wininet.dll, Rpcrt4.dll, Rasapi32.dll, Nddeapi.dll, Tapi32.dll, Dlcapi.dll, Wsnmp32.dll, mgmtapi.dll, snmpapi.dll.
- Схема завантаження та використання функцій цих DLL дозволяє реалізувати захист на рівні системних DLL наступними способами:
- змінення в Import Address Table відповідних адрес на адреси власних оброблювачів;
- створення власної DLL з тим же ім'ям, в якій замінюється частина функцій;
- зміна коду системних DLL з метою передання керування (командами jump, call) власному коду.

Дані методи є недокументованими і дуже складними в реалізації. Але не дивлячись на це, програмні продукти, побудовані на цих методах, все ж існують, наприклад, ***«PrudensSpy»*** реалізує технологію перехоплення будь-яких DLL, в тому числі й системних.

Реалізація захисту на рівні системного API

Знаючи індексний номер функції, яка реалізує системний сервіс, можна по таблиці розподілу системних сервісів (KeServiceDescriptionTable), яка експортується з ntoskrnl.exe і обробляється перериванням int 2E, визначити адресу початку необхідної функції та замінити її на адресу початку власного оброблювача режиму ядра. Після виконання власного оброблювача необхідно повернути керування за старою адресою, щоб дати можливість стандартному оброблювачу виконати запитувані дії. Таблиця KeServiceDescriptionTable знаходиться у системній області пам'яті, тому така заміна може бути зроблена тільки кодом режиму ядра – драйвером. Приклад такого засобу захисту – «***RegMon***», який виконує перехоплення системних сервісів, які реалізують звернення до реєстру Windows.

Реалізація захисту на рівні драйверу *MUP*

Драйвер mup.sys відіграє важливу роль при відкритті файлів та пристроїв за ім'ям UNC. Тому доцільно розглядати можливості реєстрування та аудиту відкриття файлів та пристроїв за допомогою драйверу-фільтру, під'єднаного до драйверу mup.sys. Ідея цього методу вперше була запропонована в [1].

Реалізація захисту на рівні драйверів файлових систем

ОС Windows XP/2000/NT надають стандартний механізм для перехоплення запитів вводу/виводу до драйверу файлової системи – використання драйверів-фільтрів. Для його реалізації можна використовувати книгу [6], присвячену розробці драйверів файлових систем, зокрема реалізації драйверів-фільтрів. Завадою реалізації захисту на рівні драйверів файлових систем, і в тому числі драйверів-фільтрів до них, є закритість і недокументованість цього інтерфейсу. Навіть в [2] розробка драйверів файлових систем не документована. Вперше Microsoft пролила світло на це питання в 1997 році, випустивши [5] – інструмент для розробки драйверів файлових систем, що містить заголовочні файли для успішної компіляції драйверів за допомогою MS Visual C++, а також приклади файлових систем. Завдяки політиці Microsoft багато запитів до драйверів файлових систем лишаються загадкою, тому їх використання (за власним досвідом) призводить до некоректної роботи стеку драйверів. Вдалим прикладом реалізації фільтру до драйверів файлових систем є програма «FileMon», розроблена Mark Russinovich, Bruce Cogwell [7].

Реалізація захисту на рівні транспортного драйверу

Розробка таких драйверів добре документована в [4] – розділ Network Drivers, підрозділ Intermediate NDIS Drivers and TDI drivers. Є приклад транспортного драйверу в DDK\src\network\tdi. Крім того в принципі є можливість реалізувати драйвер-фільтр, який буде приєднано до об'єкту пристрою, котрий створюється драйвером транспорту. Наприклад до об'єктів-пристроїв \Device\Tcp та \Device\Udp, що створюються драйвером транспорту TCP/IP.

Реалізація захисту за допомогою перехоплення функцій бібліотеки NDIS

Ідея цього методу вперше запропонована в [1], і полягає в створенні оболонки над бібліотекою NDIS. Метод дуже важко реалізувати на практиці, він недокументований, але надзвичайно ефективний. Його реалізація полягає у зміні адрес необхідних функцій бібліотеки NDIS, в результаті можна отримати повний контроль над усіма мережними операціями в системі.

Реалізація захисту на рівні мережного драйверу проміжного рівня, що підтримує інтерфейс NDIS

Розробка проміжних драйверів добре документована в [4] – розділ Network Drivers, підрозділ Intermediate NDIS drivers and TDI drivers. Є приклад проміжного драйверу в DDK\src\network\packet\driver.

Розробка захисту на рівні драйверів мережних пристроїв

Розробка власних драйверів мережних пристроїв, що підтримують інтерфейс NDIS, добре документована в [4] – розділ Network Drivers підрозділ Miniport NIC drivers. Є багато прикладів в DDK\src\network.

Можемо провести порівняльний аналіз варіантів реалізації програмних засобів захисту відносно наступних факторів (табл. 1): об'єм інформації, що контролюється, складість реалізації, можливості інтеграції захисту, прозорість захисту, можливості.

Таблиця 1 – Порівняльний аналіз варіантів реалізації програмних засобів захисту

Рівень	Об'єм інформації, що контролюється	Складність реалізації	Інтеграція з ОС	Прозорість захисту	Можливості
Застосування	Тільки дані самого застосування	Низька	Є	-	Мінімальні
Власна DLL	Дані застосувань, які використовують цю DLL	Низька	Є	-	Мінімальні
Системна DLL	Дані застосувань, які використовують системну DLL	Висока	Немає	+	Залежно від способу реалізації
Мережний сервіс	Залежить від мережного сервіса	Висока	Немає	+	Залежно від способу реалізації
Системний API	Дані всіх застосувань	Висока	Немає	+	Максимальні
Драйвер файлової системи	Дані застосувань, які використовують відповідний мережний API	Висока	Є	+	Максимальні
Транспортний драйвер	Дані застосувань, які використовують цей транспорт + дані застосувань, які напряму взаємодіють з іншого транспорту	Висока	Є	+	Максимальні
Драйвер ndis.sys	Дані всіх застосувань	Висока	Немає	+	Максимальні
Проміжний драйвер	Дані застосувань, які використовують транспорти, прив'язані знизу до цього проміжного драйверу + застосувань, які напряму взаємодіють з проміжним драйвером	Середня	Є	+	Максимальні
Драйвер мережного пристрою	Дані всіх застосувань	Середня	Є	+	Максимальні

Об'єм інформації, що контролюється,. Цей фактор визначає, які дані проходять через засіб захисту і відповідно можуть їм контролюватися та оброблятися.

Складність реалізації. Визначає складність реалізації засобу захисту, з позиції як недокументованості, так і складності відладки. Низька складність означає, що при розробці використовуються добре документовані інтерфейси і доступні засоби відладки. Середня – використовуються документовані інтерфейси, але відладка засобів захисту не проста в зв'язку з тим, що даний засіб виконується в режимі ядра, більшість помилок в ньому призводять до краху (зависання) ОС, при відладці доводиться працювати з асемблером. Висока – до проблем з відладкою додається часткова або повна відсутність документації для розробки засобу захисту.

Можливості інтеграції захисту. Цей фактор визначає, чи дозволяє ОС можливість інтеграції (встроювання) засобу захисту й розширення своєї функціональності на даному рівні.

Прозорість захисту означає, чи має користувач виконати які-небудь дії для того, аби захистити свою інформацію за допомогою засобу захисту.

Можливості, які ОС надає засобу захисту. Залежно від того, чи виконується засіб захисту в режимі користувача, або в привілегіюваному режимі – режимі ядра, йому надаються різні можливості.

В режимі ядра дозволено виконання всіх команд процесора та доступна системна область пам'яті й обладнання, на відміну від користувацького режиму, в якому деякі команди заборонено, а системні області пам'яті є недосяжними. Засоби захисту рівня ядра можуть використовувати внутрішні інтерфейси для взаємодії з компонентами ОС. Модуль захисту, який виконується в режимі ядра, може підвищувати або знижувати IRQL процесора, приховуючи таким чином переривання з рівними або меншими IRQL.

Модуль захисту, який виконується в режимі користувача, не може отримати доступ до системних даних інакше, як, викликаючи функції, які надає ОС. Останні в свою чергу звертаються до системних сервісів, здійснивши попередню перевірку параметрів виклику.

V Вимоги до програмного засобу захисту інформації

Вибір того чи іншого варіанту реалізації програмного засобу захисту інформації залежить від попередньо сформульованих початкових вимог до конкретної захищуваної інформаційної системи.

Спираючись на попередній аналіз, можемо співставити найчастіші вимоги, що висуваються до програмних засобів захисту з найбільш ефективними рівнями їх реалізації (табл. 2).

Таблиця 2 – Залежність вибору варіанту реалізації програмного засобу захисту від вимог до захищуваної ІС

Вимоги	Рівень реалізації
Необхідно реалізувати програму, яка буде забезпечувати безпечний обмін даними мережею	Реалізація на рівні застосування та DLL
Необхідно забезпечити безпечний обмін даними мережею між застосуваннями, які використовують конкретний мережний API або забезпечити контроль доступу до мережного API	Реалізація на рівні системної DLL, яка надає даний API, або на рівні системного API, або на рівні відповідного драйверу файлової системи
Необхідно забезпечити контроль доступу до мережних сервісів	Реалізація на рівні мережного сервісу (яка зводиться до реалізації захисту на рівні системної DLL *rpcrt4* або системного API) або реалізація на рівні мережних драйверів для контролю за адресами
Необхідно забезпечити безпечний обмін даними мережею між застосуваннями, які використовують конкретний транспортний протокол або забезпечити аналіз трафіку по цьому протоколу	Реалізація на рівні драйверу *ndis.sys*, або драйверу транспорту, або проміжного драйверу, або драйверу пристрою
Необхідно забезпечити безпечний обмін даними мережею між будь-якими застосуваннями, або забезпечити аналіз всього трафіку в мережі	Реалізація на рівні драйверу *ndis.sys*, або проміжного драйверу, або драйверу пристрою

VI Структура та особливості реалізації програмного засобу захисту

При створенні програмного засобу захисту необхідно реалізувати наступні функціональні підсистеми:

- підсистему ідентифікації та автентифікації користувачів;
- підсистему контролю доступу;
- підсистему аудиту.

При цьому підсистему ідентифікації та автентифікації доцільно асоціювати зі штатною підсистемою ідентифікації та автентифікації ОС Windows XP/2000/NT, яка реалізується MSGINA.DLL, а підсистему контролю доступу та аудиту – з окремо працюючим драйвером контролю доступу, котрий повинен бути пов'язаний з підсистемою ідентифікації та автентифікації. Такий зв'язок необхідний тому, що контроль доступу до об'єктів до входу конкретного користувача в систему може виявитись досить нетривіальним завданням, наприклад, якщо для конкретного користувача задано ізольоване програмне средовище (фіксований перелік задач, які він може запускати), то незрозуміло як бути із задачами, котрі запускаються до виконання успішної процедури ідентифікації та автентифікації даного користувача. Крім того, зв'язок зі штатним модулем ідентифікації та автентифікації може служити для передачі ключової інформації, необхідної для ініціалізації персональних криптографічних модулів, призначених для шифрування трафіку або для періодичного тестування коректності роботи криптографічних функцій чи інших механізмів безпеки.

Розглянемо структуру програмного засобу захисту. Основними компонентами є:

- драйвер контролю доступу, котрий призначено для перехоплення файлових операцій на рівні ядра.
- модифікована бібліотека GINA (NEWGINA), побудована за принципом повного перехоплення функцій оригінальної MSGINA (Microsoft Graphical Identification aNd Authentication DLL for Winlogon), що експортуються

При вході користувача бібліотека GINA.DLL повертає директиви процесу Winlogon, згідно з якими воно або не повинно змінювати поточний статус процесу ідентифікації та автентифікації, або виконувати деякі дії (наприклад, примусово вигружати користувача). Залежно від значення параметра dwOptions в функції WlxLoggedOutSas() бібліотеки GINA.DLL, застосування Winlogon або не повинно завантажити профіль входящого користувача, або повинно виконувати завантаження того користувацького профілю, інформацію про який повернула бібліотека GINA.DLL.

Перехоплення операцій відкриття, створення та видалення файлів

Реалізація драйвером перехоплення файлових операцій ґрунтується на офіційно не документованому компанією Microsoft механізмі перехоплення системних сервісів. Принцип функціонування драйвера базується на механізмі заміни адрес функцій, які надаються системним API. Оброблювач переривання int2E для виклику відповідного мережного сервісу використовує таблицю розподілу системних сервісів KeServiceDescriptionTable, яка експортується ntoskrnl.exe.

Під час ініціалізації драйверу в функції DriverEntry після успішного створення об'єкту-пристрою встановлюються власні оброблювачі файлових операцій. Для цього спочатку необхідно отримати адресу таблиці розподілу системних сервісів:

```
ServiceTable = KeServiceDescriptionTable;
```

Знаючи індексний номер функції, яка реалізує системний сервіс, можливо по таблиці визначити адресу її початку та замінити її на адресу початку власного оброблювача режиму ядра:

```
#define SYSCALL (_function)
ServiceTable –> ServiceTable [*(PULONG) ((PUCHAR) _function + 1)]
RealCreateFile = SYSCALL (ZwCreateFile);      //оригінальний оброблювач створення файлу;
SYSCALL (ZwCreateFile) = (PVOID) HookCreateFile;      //власний оброблювач створення файлу;
RealOpenFile = SYSCALL (ZwOpenFile);      //оригінальний оброблювач відкриття файлу;
SYSCALL (ZwOpenFile) = (PVOID) HookOpenFile;      //власний оброблювач відкриття файлу;
```

При перехопленні функцій ZwCreateFile та ZwOpenFile необхідно мати на увазі, що операції запуску виконуваних файлів, перейменовування та видалення об'єктів проводяться саме цими функціями при встановлених наступних флагах:

```
DesiredAccess & DELETE
DesiredAccess & FILE_EXECUTE
DesiredAccess & GENERIC_EXECUTE
```

Окремим завданням є розрізнення операцій з об'єктами типу PIPE або COM-порт. Це можливо зробити, використовуючи ім'я , наприклад: memcmp (ANSIString.Buffer, "\\dosdevices\\com", 15) після виконання фрагменту коду:

```
UnString.Length = (USHORT)
  ObjectAttributes –> ObjectName –> Length;
UnString.MaximumLength = (USHORT)
  ObjectAttributes –> ObjectName –> MaximumLength;
UnString.Buffer = ObjectAttributes –> ObjectName –> Buffer;
RtlUnicodeStringToANSIString (&ANSIString, &UnString, TRUE);
```

Власні оброблювачі операції створення файлу та операції відкриття файлу

Власний оброблювач створення (відкриття) файлу може, наприклад, перевірити права доступу поточного процесу до створюваного (відкриваємого) файлу, і якщо доступ заборонено, то повернути статус створення (відкриття) файлу STATUS_ACCESS_DENIED (доступ заборонено), або в іншому випадку повернути керування оригінальному оброблювачу. При цьому можливо вести журнал вдалих і невдалих спроб доступу, регіструючи ім'я процесу, в контексті якого здійснювалась спроба отримати доступ до ресурсу, та ім'я створюваного (відкриваємого) файлу. Після виконання власного оброблювача необхідно повернути керування за старою адресою, щоб дати можливість стандартному оброблювачу виконати запитувані дії.

```
NTSTATUS HookCreateFile (OUT PHANDLE FileHandle,
IN ACCESS_MASK DesiredAccess,
IN POBJECT_ATTRIBUTES ObjectAttributes,
OUT PIO_STATUS_BLOCK IoStatusBlock,
IN PLARGE_INTEGER AllocationSize, IN ULONG FileAttributes,
IN ULONG ShareAccess, IN ULONG CreateDisposition,
IN ULONG CreateOptions, IN PVOID EaBuffer, IN ULONG EaLength)
{   If (Access (ObjectAttributes))
     Return RealCreateFile (FileHandle, DesiredAccess, ObjectAttributes, IoStatusBlock, AllocationSize,
FileAttributes, ShareAccess, CreateDisposition, CreateOptions, EaBuffer, EaLength)
     Return STATUS_ACCESS_DENIED;}
```

Встановлення імені процесу

Реалізація драйвером функції встановлення імені процесу, що звертається до ресурсу, необхідна для створення моделі повноважного (мандатного) доступу, за якою може бути визначена можливість контролю доступу звернень до ресурсу від імені конкретного процесу.

Ім'я процесу, котрий звертається до ресурсу, знаходиться в структурі, яка описує об'єкт-процес . Для того щоб отримати зміщення імені процесу в об'єкті-процесі, необхідно під час ініціалізації драйверу в функції DriverEntry (яка завжди виконується в контексті процесу SYSTEM) знайти строку «System» в об'єкті-процесі, який описує процес SYSTEM:

```
ULONG GetProcessNameOffset ()
{   PEPROCESS curproc;          int i;
   curproc = PsGetCurrentProcess ();
for (i = 0; i < 3 * PAGE_SIZE; i++) {
 if (!strncmp ("System", (PCHAR) curproc + i, strlen ("System")))
 {return i;}}       //ім'я не знайдене
    return 0;}
```

Після вдалого завершення цієї функції можливо використати повернене нею значення *ProcessNameOffset* для встановлення імені процесу:

```
VOID GetProcess (PCHAR Name)
{   PEPROCESS curproc;      char *nameptr;      ULONG i;
    If (ProcessNameOffset) {
        curproc = PsGetCurrentProcess ();
        nameptr = (PCHAR) curproc + ProcessNameOffset;
        strncpy (name, nameptr, 16);
   } else { strcpy (Name, "???"); }}
```

Отримавши ім'я процесу, можливо встановити його конкретні повноваги щодо доступу до об'єктів. Наприклад, далі буде використана можливість виклику функцій драйверу тільки процесом Winlogon, в контексті якого працює GINA. Таким чином можна гарантувати надання параметрів та керування драйвером тільки з боку модифікованої бібліотеки.

Процедура розподілу драйверу контролю доступу

Для забезпечення взаємодії з драйвером модифікована бібліотека GINA може використовувати певні контрольні коди, котрі повинен обробляти драйвер, наприклад:

```
#define TDRV_hook (ULONG) CTL_CODE (FILE_DEVICE_TDRV, 0x00, METHOD_BUFFERED,
FILE_ANY_ACCESS)
#define TDRV_unhook (ULONG) CTL_CODE (FILE_DEVICE_TDRV, 0x01, METHOD_BUFFERED,
FILE_ANY_ACCESS)
```

```
#define TDRV_setkey (ULONG) CTL_CODE (FILE_DEVICE_TDRV, 0x02, METHOD_BUFFERED,
FILE_WRITE_ACCESS)
   #define TDRV_test (ULONG) CTL_CODE (FILE_DEVICE_TDRV, 0x03, METHOD_BUFFERED,
FILE_WRITE_ACCESS)
```

Тоді процедура обробки контрольних кодів драйверу *TDRV* може мати наступний вигляд:

```
   BOOLEAN TDRVDeviceControl (IN PFILE_OBJECT FileObject, IN BOOLEAN Wait,
  IN PVOID InputBuffer, IN ULONG InputBufferLength,
  OUT PVOID OutputBuffer, IN ULONG OutputBufferLength,
  IN ULONG IoControlCode, OUT PIO_STATUS_BLOCK IoStatus,
  IN PDEVICE_OBJECT DeviceObject)
{ UCHAR ProcName[256];        int tg_gina_work, i;
IoStatus –> Status = STATUS_SUCCESS;
IoStatus –> Information = 0;
Switch (IoControlCode) {
case TDRV_version:
  *(ULONG *) OutputBuffer = TDRVversion;
  IoStatus –> Information = sizeof (ULONG);  Break;
case TDRV_hook:
  HookFileOperation ();  Break;
case TDRV_unhook:
  UnhookRegistry ();  Break;
case TDRV_test:
  { tg_no_work = 0;
  If ( test_crypto () ! = 0) tg_no_work = 1;  break;
case TDRV_setkey:
{ tg_gina_work = 0;
GetProcess (ProcName);        ToLowerStr (ProcName);
If (strcmp (ProcName, "winlogon.exe") = = 0) tg_gina_work = 1;
If ((InputBufferLength != sizeof (TDRV_IOCTL)) || (InputBuffer = = NULL) || (tg_gina_work = = 0))
  {IoStatus –> Status = 1;  Break;}
  RtlMoveMemory (UserKey, ((PTDRV_IOCTL) InputBuffer) –> UserKey, KEY_SIZE);
  RtlMoveMemory (UserID, (PTDRV_IOCTL) InputBuffer) –> UserID, USER_ID_SIZE);
  RtlMoveMemory (UserStatus, ((PTDRV_IOCTL) InputBuffer) –> UserStatus, USER_STATUS_SIZE);
  IoStatus –> Status = STATUS_SUCCESS;  Break;}
default: IoStatus –> Status = STATUS_INVALID_DEVICE_REQUEST;  Break; }
   return TRUE;}
```

Драйвер може викликатись періодично за таймером з модифікованої бібліотеки GINA за допомогою функції DeviceIOControl для тестування криптографічних функцій та інших функцій безпеки із встановленою періодичністю.

Модифікована бібліотека *GINA*

Модифікована бібліотека GINA NEWGINA.dll реалізує перехоплення функцій, що експортуються оригінальною бібліотекою MSGINA з наступним наданням їм керування. Власні оброблювачі повинні мати ті ж самі імена, що й оригінальні (що експортуються бібліотекою MSGINA).

NEWGINA.dll під час завантаження в процес Winlogon повинна спочатку підгрузити оригінальну бібліотеку MSGINA.dll, а потім отримати адреси функцій, що експортуються цією бібліотекою, для того щоб використовувати ці адреси у власних оброблювачах.

Файл експорту:

```
   LIBRARY NEWGINA
   DESCRIPTION    'Modified Windows XP/2000/NT Logon Graphical User Interface'
EXPORTS
  WlxNegotiate
  WlxInitialize
  WlxDisplaySASNotice
  WlxLoggedOutSAS
  WlxActivateUserShell
  WlxLoggedOnSAS
```

```
   WlxDisplayLockedNotice
   WlxWkstaLockedSAS
   WlxIsLockOk
   WlxIsLogoOffOk
   WlxLogoOff
      WlxShutdown
```

Файл *NEWGINA.h*

```
    #ifndef GINA_GINA_H_
#define GINA_GINA_H
Extern HINSTANCE                          hDllInstance;
Extern HANDLE                             hGlobalWlx;
Extern PWLX_DISPATCH_VERSION_1_0    pWlxFuncs;
typedef struct _Globals
{  BOOL            fAutoLogonAtBoot;
   BOOL            fAutoLogonAlways;
   HANDLE          hUserToken;
   PWSTR           pszUserName;
   PWSTR           pszDomain;
   PWSTR           pszPassword;
   SYSTEMTIME   timeOfLogin; }
Globals,  *PGlobals;
    #endif // GINA_GINA_H_
```

Для того, щоб замість оригінальної бібліотеки MSGINA.dll в процес Winlogon підгружалась модифікована бібліотека NEWGINA.dll, треба задати в ключі реєстру
\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\Current Version\Winlogon
запис на зразок: "*GinaDll = C:\WINDOWS\system32\NEWGINA.dll*".

VII Висновки

З приведеного вище аналізу мережної архітектури ОС Windows XP/2000/NT та порівняльної характеристики варіантів реалізації програмних засобів захисту інформації можна зробити висновок, що варіанти реалізації засобу захисту на рівні проміжних драйверів та драйверів мережних пристроїв є найбільш прийнятними. Розробка таких драйверів добре документована, й ОС Windows XP/2000/NT надає можливості щодо їх інтеграції до стеку мережних драйверів без необхідності в модифікації вже існуючих компонентів. Вони можуть взаємодіяти з компонентами виконавчої системи, ядром та HAL, отримуючи тим самим максимальні можливості щодо реалізації функцій захисту. Ці драйвери спроможні контролювати весь мережний трафік, через них проходять дані всіх застосувань, які виконуються в системі.

На всіх рівнях мережної архітектури ОС Windows XP/2000/NT дозволяє розширювати свої можливості щодо реалізації та інтеграції програмних засобів захисту інформації. Навіть там, де мережна архітектура ОС Windows XP/2000/NT надає можливість розширення своєї функціональності, компанія Microsoft не завжди документує таку можливість.

Література: 1. Программирование драйверов и систем безопасности: Учеб. пособие / Сорокина С. И., Тихонов А. Ю., Щербаков А. Ю. – СПб.: БХВ-Петербург, 2003. – 256 с.: ил. ***2.*** *Microsoft Software Developer's Network (MSDN).* ***3.*** *Microsoft Platform Software Development Kit.* ***4.*** *Microsoft Windows NT Device Driver Kit.* ***5****. Microsoft Windows NT Installable File System Kit.* ***6.*** *Rajeev Negar. Windows NT File System Internals: Developer's guide. USA, O'Reilly & Associate, Inc., 1997. 7. http://www.sysinternals.com/ntdll.htm. Mark Russinovich. Inside the native API.*