МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Центральноукраїнський національний технічний університет

Кафедра кібербезпеки та програмного забезпечення

На правах рукопису

Федоров Богдан Сергійович

Програмне забезпечення системи кібербезпеки для аудиту IT інфраструктури методом «Білий ящик»

Спеціальність: 125 «Кібербезпека»

Освітній ступінь: бакалавр

Науковий керівник:

Смірнов Сергій Анатолійович _____

(підпис) (дата)

кандидат технічних наук

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри

_____ О.А. Смірнов

(підпис) ПІБ

«____»_____2021 р.

Кропивницький – 2021

Міністерство освіти і науки України
Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Освітній ступінь *бакалавр*
Спеціальність *125 Кібербезпека*

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____О.А.Смірнов
« 11 » січня 2021 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Федорову Богдану Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення системи кібербезпеки для аудиту IT інфраструктури методом «Білий ящик»*

керівник роботи *Смірнов Сергій Анатолійович, канд. техн. наук*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 185-02 від 28.12.2020 року

2. Строк подання студентом роботи до захисту *22.05.2021 р.*

3. Мета та завдання кваліфікаційної бакалаврської роботи*: Метою розробки є програмне забезпечення системи кібербезпеки для аудиту IT інфраструктури методом «Білий ящик»*

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)

Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

6. Дата видачі завдання « 11 » січня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної бакалаврської роботи	Строк виконання етапів кваліфікаційної бакалаврської роботи	Примітка
1.	Аналіз існуючих систем	10.03.2021 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2021 р.	
3.	Розробка моделі компонента	20.03.2021 р.	
4.	Розробка структур даних	25.03.2021 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2021 р.	
6.	Програмування алгоритмів	10.04.2021 р.	
7.	Оформлення ПЗ	17.04.2021 р.	
8.	Попередній захист роботи	14.05.2021 р.	

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Федоров Б.С. Програмне забезпечення системи кібербезпеки для аудиту IT інфраструктури методом «Білий ящик». 125 Кібербезпека. Центральноукраїнський національний технічний університет. Кропивницький. 2021.

В даній кваліфікаційній бакалаврській розроблено програмне забезпечення, яке призначено для системи кібербезпеки для аудиту IT інфраструктури методом «Білий ящик».

Метою розробки є програмне забезпечення системи кібербезпеки для аудиту IT інфраструктури методом «Білий ящик».

Результат роботи – програмна реалізація системи кібербезпеки для аудиту IT інфраструктури методом «Білий ящик».

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ архітектури IBM PC з ОС Windows XP/Vista/7/8/10.

Програму розроблено в середовищі RAD Studio Delphi 10.4.1.

Ключові слова: кібербезпека, аудит IT інфраструктури

ABSTRACT

Fedorov B.S. Cybersecurity system software for IT infrastructure audit using the White Box method. 125 Cybersecurity. Central Ukrainian National Technical University. Kropyvnytskyi. 2021

In this bachelor's qualification, software has been developed that is designed for a cybersecurity system for auditing IT infrastructure using the White Box method.

The purpose of the development is cybersecurity system software for IT infrastructure audit using the White Box method.

The result is a software implementation of a cybersecurity system for auditing the IT infrastructure using the White Box method.

In the process of working on the software model, an analysis of existing hardware and software was performed. All components of the developed software are fully described.

Developed user-friendly interface. Instructions for working with software are given.

The program can be used on an IBM PC with Windows XP / Vista / 7/8/10.

The program is developed in the environment of RAD Studio Delphi 10.4.1.

Keywords: cybersecurity, IT infrastructure audit

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ 2
ВСТУП..... 3
1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ 5
1.1 Призначення системи..... 5
1.2 Область застосування 6
2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ 9
2.1 Огляд існуючих систем, технологій, архітектур та програмних рішень за профілем теми кваліфікаційної бакалаврської роботи..... 9
2.2 Обґрунтування вибору засобів для побудови системи та мови програмування 35
2.3 Розгорнута постановка завдання 41
3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ 43
3.1 Опис функціонування системи 43
3.2 Розробка структурної схеми..... 46
3.3 Розробка функціональної схеми 48
3.4 Розробка діаграми процесів 50
4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ 53
4.1 Розробка блок-схем та опис алгоритмів функціонування системи 53
4.2 Захист розробленого програмного забезпечення.....
5 ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ..... 65
6 ОСНОВНІ ВИСНОВКИ 69
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 71

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

БД	–	база даних
ЛОМ	–	локальна обчислювальна мережа
ASP	–	Active Server Pages – активні серверні сторінки
DHCP	–	Dynamic Host Configuration Protocol – протокол динамичної конфігурації вузла
HTTP	–	HyperText Transfer Protocol – протокол передачі гіпер тексту
IMAP	–	Internet Message Access Protocol – протокол доступу до електроної пошти Інтернету
ICMP	–	Internet Control Message Protocol – міжмережний протокол керуючих повідомлень
MMC	–	Microsoft Management Console
POP3	–	Post Office Protocol Version 3 – протокол поштового відділення, версія 3
SQL	–	Structured Query Language – мова структурованих запросів
SMTP	–	Simple Mail Transfer Protocol – простий протокол передачі пошти
SNMP	–	Simple Network Management Protocol – простий протокол керування мережею
Syslog	–	стандарт відправки повідомлень про зміни які відбуваються в мережі
UDP	–	User Datagram Protocol – протокол користувальницьких дейтаграм

ВСТУП

Актуальність теми. Існує велика кількість термінів (тестування на проникнення, аналіз захищеності, редтиммінг), що відносяться до оцінки інформаційної безпеки, що й позначають подібні, хоч і різні поняття. Дана робота допоможе структурувати наявні на теперішній момент підходи й вибрати той, який найбільше відповідає поставленим завданням.

Відразу хотілося б обмовитися, що інформація, наведена нижче, по більшій частині є переробкою/адаптацією методик, описаних в Open Source Security Testing Methodology Manual (OSSTMM) і NIST Special Publication 800-115 Technical Guide to Information Security Testing and Assessment.

У першу чергу власникові інформаційної системи необхідно визначити, що й з якою метою буде виконуватись оцінка. Глобально ціль звичайно ставиться скоріше до одному із двох варіантів:

– комплаєнс або відповідність ресурсу тем або іншим вимогам по безпеці (це може бути вимоги по безпеці, пропоновані до PCI DSS);

– оцінка безпеки, викликана власною потребою, що виходить із моделі потенційного порушника (може варіюватися від конкурента, що здійснює промислове шпигунство, до мінімально технічно підкованого школяра, що розважається у вільний час).

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи кібербезпеки для аудиту IT інфраструктури методом «Білий ящик».

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем кібербезпеки для аудиту IT інфраструктури методом «Білий ящик».

– Дослідження системи кібербезпеки для аудиту IT інфраструктури методом «Білий ящик».

– Програмна реалізація системи кібербезпеки для аудиту ІТ інфраструктури методом «Білий ящик».

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі для аудиту ІТ інфраструктури методом «Білий ящик».

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки для аудиту ІТ інфраструктури методом «Білий ящик», є актуальною задачею, яка потребує вирішення у даній кваліфікаційній бакалаврській роботі.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Виходячи з поставленої мети, вибирається один з декількох типів проведення робіт, що діляться по кількості вихідної інформації, відомої як аудиторові з одного боку, так і персоналу організації, пов'язаному з IT-інфраструктурою, з іншої:

– «білий ящик» – обидві сторони мають повну інформацію як про досліджувану систему, так і про проведені роботи – може проводитися для оцінки достатності механізмів захисту, розгляду необхідності впровадження додаткових заходів (наприклад, при проектуванні системи), або при проведенні аудита на відповідність конкретним вимогам;

– «чорний ящик» – обидві сторони мають мінімальну необхідну інформацію (зокрема навіть пошук цілей для дослідження відбувається шляхом розвідки на основі відкритих джерел і лише потім узгодиться із власником системи) – дозволяє визначити: наскільки досліджуваний об'єкт справляється з реальними можливими сценаріями атак, рівень складності, який зловмисникові прийде подолати для компрометації системи, додаткові контрзаходи, що дозволяють мінімізувати ризики, у випадку успішної реалізації уразливості, здатність захисних механізмів виявляти подібні атаки й реагувати належним чином;

– «сірий ящик» – щось середнє між двома попередніми типами, коли обом сторонам відома лише деяка частина інформації, але не мінімальна – звичайно вибирається при проведенні регулярних планових перевірок за аналізом захищеності, у т.ч. для самоперевірок;

– «сліпий» аудит – коли дослідникові не надається ніякій інформації, а він у свою чергу докладно звітує про всі плановані дії-може проводиться в якості

тренування по відточуванню навичок команди, відповідальної за реакцію на інциденти ІБ, або з метою оцінки компетентності аудитора;

– «зворотний» аудит – аудиторові надається докладна інформація про систему, тоді як співробітникам організації й службі безпеки зокрема може бути невідомо навіть про самий факт проведення аудита-проводиться для оцінки достатності й коректності налаштування засобів захисту, що фіксують інциденти ІБ, і підготовленості до реагування на них служби безпеки.

1.2 Область застосування

Крім вищеописаного, при плануванні робіт повинні бути визначені канали взаємодії з оцінюваною системою. З них можуть випливати:

– оцінка фізичної безпеки може розглядати такі атаки, як проникнення на контрольовану територію в обхід пропускного режиму, фізичне «врізання» у мережу, крадіжку встаткування і так далі. Також подібна оцінка містить у собі різні методи соціальної інженерії, наприклад, здійснення запиту пароля користувача під видом фахівця технічної підтримки;

– оцінка бездротових каналів зв'язку може включати всі види електронних комунікацій, пов'язаних із сигналами й випромінюваннями у відомому електромагнітному спектрі. У першу чергу сюди варто віднести тестування таких повсюдне використовуваних технологій, як Wi-Fi і Bluetooth;

– оцінка комунікацій. У цьому випадку розглядається атака на систему через інформаційно-телекомунікаційні мережі.

Також при проведенні аудита можна виділити наступні методи взаємодії з досліджуваним об'єктом:

1. Експертні оцінки розпорядчої документації, прийнятих політик, журналів подій, конфігурацій систем, аналіз трафіку мережі передачі даних, вихідних кодів програмного забезпечення.

2. Ідентифікація й аналіз об'єктів, мереж, систем, відкритих портів, запущених сервісів.

3. Експлуатація виявлених при роботі з об'єктом двома попередніми методами проблем у безпеці з метою наочної демонстрації реальних можливостей, одержуваних зловмисником, а також для виключення ложнопозитивних і ложнонегативних результатів.

Залежно від типу проведених робіт можуть бути обрані різні методи взаємодії, а також їх комбінації. Наприклад, при проведенні планової перевірки за аналізом захищеності, швидше за все буде використовуватися другий метод, а при класичному тестуванні на проникнення другий і третій. У свою чергу перший спосіб далеко не завжди можливий, але обов'язково буде використовуватися при проведенні робіт із принципу «білого ящика».

Виходячи з моделі потенційного порушника, аудит також може ділитися на зовнішній і внутрішній. При зовнішньому аудитові розглядаються атаки, що розвиваються із точки, що перебуває за межами периметра безпеки організації, і, відповідно, розглядається зовнішній зловмисник. При внутрішньому ж аудитор перебуває в межах контрольованої зони: розглядаються внутрішній зловмисник або зовнішній, що зумів проникнути крізь периметр безпеки – у цьому випадку аудиторові звичайно видаються мінімальні права в системі й однієї із завдань ставиться аналіз можливості підвищення привілеїв.

В остаточному підсумку, на основі вищевикладеного ми одержуємо можливість зручного й зрозумілого як виконавцеві, так і замовникові формування переліку проведених робіт. Як приклад, приведу найбільш популярні види аудитів:

– Оцінка відповідності вимогам по безпеці – «Білий ящик» – Залежить від стандарту, на відповідність якому проводиться аудит. Можуть розглядатися всі перераховані канали. – Експертні оцінки – завжди. Ідентифікація й аналіз – рідше. Експлуатація уразливостей – зовсім рідко. – Система розглядається з обох позицій

– Оцінка потенційного збитку від реальної атаки – «Чорний ящик», рідше «сірий ящик» або «зворотний аудит» – Звичайно використовується канал комунікацій, рідше бездротові канали зв'язку, ще рідше фізична безпека – Ідентифікація й аналіз, експлуатація – Найчастіше аудит проводиться зовнішній

– Виконання вимог стандартів по безпеці в частині періодичних перевірок – «Сірий ящик» – Звичайно використовується канал комунікацій, але в цілому залежить від стандарту – Ідентифікація й аналіз, рідше – експлуатація – Залежить від вимог

Нові уразливості виявляються постійно. Таким чином, захист інформації в компанії повинна стати безперервним процесом, а не разовим заходом. Тести на проникнення необхідно робити як мінімум раз у рік, у цьому вам можуть допомогти наші фахівці. За результатами тестування на проникнення буде наданий детальний звіт з виявленими уразливостями, рекомендаціями з їхнього усунення, прикладами атак і описами можливих сценаріїв проникнення.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки для аудиту ІТ інфраструктури методом «Білий ящик», є актуальною задачею, яка потребує вирішення у даній кваліфікаційній бакалаврській роботі.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми кваліфікаційної бакалаврської роботи

Тестування білого ящика

Розробка програм високої якості має на увазі, що програма і її частини зазнають тестуванню. Класичне модульне (unit) тестування має на увазі розбивка великої програми на маленькі блоки, зручні для тестів. Або, якщо розробка тестів відбувається паралельно з розробкою коду або тести розробляються до програми (TDD – test driven development), то програма споконвічно розробляється невеликими блоками, що підходять під вимоги тестів.

Однієї з різновидів модульного тестування можна вважати propery-based testing (такий підхід реалізований, наприклад, у бібліотеках Quickcheck, Scalacheck). Цей підхід заснований на знаходженні універсальних властивостей, які повинні бути слушні для будь-яких вхідних даних. Наприклад, серіалізація з наступної десеріалізацією повинна давати такий же об'єкт. Або, повторне сортування не повинна міняти порядок елементів у списку. Для перевірки таких універсальних властивостей у вищезгаданих бібліотеках підтримується механізм генерації випадкових вхідних даних. Особливо добре такий підхід працює для програм, заснованих на математичних законах, які служать універсальними властивостями, слушними для широкого класу програм. Є навіть бібліотека готових математичних властивостей – discipline –, що дозволяє перевірити виконання цих властивостей у нових програмах (гарний приклад повторного використання тестів).

Іноді виявляється, що необхідно протестувати складну програму, не маючи можливості розібрати її на частини, що незалежно перевіряються. У

такому випадку тестуєма програма являє собою білий ящик (білий – тому що ми маємо можливість вивчати внутрішній пристрій програми).

Під катом описано кілька підходів до тестування складних програм з одним входом з різним ступенем складності (залучення) і різним ступенем покриття.

У цій роботі припускаємо, що тестуєму програму можна представити у вигляді чистої функції без внутрішнього стану. (Деякі міркування, наведені далі, можна застосовувати й у тому випадку, якщо внутрішній стан присутнє, але є можливість скидання цього стану до фіксованого значення.)

Тестовий стенд (test bench)

Насамперед, тому що тестується всього одна функція, код виклику якої завжди однаковий, то в нас немає необхідності створювати окремі unit test'и. Усі такі тести були б однаковими з точністю до вхідних даних і перевірок. Цілком достатньо в циклі передавати вихідні дані (input) і перевіряти результати (expectedoutput). Щоб у випадку виявлення помилки можна було ідентифікувати проблемний набір тестових даних, усі тестові дані треба постачити міткою (label). Таким чином, один набір тестових даних можна представити у вигляді трійки:

```
case class Testcase[A, B](label: String, input: A, expectedoutput: B)
```

Результат одного прогону можна представити у вигляді Testcaseresult:

```
case class Testcaseresult[A, B](testcase: Testcase[A, B], actualoutput: Try[B])
```

Результат запуску ми представляємо за допомогою Try, щоб захопити можливі виключення)

Щоб спростити прогін усіх тестових даних через тестуєму програму, можна використовувати допоміжну функцію, яка буде викликати програму для кожного вхідного значення:

```
def runtestcases[A, B](cases: Seq[Testcase[A, B]])(f: A => B): Seq[Testcaseresult[A, B]] =
  cases
    .map{ testcase =>
```

```
        Testcaseresult(testcase,
          Try{ f(testcase.input) }
        )
      }
      .filter(r => r.actualoutput != Success(r.testcase.expectedoutput))
```

Ця допоміжна функція поверне проблемні дані й результати, які відрізняються від очікуваних.

Для зручності можна відформатувати результати тестування

```
def report(results: Seq[Testcaseresult[_, _]]): String =
  s"Failed ${results.length}:\n" +
    results
      .map(r => r.testcase.label + ": expected " + r.testcase.expectedoutput + ", but got " + r.actualoutput)
      .mkstring("\n")
```

і виводити звіт тільки у випадку помилок:

```
val testcases = Seq(
  Testcase("1", 0, 0)
)
test("all test cases"){
  val testbench = runtestcases(testcases) _
  val results = testbench(f)
  assert(results.isempty, report(results))
}
```

Підготовка вхідних даних

У найпростішому випадку можна вручну створити тестові дані для перевірки програми, записати їх прямо в тестовому коді, і використовувати, як продемонстровано вище. Часто виявляється, що цікаві випадки тестових даних мають багато загального й можуть бути представлені як деякий базовий екземпляр, з невеликими змінами.

```
val baseline = Myobject(...) // вхідний об'єкт можна створити вручну або згенерувати
val testcases = Seq(
```

```
  Testcase("baseline", baseline, ???),
  Testcase("baseline + (field1 = 123)", baseline.copy(field1 =
"123"), ???)
)
```

При роботі із вкладеними незмінними структурами даних більшою підмогою є лінзи, наприклад, з бібліотеки Monocle:

```
val baseline = ???
val testobject1 = (field1 composelens
field2).set("123")(baseline)
// що еквівалентно наступній рядку:
val testobject1 = baseline.copy(field1 =
baseline.field1.copy(field2 = "123"))
```

Лінзи дозволяють елегантно "модифікувати" глибоко вкладені частини структур даних: Кожна лінза являє собою getter і setter для одного властивості. Лінзи можна з'єднувати й одержувати лінзи, "які фокусуються" на наступному рівні.

Використання DSL для вистави змін

Далі будемо розглядати формування тестових даних шляхом внесення змін у деякий вихідний вхідний об'єкт. Звичайно для одержання потрібного нам тестового об'єкта потрібно внести кілька змін. При цьому досить корисно в текстовий опис Testcase'а включити перелік змін:

```
val testcases = Seq(
  Testcase("baseline", baseline, ???),
  Testcase("baseline + " +
              "(field1 = 123) + " +  // опис 1-го зміни
              "(field2 = 456) + " +  // 2-го
              "(field3 = 789)",      // 3-го
           baseline
             .copy(field1 = "123")   // 1-е зміна
             .copy(field2 = "456")   // 2-е зміна
             .copy(field3 = "789"),  // 3-е зміна
           ???)
)
```

Тоді ми завжди будемо знати, для яких тестових даних виконується тестування.

Щоб текстовий перелік змін не розходився з фактичними змінами, необхідно додержуватися принципу " єдиної версії правди". (Якщо та сама інформація потрібно/використовується в декількох точках, то слід мати єдине первинне джерело унікальної інформації, а в усі інші точки використання інформація повинні поширюватися автоматично, з необхідними перетвореннями. Якщо цей принцип порушувати, і копіювати інформацію вручну, то неминуча розбіжність версій інформації в різних точках. Тобто в описі тестових даних ми побачимо одне, а в тестові даних – інше. Наприклад, копіюючи зміну field2 = "456" і коректуючи його в field3 = "789" ми можемо випадково забути виправити опис. У підсумку опис буде відбивати тільки дві зміни із трьох.)

У нашому випадку первинним джерелом інформації є самі зміни, вірніше, вихідний код програми, яка вносить зміни. Нам хотілося б вивести з них текст, що описує зміни. Навскидку, у якості першого варіанта, можна запропонувати використовувати макрос, який буде захоплювати вихідний код змін, і використовувати вихідний код як документацію. Це, очевидно, гарний і відносно нескладний спосіб задокументувати фактичні зміни й він цілком може застосовуватися в деяких випадках. На жаль, якщо ми представляємо зміни у вигляді простого тексту, ми втрачаємо можливість виконувати осмислені трансформації переліку змін. Наприклад, виявляти й усувати, що дублюються або перекриваються зміни, оформляти перелік змін зручним для кінцевого користувача способом.

Щоб мати можливість оперувати змінами, необхідно мати їхню структуровану модель. Модель повинна бути досить виразної, щоб описувати всі зміни, що цікавлять нас. Частиною цієї моделі, наприклад, буде адресація полів об'єктів, константи, операції присвоювання.

Модель змін повинна дозволяти вирішувати наступні завдання:

1. Породження екземплярів моделі змін. (Тобто фактично створення конкретного списку змін.)

2. Формування текстового опису змін.

3. Застосування змін до об'єктів предметної області.

4. Виконання оптимізаційних перетворень над моделлю.

Якщо для внесення змін буде використовуватися універсальна мова програмування, то можуть виникнути утруднення для того, щоб представити ці зміни в моделі. У вихідному тексті програми можуть використовуватися складні конструкції, які не підтримуються моделлю. Така програма для зміни полів об'єкта може використовувати вторинні паттерни, на зразок лінз або методу сору, які є більш низькорівневими абстракціями стосовно рівня моделі змін. У результаті, для виводів екземплярів змін може знадобитися додатковий аналіз таких паттернів. Тим самим, споконвічно непоганий варіант із використанням макросу, виявляється не дуже зручним.

Іншим способом формування екземплярів моделі змін може служити спеціалізована мова (DSL), що створює об'єкти моделей зміни за допомогою набору extension-методів і допоміжних операторів. Ну а в найпростіших випадках екземпляри моделі змін можна створювати безпосередньо, через конструктори.

Подробиці мови змін

Такий підхід у цілому працює, і дійсно дозволяє описувати зміни один раз, однак поступово з'являється потреба предвідносити усе більш складні зміни й модель змін дещо розростається. Наприклад, якщо необхідно змінити якусь властивість із використанням значення іншої властивості того ж об'єкта (наприклад, field1 = field2 + 1), те виникає необхідність у підтримки змінних на рівні DSL. А якщо зміна властивості нетривіально, те на рівні DSL буде потрібно підтримка арифметичних виражень і функцій.

Тестування галузей

Тестуємий код мо же бути лінійним, і тоді нам по великому рахункові досить одного набору тестових даних, щоб зрозуміти, чи працює він. У випадку наявності розгалуження (if-then-else), необхідно запускати білий ящик як мінімум двічі з різними вхідними даними, щоб були виконано обидві вітки. Кількість наборів вхідних даних, достатніх для покриття всіх галузей, очевидно, чисельно рівно цикломатичной складності коду з розгалуженнями.

Як сформувати всі набори вхідних даних? Тому що ми маємо справу з білим ящиком, то ми можемо вичленувати умови розгалуження й двічі модифікувати вхідний об'єкт так, щоб в одному випадку виконувалася одна галузі, в іншому випадку – інша. Розглянемо приклад:

```
if (object.field1 == "123") A else B
```

Маючи таку умову, ми можемо сформувати два тестові випадки:

```
val testcase1 = Testcase("A", field1.set("123")(baseline), /* result of A */)
val testcase2 = Testcase("B", field1.set(/* не "123", а, наприклад, */"123" + "1">)(baseline), /*result of B*/)
```

У випадку, якщо один з тестових сценаріїв неможливо створити, то можна вважати, що виявлений мертвий код, і умова разом з відповідною віткою можна спокійно вилучити.

Якщо в декількох розгалуженнях перевіряються незалежні властивості об'єкта, то можна досить просто сформувати вичерпний набір змінених тестових об'єктів, який повністю покриває всі можливі комбінації.

DSL для формування всіх комбінацій змін

Автоматичне формування тестових даних

Тому що ми маємо справу з білим ящиком, то можемо бачити всі розгалуження. Це дає можливість побудови моделі логіки, що втримується в білому ящику, і використання моделі для генерації тестових даних. У випадку, якщо тестуємий код написаний на Scala, можна, наприклад, використовувати scalameta для читання коду, з наступному перетворенням у модель логіки. Знову

ж, як і в розглянутому раніше питанні моделювання логіки змін, для нас скрутне моделювання всіх можливостей універсальної мови. Далі будемо припускати, що тестуємий код реалізований з використанням обмеженого підмножини мови, або на іншій мові або DSL, який споконвічно обмежений. Це дозволяє зосередитися на тих аспектах мови, які представляють для нас інтерес.

Розглянемо приклад коду, що містить єдине розгалуження:

```
if(object.field1 == "123") A else B
```

Умова розбиває множину значень поля field1 на два класи еквівалентності: == "123" і != "123". Тим самим, уся множину вхідних даних також розбивається на два класи еквівалентності стосовно цієї умови – Classcondition1Istrue і Classcondition1Isfalse. З погляду повноти покриття нам досить побрати хоча б по одному прикладу із цих двох класів, щоб покрити обидві галузі A і B. Для першого класу ми можемо побудувати приклад, у якімсь змісті, єдиним образом: побрати випадковий об'єкт, але змінити поле field1 на "123". При цьому об'єкт обов'язково виявиться в класі еквівалентності Classcondition1Istrue і обчислення підуть по галузі A. Для другого класу прикладів більше. Один зі способів генерації якогось прикладу другого класу: генерувати довільні вхідні об'єкти й відкидати такі, у яких field1 == "123". Ще один спосіб: побрати випадковий об'єкт, але змінити поле field1 на "123" + "*" (для модифікації можна використовувати будь-яка зміну контрольного рядка, що гарантує, що новий рядок не рівний контрольної).

У якості генераторів випадкових даних цілком підходить механізм генераторів Arbitrary і Gen з бібліотеки Scalacheck.

По-суті, ми виконуємо звертання булевой функції, використовуваної в операторові if. Тобто знаходимо всі значення вхідного об'єкта, для яких ця булева функція ухвалює значення true – Classcondition1Istrue, і всі значення вхідного об'єкта, для яких вона ухвалює значення false – Classcondition1Isfalse.

Подібним чином можна генерувати дані, що підходять під обмеження, породжувані простими умовними операторами з константами (більше/менше константи, входить у множину, починається з константи). Такі умови неважко

звернути. Навіть якщо в тестуємому коді викликаються нескладні функції, то ми можемо замінити їхній виклик на їхнє визначення (inline) і все-таки здійснити звертання умовних виражень.

Важко оборотні функції

Інакше обстоїть справа в тому випадку, коли в умові використовується функція, яку важко звернути. Наприклад, якщо використовується хеш-функція, то автоматично генерувати приклад, що дає необхідне значення хеш-коду, очевидно, не вийде.

У такому випадку можна додати у вхідний об'єкт додатковий параметр, що представляє результат обчислення функції, замінити виклик функції на звертання до цього параметра, і обновляти цей параметр, незважаючи на порушення функціонального зв'язку:

```
if(sha(object.field1)=="a9403...") ...
// після введення додаткового параметра ==>
if(object.sha_field1 == "a9403...") ...
```

Додатковий параметр дозволяє забезпечити виконання коду усередині вітки, але, мабуть, може привести до фактично некоректних результатів. Тобто тестуєма програма буде видавати результати, які ніколи не можуть спостерігатися в реальності. Проте, перевірки частини коду, яка інакше нам недоступна, однаково корисна й може розглядатися як різновид модульного тестування. Адже й при модульному тестуванні підфункція викликається з такими аргументами, які, можливо, ніколи не будуть використовуватися в програмі.

При таких маніпуляціях ми заміняємо (підмінюємо) об'єкт тестування. Проте, у якімсь змісті нова побудована програма включає логіку колишньої програми. Дійсно, якщо в якості значень нових штучних параметрів брати результати обчислення функцій, які ми замінили на параметри, то програма видасть ті ж самі результати. Очевидно, тестування зміненої програми як і раніше може становити інтерес. Треба лише пам'ятати, при яких умовах змінена програма буде поводитися також, як вихідна.

Залежні умови

Якщо розгалуження в коді опираються на незалежні поля об'єкта, то завдання повного покриття вирішується прямолінійно. Досить записати всі підмножини для кожного тестуємого поля, а потім, використовуючи підходящі генератори, згенерувати нові значення для всіх полів. Якщо ж наступні умови уточнюють множину значень поля, то необхідно брати перетинання підмножин. (Наприклад, перша умови, x > 0, а наступне – x <= 1. Кожне з умов могло б дати по дві підмножини, але за рахунок перетинань у підсумку вийде тільки три підмножини – (-∞, 0], (0, 1], (1, +∞), – приклади з яких треба буде згенерувати.)

Якщо при формуванні уточнюючих множин ми виявимо, що одне з підмножин порожньо, то це означає, що умова завжди буде ухвалювати фіксоване значення true або false незалежно від вхідних значень. Тому відповідна вітка, яка ніколи не викликається, є "мертвим кодом" і може бути вилучена з коду разом з умовою.

Зв'язані параметри

Розглянемо випадок, коли умови розгалуження заснована на двох полях об'єкта, також зв'язаних умовами:

```
if(x > 0)
  if(y > 0)
   if (y > x)
```

Тут кожне з полів обмежене > 0, і обоє поля спільно теж обмежені – y > x.

Якщо умова є "вільним", як, у цьому прикладі, то досить згенерувати приклади значень полів, перевірити виконання всіх умов, і відкинути невідповідні значення властивостей. Тому що область припустимих значень досить велика в порівнянні з областю невідповідних значень, то у великій кількості випадків ми будемо генерувати підходящі комбінації властивостей і "коефіцієнт корисної дії" такого методу буде досить більшим.

У випадку, якщо умови "тверде", функціональне (y == x + 1), те можна побудувати функцію, що обчислює значення другого поля на основі сгенерованного першого.

Якщо умова "напівжорстке" (y > x + 1 && y < x + 2), то можна побудувати залежний генератор, діапазон значень якого визначається сгенерованним значенням першого поля.

Символьне виконання

Для того, щоб зібрати всі умови, що породжують результат по який-небудь із галузей, можна скористатися "символьним виконанням" (Symbolic Execution, Символьне виконання програм), суть якого полягає в наступному. Вхідні дані ухвалюються рівними деяким символьним значенням (field1 = field1_initial_value). Потім над символьними значеннями проводяться всі маніпуляції, описані в тестуємом коді. Усі маніпуляції виконуються в символьному ж виді:

```
val a = field1 + 10
// додаємо в контекст a = field_initial_value + 10
val b = a * 3
// додаємо в контекст b = 3 * field_initial_value + 30
```

Для обробки розгалуження ухвалюються можливими обоє варіанта умови – true і false. І розрахунки проводиться відразу для двох віток. При цьому на символьні значення в кожній вітці накладаються протилежні обмеження. Наприклад,

```
if(a > 0) A else B
// у вітці A ухвалюємо, що field_initial_value + 10  > 0
// у вітці B ухвалюємо, що field_initial_value + 10 <= 0
```

Накопичені в символьному виді обмежень можна використовувати або для формування генератора, що породжує значення, що задовольняють цим обмеженням, або для перевірки випадкових значень, формованих менш точним генератором. У кожному разі з'являється можливості генерувати випадкові дані, що приводять до виконання заздалегідь відомого шляху (і, можливо, до відомого результату).

Тестування циклів і рекурсивних функцій

Дотепер ми обходили питання циклів стороною. Зв'язане це з тим, що в циклі міняється стан, тобто цикл обов'язково використовує змінювану змінну. Ми

ж позначили границі нашого розгляду чистими функціями, що має на увазі використання тільки незмінних структур даних. Також при наявності циклів існує ризик формування таких умов, при яких результат не буде отриманий за розумний час.

Відомо, що будь-який цикл можна замінити рекурсією. Це може бути непросто для складних циклів. Але, допустимо, що в нашому випадку така операція була зроблена. Тим самим, у тестуємом коді будуть зустрічатися рекурсивні виклики, а ми можемо продовжувати наші міркування, зберігаючи вихідне припущення про розгляд тільки чистих функцій. Як ми могли б протестувати такий білий ящик, враховуючи той факт, що рекурсивні виклики, так само, як і цикли, можуть не завершитися за розумний час?

Скористаємося такою конструкцією як Y-комбінатор ("комбінатор нерухливої точки", stackoverflow:What is a Y-combinator? (2-ой відповідь), habr: Одержання Y-комбінатора в 7 простих кроків). Комбінатор дозволяє реалізувати рекурсію в мовах, які рекурсію в чистому виді не підтримують. (Сам комбінатор є рекурсивним, тому повинен бути реалізований мовою, що підтримує рекурсію.) Працює він у такий спосіб. З рекурсивної функції віддаляються всі рекурсивні виклики й заміняються на виклики функції, яка передається в якості додаткового аргументу. Така перероблена функція вже не буде рекурсивної, а служить тільки "заготовкою" для одержання цільової функції. Y-комбінатор перетворює таку "заготовку рекурсивної функції" у повноцінну рекурсивну функцію (передаючи в якості аргументу власне продовження).

Розглянемо спочатку важливий окремий випадок хвостової рекурсії (Хвостова рекурсія). Перепишемо тестуємий код, замінивши рекурсивні виклики на виклики допоміжної функції. Для цілей тестування ми передамо власну реалізацію допоміжної функції, яка не буде формувати рекурсію. Така допоміжна функція може формувати результат, що вертається, відповідний до типу значень, що вертаються, білого ящика. Наприклад, якщо вертаються рядки, те допоміжна функція також буде формувати рядок, який ми зможемо перевірити в рамках

TestCase'a. У випадку, якщо тип значень, що вертаються, нам виявиться незручний, ми можемо викидати виключення (throw має тип Nothing або bottom, що є підтипом усіх інших). Тим самим ми одержимо можливість тестування коду без циклів і ризику зависання.

У випадку загальної рекурсії рекурсивний виклик повертає результат, який потім використовується. У цьому випадку вищенаведений підхід прямо не працює. Можна спробувати застосувати підхід, аналогічний тому, що ми використовували для викликів важко оборотних функцій. А саме, замінимо кожний рекурсивний виклик на новий параметр. Значення цих параметрів можна буде генерувати як звичайно, виходячи з умов розгалужень, у яких ці параметри використовуються. Як і у випадку із заміною викликів функцій на параметри, результати, які ми будемо одержувати, можуть відрізнятися від результатів, які ми можемо одержати в дійсності. Збіг буде досягатися в тому випадку, якщо значення параметра збігається зі значенням рекурсивної функції. Такий підхід дозволяє нам протестувати кроки, виконувані після рекурсивного викликів.

Зміст тестування білого ящика

При певній ретельності можна добитися того, що тести, написані вручну або сгенерованні автоматично, будуть покривати всі галузей тестуємого коду, тобто забезпечать 100% покриття. Тим самим ми зможемо із упевненістю сказати, що білий ящик робить те, що він робить.

У деяких випадках такий набір тестів все-таки може мати сенс:

1. Якщо білий ящик є прототипом і існує ненульова ймовірність помилки для дійсної реалізації в іншому середовищі.

2. Якщо існує кілька реалізацій однакової логіки (на різних мовах або в різних оточеннях).

3. Код білого ящика еволюціонує або зазнає рефакторингу, при цьому ми прагли б виявляти відмінності від еталонної поведінки.

4. Ми прагнемо виявити підмножини вхідних даних, що забезпечують необхідні нам результати.

Слід мати у виді деякі особливості тестування, заснованого на реалізації, на відміну від тестування на основі специфікації. По-перше, якщо споконвічна реалізація не підтримувала деяку функціональність, яку можна було б очікувати, ґрунтуючись на специфікації, те наші тести не помітять її відсутності. По-друге, якщо така функціональність була присутня, але працювала інакше, чим зазначене в специфікації (тобто, з помилками), те наші тести не просто цих помилок не виявлять, а напроти, помилки будуть "кодифіковані" у тестах. І якщо наступні/альтернативні реалізації спробують виправити помилки, те такі тести не дозволять цього просто так зробити.

Тестування білого ящика зміщає акцент із питання "що повинен робити код" на "що фактично робить код". Іншими словами, замість використання більш високого рівнів абстракції, формування тестів на основі специфікації, використовується точно той же рівень абстракції, що й при реалізації коду. Ми можемо одержати гарні результати в плані покриття коду, але при цьому таке тестування має сенс в обмеженому наборі випадків.

Якщо ви зіштовхнулися з таким випадком, у якому тестування білого ящика виправдане, то міркування, наведені вище, можуть придатися. По-перше, основні зусилля має сенс зосередити на формуванні тестових наборів даних, тому що вхід у білого ящика один (виклик функції), а протестувати хотілося б усі галузей. По-друге, очевидно, має сенс побудувати модель тестуємого коду. Для цього може використовуватися спеціалізований DSL, досить виразний, щоб предвідносити тестуєму логіку. По-третє, користуючись моделлю тестуємої логіки можна спробувати автоматично сформувати тестові дані галузей, що покривають усі. По-четверте, тестуємий код може бути піддадуть автоматичним перетворенням, які роблять його більш зручним для тестування (виключення викликів тяжкообратимих функцій, перехід від циклів до рекурсії, виключення рекурсивних викликів). При використанні цих підходів можна одержати гарні результати в плані покриття коду.

Таким чином, у сприятливих умовах і при реалізації деяких з вищенаведених підходів, з'являється можливість автоматичної генерації змістовних тестів. Можливо, зацікавлені читачі запропонують і інші області, де могло б застосовуватися тестування білого ящика або які-небудь із розглянутих підходів.

Огляд Solar inCode 1.2

Solar inCode – програмний продукт для статичного аналізу коду. У ньому зручно сполучається функціонал аналізу коду по вимогах фахівців в області інформаційної безпеки й розроблювачів самих застосунків. Продукт дозволяє виконувати аналіз коду як по вихідних текстах програм, так і по бінарному коду, що виконується. У матеріалі перераховані технічні можливості продукту, представлена його архітектура й сценарії роботи.

У цей час у кожній компанії, яка оперує чутливою інформацією й віддаленно надає доступ до своїх даних користувачів через веб-сервіси й мобільні застосунки, необхідно використовувати захищені програмні системи.

Сьогодні програмні системи повинні не тільки працювати без перебоїв, але й не містити помилок, скориставшись якими зловмисник міг би одержати несанкціонований доступ до конфіденційних даних.

Щоб оцінити, наскільки ПЗ надійно захищає дані, з якими працює, широко застосовуються статичний аналіз програми й динамічний аналіз програмного застосунку. Статичний аналіз працює з текстом програми – це перевірка методом «білого ящика». Динамічний аналіз – це тестування системи в процесі роботи, інакше кажучи, перевірка методом «чорного ящика».

При виконанні динамічного аналізу фахівець імітує діяльність зловмисника, намагається застосувати шаблони поведінки, здатні вивести програму із запланованого сценарію виконання, щоб одержати несанкціонований доступ до керування програмою. Статичний аналіз звичайно не вимагає запуску програми на виконання, тому що фахівець працює з текстом програми. Статичний аналіз більш повний і якісний, чому динамічний, тому що робота виконується з текстом

програми й усі помилки видні. При динамічному аналізі експерт (як і зловмисник) по суті вгадує, які помилки міг зробити програміст, і намагається ними скористатися.

Інструментальний засіб статичного аналізу коду Solar inCode пропонує комплексну перевірки програмного забезпечення, у першу чергу орієнтованого на надання дистанційних послуг користувачам: перевірка від серверної частини до мобільних і веб-застосунків, включаючи можливість перевірки застосунків без надання вихідних кодів. Але в цілому inCode може ефективно перевіряти будь-які застосунки на мовах, які він підтримує.

Solar inCode інтерпретує результати не тільки для розроблювачів, але й для фахівців інформаційної безпеки. Крім цього, inCode дозволяє цілеспрямовано працювати із програмним забезпеченням дистанційного доступу.

Продукт орієнтований на наступні типи користувачів:

- Розроблювач.
- Фахівець інформаційної безпеки.
- Керівник.

Для кожного типу користувачів пропонується свій набір опцій, можливостей, генеруються різні звіти. Інтерфейс системи повністю локалізований.

Схема роботи Solar inCode

Solar InCode доступний у двох варіантах: як сервіс і як самостійний продукт із коробки. Solar InCode як коробковий продукт повністю вбудовується в життєвий цикл розробки ПЗ від завантаження даних для сканування з репозиторія до вивантаження дефектів, виявлених при скануванні, прямо в систему керування проектом.

Solar InCode Online – це сервіс по статичному аналізі коду, який надається віддаленно. Дані для сканування завантажуються через веб-інтерфейс, результати сканування доступні у вигляді докладного звіту.

Solar inCode працює повністю автоматично. Можна виділити два типи статичного аналізу, який виконує Solar inCode – аналіз тексту програми й аналіз, що виконується коду програми.

Зручно завантажувати дані для аналізу: зокрема, мобільні застосунки завантажуються прямо з Google Play і Apple Store, і все це робиться однією кнопкою. Робочі файли онлайн-застосунків завантажуються у два кличі функцією upload. Аналогічно вихідні коди можна завантажувати як локально, так і прямо з репозиторія розробки, а дефекти відразу відсилати в системи керування проектами.

Важливо відзначити, що Solar inCode – єдиний в Україні інструментальний засіб статичного аналізу коду, яке пропонує аналіз мобільних і веб-застосунків без вихідних кодів.

При роботі з вихідними текстами програми аналіз починається зі складання застосунку, далі програма трансформується в особливу внутрішню структуру вистави програми – абстрактне синтаксичне дерево. Після чого виконується пошук уразливостей: використовується спочатку набір методів пошуку по шаблонах, а потім taint-аналіз. Виявлені уразливості позначаються в тексті програми, до кожної уразливості видаються рекомендації.

При аналізі застосунків для Android спочатку виконується розплутуванні byte-коду, яке виконано спеціально для захисту від відновлення його в текст програми. Розплутаний byte-код транслюється у внутрішню виставу. Після цього паралельно виконується відновлення з byte-коду в java-текст і сам статичний аналіз розшифрованого byte-коду по внутрішньому представленню. Після того як відновлення завершене й статичний аналіз виконаний, запускається процес відображення виявлених уразливостей на відновлений java-текст.

При аналізі мобільних застосунків для платформи iOS спочатку виконується розшифрування коду, що виконується, потім уже розшифрований застосунок зазнає паралельно двом процесам: статичному аналізу й декомпіляції. Статичний аналіз виконується на рівні LLVM, у який розшифрований код

застосунку транслюється як у внутрішню виставу. Декомпіляція виконується з LLVM у високорівневе представлення Objective-C-подібної програми.

Представлення результатів сканування залежить від того, хто працює із системою: тобто результати аналізу представляються по-різному для розроблювачів і для фахівців інформаційної безпеки.

Системні вимоги Solar inCode

Типова конфігурація програмно-апаратного комплексу Solar inCode для роботи статичного аналізатора коду – це один скануючий сервер, який підключений до загального репозиторію, і багато робочих станцій, підключених до скануючему серверу. З робочої станції фахівець може налаштовувати сценарій виконання сканувань, управляти скануванням і одержувати результати.

Для стабільного функціонування робочої станції необхідна машина, де вона буде встановлен, що відповідає наступним системним вимогам: 2 ядра, 4Gb RAM, 100 GB диск.

Для стабільної роботи самого ядра статичного аналізу необхідне встаткування наступної конфігурації: 2-4 Gb RAM, 0,5 Gb диск. На такому сервері можна впевнено сканувати приблизно кілька мільйонів рядків коду на одному ядрі.

Для зберігання робочих результатів сканування, включаючи історію пересканування, досить файлового сховища 0,5 Gb для роботи з декількома мільйонами рядків коду.

Виходячи з вищеописаних вимог, були перелічені вимоги для системи з навантаженням 40 аналізів у день по 10 на кожному ядру, які виконуються паралельно. Для піврічної експлуатації системи без видалення історії сканування в середньому буде досить:

- 12 ядер;
- 32 Gb RAM;
- 4 Tb диск.

Мінімальна конфігурація системи для виконання самого аналізу й керування їм – наприклад, проста робоча станція для офіцера інформаційної

безпеки, який у середньому виконує 1-2 аналізу паралельно й зберігає для роботи історію приблизно 50-150 сканувань:

- 4 ядра;
- 8 Gb RAM;
- 500Gb диск.

У цей час інструментальний засіб аналізу коду Solar inCode працює як під керуванням ОС Windows (7/8/8.1), так і під керуванням Mac OS і ОС Linux.

Сканування мобільних застосунків iOS і аналіз застосунків, реалізованих мовою програмування Objective-C, вимагає Mac OS. Якщо клієнтові незручно розвертати в себе Mac OS, обробка застосунків може відбуватися на сервері Solar Security.

Основні функціональні можливості Solar inCode

- Аналіз текстів програм по вимогах інформаційної безпеки з наданням результатів сканування:

a) для розроблювачів;

b) для фахівців інформаційної безпеки;

c) для керівників.

- Аналіз програмних застосунків по бінарних кодах, у тому числі мобільних застосунків для платформ Android і iOS із завантаженням безпосередньо з Google Play і Apple Store.
- Відображення результатів сканування завжди виконується на текст програми.
- Гнучкий інтерфейс роботи із системою, що налаштовується для роботи різних фахівців.
- Складання звітів налаштовується для прочитання різними фахівцями.

Як ми вже відзначали вище, роботу статичного аналізатора коду можна розділити на статичний аналіз до текстів програм і сканування коду, що виконується, без тексту програми.

Статичний аналіз по текстах програм

Тут можна виділити два напрямки використання Solar inCode:

- для підвищення якості розробки ПЗ;
- для підвищення якості приймання ПЗ.

Якщо статичний аналізатор коду Solar inCode використовується для підвищення якості розробки, то головний користувач – це команда розроблювачів.

У цьому випадку Solar inCode вбудовується в процес автоматичного складання, що дозволяє виконувати сканування на регулярній основі по сценаріях – наприклад, під час нічних складань.

Інтерфейс роботи з додатком налаштовується для розроблювача, а інтерпретація результатів сканування видається з обліком того, хто їх буде читати. Розроблювач має великий досвід створення програмного коду, але не розташовано витрачати багато часу на читання довгих описів можливих ризиків, які несе дана уразливість для бізнесу. Тому при видачі рекомендацій для розроблювачів акцент зроблений на опис походження уразливості, простежування всього шляхи життя уразливості по вихідному коду – від зародження до точки експлуатації. Також розроблювачеві для ознайомлення приводиться сценарій можливого вектора атаки на застосунок з використанням виявленої уразливості. При бажанні можна настроїти показ посилань на опис виявленої уразливості на офіційних інформаційних ресурсах інтернету.

Можливий сценарій роботи, коли сканування виконує тільки лідер проекту. Це відбувається одночасно з виконанням перевірки коду (code review) своєї команди. У цьому випадку лідер групи може делегувати усунення виявлених дефектів саме тем членам команди, які розробляли відповідний код. Тому що Solar inCode підтримує автоматичну інтеграцію із системами керування проектами, то вивантаження дефектів виконується автоматично.

У випадку експлуатації Solar InCode для підвищення якості приймання ПЗ основним користувачем системи є офіцер інформаційної безпеки. На відміну від розроблювача, офіцерові ІБ звичайно не потрібно зберігати багато історій

сканування одного застосунку, тому що він виконує сканування готової версії при прийманні. Тому він може виконувати сканування як на загальному з розроблювачами сервері сканування, так і на локальній станції. Вихідні коди так само, як і для розроблювачів, можна автоматично забирати з репозиторія.

Результати сканування також відображаються на вихідний код, однак інтерпретація результатів трохи інша, ніж для розроблювачів. По-перше, офіцерові ІБ важливо розуміти, які ризики несе виявлений дефект для стабільної роботи системи, чи можна даний дефект закрити засобами захисту периметра експлуатації, яка трудомісткість виправлення виявленого дефекту і яка ймовірність його експлуатації. У випадку якщо дефект важко експлуатувати й можна закрити засобами захисту периметра, а також він несе малий ризик витоку чутливих даних, офіцер ІБ швидше за все прийме розв'язок відкласти виправлення такого дефекту. Тому звіт, який формується для офіцера ІБ, у першу чергу дає відповіді на подібні питання.

Для деяких дефектів Solar inCode видає рекомендація з налаштування захисту периметра експлуатації.

Тому що є інтеграція із системами керування проектами, то вивантаження виявлених у процесі приймання дефектів може бути виконана офіцером ІБ автоматично.

Статичний аналіз без вихідних кодів

Якщо вихідний код застосунку відсутній, то його перевіркою займається служба інформаційної безпеки. Вхідні дані для аналізу можна завантажити в застосунок або скачати по посиланню з інтернету. Мобільні застосунки для аналізу можна прямо завантажувати по посиланню з Google Play або Apple Store відповідно.

За результатами сканування видається розгорнутий звіт, який містить усю необхідну для фахівця ІБ інформацію про знайдені дефекти, який ризик вони несуть і до якої інформації потенційно можна одержати доступ у випадку успішної атаки на застосунок. Також звіт містить сценарії можливих атак.

Щоб підтвердити наявність дефектів зовнішнім розроблювачам, а також для локалізації уразливостей – результат сканування відображається на відновлений текст програми.

У випадку мобільних застосунків для платформи Android уразливості відображаються на java-код, у випадку мобільних застосунків для платформи iOS уразливості відображаються на Objective-C-код.

Статичний аналіз коду для керівника

Відмітною функціональною особливістю Solar inCode є генерація звітів спеціально для керівників.

У керівника звичайно мало часу, але він здатний сприймати інформацію швидко, тому у звіт треба збирати тільки необхідну інформацію для ухвалення рішення. Також звіти, які генеруються для керівників, відрізняє те, що інформація подається не в абсолютних, а у відносних величинах. Крім цього, розроблювачі подбали про наочність і лаконічності вистави.

Особливості Solar inCode

Автоматизований аналіз робочих файлів

Solar inCode дозволяє аналізувати мобільні застосунки для Android і iOS, а також застосунку, написані мовою Java і Scala, по робочих файлах, тобто без вихідного тексту програми (файли з розширенням jar, war, apk, ipa і інші). Аналіз цілком проходить в автоматичному режимі – від завантаження робочого файлу до відображення результатів аналізу. Для мобільних застосунків в inCode реалізована можливість запуску аналізу по посиланню на сторінку застосунку в магазині, через який застосунок поширюється: Google Play Market або App Store.

Відображення знайдених уразливостей на вихідний код

При аналізі робочих файлів (без вихідного коду) в inCode знайдені уразливості транслюються в інтерфейсі системи на відновлений вихідний код застосунку в точності до номера рядка. Це реалізоване за допомогою технології відновлення інформації, що зв'язує бінарні й вихідні коди в процесі аналізу робочого файлу.

Облік специфіки мобільних платформ

При аналізі мобільних застосунків в inCode застосовується розширена база правил по виявленню уразливостей, специфічних для мобільних платформ. При складанні розширеної бази правил використовувалися результати ручного аналізу великої кількості мобільних застосунків, до яких пред'являються підвищені вимоги інформаційної безпеки.

Аналіз застосунків мовою Scala

Solar inCode дозволяє аналізувати застосунки, написані мовою Scala. При аналізі Scala-застосунків використовуються не тільки правила, застосовувані при аналізі Java-застосунків, але й правила, за допомогою яких відбувається пошук уразливостей, специфічних для мови Scala. Для розробки правил було проведене дослідження основних фреймворків і бібліотек, що використовуються в Scala-Додатках, на предмет можливості їх небезпечного використання.

Пошук коду, підозрілого на НДМ

Крім стандартних уразливостей, inCode дозволяє шукати ділянки коду, які є підозрілими на наявність у них недокументованих можливостей (закладок). У результаті дослідження, проведеного експертною групою, була складена база правил, по якій відбувається пошук закладок, – перевірка спеціальних облікових записів, «тимчасових бомб» і так далі.

Рекомендації з налаштування СЗІ

За результатами аналізу веб-застосунків inCode формує рекомендації з налаштування засобів захисту інформації (наприклад, Web Application Firewall) для запобігання можливості експлуатації знайдених уразливостей. Рекомендації являють собою докладні інструкції зі скріншотами по налаштуванню основних використовуваних СЗІ.

Fuzzy Logic Engine

Найважливішим параметром статичного аналізатора коду є кількість неправильних спрацьовувань. Для їхньої мінімізації в inCode використовується

математичний апарат нечіткої логіки (fuzzy logic). Модуль Fuzzy Logic Engine дозволяє відкидати уразливості, які були невірно визначені в результаті аналізу.

Мови програмування, аналізовані Solar inCode

У цей час Solar inCode на високому рівні підтримує сканування наступних мов програмування:

- Java.
- Scala.
- PHP.
- Objective-C.
- Java for Android.

Найближчим часом будуть додані:

- C#.
- PL/SQL.
- Java Script.

В 2016 році планується підключити підтримку наступних мов програмування:

- Visual Basic.
- 1C.
- Swift.
- Perl.
- Python.
- C/C++.

Робота з Solar inCode

Для початку роботи з Solar inCode необхідно зареєструвати нового користувача в системі. Це може зробити адміністратор, обліковий запис для якого створюється автоматично при установці системи.

Якщо користувач зареєстрований у системі, то він повинен виконати авторизацію для початку роботи.

Для створення нового проекту треба вибрати, який проект передбачається сканувати, а далі виконати завантаження вихідного коду для аналізу. Завантажити застосунок для сканування можна:

- с локального пристрою;
- з Google Play;
- з Apple Store;
- з репозиторія.

На рисунку 2.1 показаний інтерфейс завантаження застосунку для сканування.

Сканування може займати якийсь час – воно потрібно для побудови внутрішньої вистави й виконання самого аналізу. Звичайне пересканування займає менше часу, чому первісна перевірка, тому що внутрішня представлення не будується заново для кожного сканування, а з метою оптимізації добудовується. На рисунку 2.1 представлений інтерфейс відображення результатів сканування у вихідний код, а також показане вікно з описом виявленої уразливості.

Рисунок 2.1 – Відображення результатів сканування у вихідний код

Кругова діаграма наочно демонструє кількість уразливостей по типу, які виявлені в проекті. Також видається оцінка рівня безпеки проекту, яка будується по декільком показникам, узятим з певними вагами. Показники, які використовуються для оцінки безпеки: концентрація уразливостей, оцінка рівня критичності кожної уразливості, трудомісткість усунення виявлених дефектів, кількість входжень кожної уразливості й інші.

Крім лаконічного звіту, який доступний у системі, можна зробити вивантаження результатів сканування у форматі pdf. Інтерфейс дозволяє вибрати інформацію, яка буде відображена у звіті.

З моменту виявлення уразливостей до моменту їх виправлення звичайно проходить багато часу, протягом якого немає можливості зняти застосунок з експлуатації або відкласти впровадження. У цьому випадку для деяких дефектів можна виконати налаштування СЗІ, щоб закрити можливість експлуатації цих уразливостей до виходу наступної версії застосунку, де дані уразливості будуть виправлені.

Ринок програмного забезпечення для статичного аналізу коду поповнився гідним продуктом – Solar inCode.

З однієї сторони цей продукт є практичним інструментом для розроблювачів, який зручно вбудовується в життєвий цикл програмного забезпечення, а з іншого – являє собою ефективний засіб у допомогу службі інформаційної безпеки.

Особливо слід зазначити, що Solar inCode представляє функціональність, яка поки нова для ринку – аналіз застосунків без вихідних кодів.

Solar inCode вигідно відрізняється від конкурентних розв'язків тим, що надає опис виявлених дефектів і рекомендацій з їхнього усунення російською мовою. Крім цього, відмінною рисою є можливість налаштування генерації звітів. Істотна позитивна сторона Solar inCode – це відмінність інтерфейсу системи для різних груп користувачів: розроблювач, офіцер інформаційної безпеки й керівник.

Також треба відзначити, що Solar InCode продається як ПЗ, яке можна розгорнути в себе на майданчику й повністю інтегрувати у свій процес розробки. Для виконання разового сканування програмного коду можна скористатися сервісом Solar InCode Online по моделі Saas. При цьому всі сервера, на яких розгорнуть Solar InCode Online, розташовані строго на території Україні.

Продукт, що розвертається на майданчику, ліцензується по кількості користувачів. Solar inCode Online – по кількості сканувань.

У якості недоліків слід зазначити відносно невелика кількість підтримуваних мов програмування й місцями недопрацьований дизайн інтерфейсу.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Embarcadero Delphi, раніше Borland Delphi і Codegear Delphi, – інтегроване середовище розробки ПЗ для Microsoft Windows, Mac OS, iOS і Android мовою Delphi (що раніше носила назву Object Pascal), створена спочатку фірмою Borland і на даний момент приналежна й розроблювальна Embarcadero Technologies. Embarcadero Delphi є частиною пакета Embarcadero RAD Studio і поставляється в чотирьох редакціях: Community (поширюється безкоштовно й має обмежену ліцензію на використання в комерційних цілях), Professional, Enterprise і Architect.

Delphi 10.4 Sydney

Випущено 26 травня 2020 року. RAD Studio Delphi 10.4 забезпечує значно поліпшену високопродуктивну нативну підтримку Windows, кращу продуктивність розробки, миттєві підказки code completion, прискорення виконання коду із синтаксисом керованих записів, поліпшення виконання паралельних завдань на сучасних багатоядерних CPU, а також містить більш 1000 виправлень багів, поліпшення продуктивності середовища й бібліотек і багато чого крім того.

Основні можливості Delphi 10.4.1:

− Істотні розширення для Windows: поліпшення для застосунків на моніторах 4K High DPI, інтеграція з новим WebView2 на базі Chromium, використання розширених title bars, таких же, як в Office, Explorer, Google Chrome.

− Керування пам'яттю в Delphi тепер стандартизоване на всіх підтримуваних платформах – мобільних, настільних і серверних – використовучи класичну реалізацію керування пам'яттю об'єктів.

− Істотне поліпшення Delphi Code Insight (без можливого блокування IDE – в окремому процесі), що допоможе при роботі з великими проектами.

− Тип даних Delphi «record» тепер підтримуть довільні ініціалізацію, фіналізацію й операції копіювання.

− Розширена підтримка бібліотек C++: ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode.

− Відладник Win 64 (на LLDB) і збирач для C++.

− Поліпшення для C++: Включена велика кількість поліпшень STL з Dinkumware.

− Підтримка Metal Driver GPU для macOS і iOS.

− Вбудований Fmxlinux.

− Компонент Twebbrowser для iOS тепер реалізований на Wkwebview API. Реалізація компонента Media Player для macOS тепер використовує Avfoundation. Реалізований заново стилізуємий FMX компонент TMemo на платформі Windows значно поліпшений і тепер має відмінну підтримку IME.

− Численні поліпшення швидкості й стабільності роботи нашої бібліотеки The Parallel Programming Library (PPL).

− Додані оновлені драйвери для FireBird, PostgreSQL і SQLite.

− Клієнтські бібліотеки HTTP і REST Client розширені застосунковими можливостями роботи з HTTPS. Також були розширені можливості підтримки Amazon AWS services

– У технологію Visual LiveBindings внесена множину поліпшень, у тому числі швидкодії, що стосуться, застосунків на VCL і FireMonkey

RAD Studio 10.4 Короткий огляд:

– Істотні розширення для Windows. Створення застосунків, що чудово виглядають, із чіткими елементами інтерфейсу на 4k моніторах High DPI за допомогою нової гнучкої підтримки стилів елементів керування на екрані. Інтеграція із сучасними, безпечними web-технологіями від Microsoft – новим WebView2 на базі Chromium. Використання сучасних розширених title bars, таких же, як в Office, Explorer, Google Chrome, у своїх проектах. Істотні поліпшення надійності налагодження в новому відладнику для C++ Windows 64-bit.

– Зросла продуктивність розробки. Ріст продуктивності за рахунок миттєвої реакції підказок code completion у середовищі IDE. Краща сумісність із уже наявною кодовою базою, і спрощення програмування за рахунок уніфікованої архітектури керування пам'яттю. Швидке зв'язування даних і візуальних елементів за допомогою розширеної технології Visual LiveBindings з підвищеною швидкодією. Просте використання розповсюджених бібліотек C++, наприклад, ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode. Оновлена підтримка Amazon AWS cloud.

– Поліпшення швидкодії і якості. Більш 1000 поліпшень швидкодії і якості. Краща ефективність коду за допомогою нового синтаксису custom managed records. Більш швидке виконання паралельних завдань на сучасних багатоядерних CPU. Переконаєтеся в прискоренні відображення на екрані з підтримкою Metal API на macOS і iOS. Краща сумісність із уже наявною кодовою базою й спрощення програмування за рахунок уніфікованої архітектури керування пам'яттю.

Істотне поліпшення Delphi Code Insight

Як найбільше й головне поліпшення інструментів програмування Delphi за багато років, в 10.4 Delphi Code Insight реалізований через Language Server Protocol (LSP). LSP – це технологія генерації результатів для code completion,

навігації й інших сервісів в окремому процесі. Це значить, що code completion і Code Insight одержать більш точні результати без блокування IDE. 10.4 забезпечує набагато більш високу продуктивність розроблювачів, які працюють із більшими проектами, що містять мільйони рядків коду.

Delphi Custom Managed Records

Ключове розширення мови Delphi: тип даних Delphi «record» тепер підтримуть довільні ініціалізацію, фіналізацію й операції копіювання. Управляйте тем, як ці структури створюються, копіюються й звільняються з допомогу вашого коду, який буде виконуватися у відповідний момент.

Це розширює потужність конструкцій records в Delphi, які використовуться щоб одержати більшу ефективність у порівнянні із класами.

Єдине керування пам'яттю

Керування пам'яттю в Delphi тепер стандартизоване на всіх підтримуваних платформах – мобільних, настільних і серверних – використовучи класичну реалізацію керування пам'яттю об'єктів.

У порівнянні з Automatic Reference Counting (ARC), це дає кращу сумісність із існучим кодом і спрощує написання компонентів, бібліотек і застосунків.

ARC модель керування пам'яттю model залишилася для керування рядками й посиланнями на тип інтерфейсу на всіх платформах. Для C++ це означає, що при створенні й звільненні Delphi-style класів в C++ використовується звичайне керування пам'яттю, як у будь-якого heap-allocated класу C++, що значно знижує складність коду.

Розширена підтримка бібліотек C++

В 10.4 ми портували багато популярних бібліотек C++ у C++Builder.

Забезпечивши оптимізовану підтримку бібліотек ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode, поряд із уже підтримуваними Boost і Eigen, які можуть бути додані за допомогою менеджера пакетів Getit.

Win 64-відладник і збирач для C++

В 10.4 з'явився новий відладник C++ для Windows 64-bit. Відладник заснований на LLDB і показує значне збільшення стабільності при налагодженні 64-bit застосунків поряд з новими відладочними можливостями, такими як перегляд і інспекція типів начебто рядків C++ і Delphi, а також колекцій STL, включаючи std::vector, std::map і інших. Крім того, згенерована для застосунку відладочна інформація має інший внутрішній формат, сприяючи більш стабільному й багатому на можливості процесу налагодження, більш докладним перегляду й інспекції в debug-time.

Підвищення якості й швидкодії інструментів

- Велика кількість поліпшень STL від Dinkumware.
- Поліпшені деякі найважливіші методи й області RTL, на базі поліпшень сумісності з популярними бібліотеками C++.
- Поліпшена підтримка Cmake.
- Велика кількість виправлень для підвищення стабільності і якості.
- Відновлення Windows API – Обновлено й додали множину декларацій API щоб добитися ще більшої інтеграції із платформою Windows.
- Загальні вдосконалення в бібліотеці доступу до БД FireDAC, включаючи оновлені драйвера для FireBird, PostgreSQL і SQLite. Вибір статичного або динамічного підключення SQLite до застосунку.

Змінені стилі VCL для High DPI

В 10.4, архітектура стилізації VCL була суттєво розширена для підтримки High DPI і 4K моніторів. Тепер усі елементи UI на формі VCL автоматично масштабуться під відповідне до монітора дозвіл для показу форми. Був оновлений API стилізації для підтримки стилів high DPI.

Кожний графічний елемент UI може бути обраний з наборів різних масштабів і масштабований до потрібного DPI, що дає чітке зображення елементів UI на всіх моніторах.

Нові High DPI стилі й стилізація окремих VCL компонент

Обновлено велике число вбудованих і преміальних VCL стилів для підтримки нового режиму стилізації High-dpi. Це дозволяє вам створювати застосунку з відмінним дизайном для всіх моніторів.

Розроблювачі VCL застосунків тепер можуть використовувати трохи VCL стилів на різних формах в одному застосунку або в різних компонентів на одній формі. Це також включає стилізацію компонентів загальною темою для платформи. Крім застосункової гнучкості використання стилів, це дозволяє використовувати нестилізуємі компоненти із зовнішніх бібліотек в VCL застосунках, що використовуть стиль.

Поліпшена кроссплатформеність

- Додана підтримка Metal Driver GPU для macOS і iOS.
- Крім підтримки останнього iOS SDK, в RAD Studio 10.4 розроблювачі можуть задовольнити нові вимоги Apple до набору стартових екранів.
- Реалізований заново стилізуємий FMX компонент TMemo на платформі Windows значно поліпшений і тепер має відмінну підтримку IME.
- Користувачам редакцій Enterprise або Architect доступна повна інтеграція Fmxlinux з IDE для створення клієнтських застосунків Linux з GUI.
- Компонент Twebbrowser для iOS тепер реалізований на Wkwebview API.
- Реалізація компонента Media Player для macOS тепер використовує Avfoundation.

Оновлений менеджер пакетів Getit

Менеджер пакетів Getit в IDE був значно вдосконалений.

Дати випуску релізів пакетів тепер видні, і можливе сортування списку по цих датах; відбір тільки встановлених пакетів, контенту, доступного тільки при наявності підписки, багато чого іншого.

Універсальний інсталятор для установки Online і Offline

В 10.4 включений новий універсальний інсталятор, який використовує технологію на базі Getit. Цей інсталятор підтримує як online, так і offline (з ISO) варіанти установки.

Тепер обоє варіанта установки дозволяють вам указати початковий набір можливостей RAD Studio для установки, наприклад, свою комбінацію мов програмування й цільових платформ, мов інтерфейсу, і додавати до нього або видаляти непотрібне в будь-який момент.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на кваліфікаційну бакалаврську роботу, реалізації підлягає програмне забезпечення, яке призначено для системи кібербезпеки для аудиту IT інфраструктури методом «Білий ящик».

В процесі розробки кваліфікаційної бакалаврської роботи 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 контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи кібербезпеки, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу

успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Аналіз захищеності, тести на проникнення, ASV-сканування – усе це нестаріючі теми. Пропозицій на ринку вистачає, і кожний виконавець пропонує свою методику: простої сканування на наявність уразливостей або тести на проникнення, дослідження «чорним», «сірим» або «білим ящиком», DDoS-атаки, фішингові розсилання, «дорожнє яблуко» і множину інших опцій на вибір. Очевидно, що не всі методики однаково корисні й вибір повинен залежати від переслідуваної мети. Давайте ж спробуємо розібратися, як найбільш оптимальним образом сформулювати завдання по аналізі захищеності.

Тестування «**чорним ящиком**» – це тестування з нульовим знанням про об'єкт. На противагу йому тестування «**білим ящиком**» – тестування з повним знанням про об'єкт. У проектах по аналізі захищеності під «білим ящиком» найчастіше розуміється дослідження на рівні вихідних кодів застосунку.

Ціль – оцінити стійкість організації до хакерських атак

Це, мабуть, сама комплексна мета, яку можна визначити для проекту по аналізі захищеності. Вона декомпозується на множину складових. Наприклад, оцінка захищеності мережного периметра, оцінка можливостей зовнішнього або внутрішнього порушника, оцінка можливої шкоди. Іноді в якості вторинної мети на таких проектах фігурує оцінка ефективності роботи служби ІБ, коли хочеться на практиці перевірити, як співробітники служби будуть виявляти атаки й протидіяти їм.

Метод тестування слід вибирати виходячи з можливостей порушника, від якого хочеться захиститися. Якщо передбачається, що проти організації буде діяти «випадковий» порушник, який мало що знає про об'єкт атаки – це тестування «чорним ящиком». Якщо передбачається, що порушником може бути

відвідувач офісу, варто задуматися на тему аналізу захищеності Wifi-Мережі. Якщо можливі атаки зсередини організації, то й аналіз також необхідно виконувати від імені внутрішнього порушника.

Найпоширеніша модель порушника – це зовнішній порушник, досить компетентний у питаннях комп'ютерних технологій, але, що не володіє спецзасобами для здійснення атак на професійному рівні. У цьому випадку дослідникові безпеки досить повідомити лише бренд або назва організації. Усе інше він повинен з'ясувати сам. Крім основної мети дослідження такий підхід покаже, наскільки легко можна зібрати інформацію про організацію і якого роду буде ця інформація. Наприклад, наскільки точно і як швидко порушник зможе окреслити мережний периметр. Які сервіси й служби виявляться й наскільки вони захищені. Яку інформацію можна одержати з опису доменної зони, з розвідувачів, із соціальних мереж, використовуючи спеціальні приймання збору інформації.

Важливо не виключати з області дослідження істотні частини інфраструктури. Обмежуючи завдання по аналізі захищеності, можна легко пропустити уразливості, якими реальний порушник неодмінно скористається. У нашої практиці була множину робіт, коли в локальну мережу вдавалося проникнути через уразливість у якому-небудь службовому сервісі, який уже довгий час не використовується й залишився опублікованим лише по недогляду.

Ціль – протестувати певний продукт або сервіс перед релізом

У лікарів є гарна приказка: «Не буває абсолютно здоровіших людей, бувають недообстежені». Так і з уразливостями. Ніколи не можна бути повністю певен у їхній відсутності. Рано або пізно уразливості виявляються майже в кожному застосунку. Усе питання в тому, скільки зусиль потрібно прикласти для пошуку чергової уразливості.

При постановці завдання тестування застосунку на предмет наявності уразливостей слід відштовхуватися від тієї цінності, яку представляють застосунок, що й утримуються в ньому дані для потенційного порушника. В

ідеалі застосунок повинний бути протестоване настільки, щоб пошук чергової уразливості був недоцільний. Якщо точніше, вигода, яку можна одержати, зламавши застосунок, повинна бути значно менше вартості злому.

У загальному випадку метод тестування повинен також вибиратися виходячи із критичності застосунку. Іноді досить автоматизованого тестування. Більш ефективним, але й більш витратним є дослідження на рівні вихідних кодів. Для критичних застосунків рекомендується вбудовувати процедури аналізу безпеки в цикл розробки застосунку. Загальне правило таке: чим більше вихідних даних буде в дослідника безпеки, тим більше уразливостей буде знайдено.

Ціль – compliance, відповідність вимогам

Тести на проникнення й ASV-сканування включені в багато стандартів по інформаційній безпеці як обов'язкові процедури. Наприклад, відповідність стандарту PCI DSS припускає проведення ASV-сканування не рідше, чим раз у квартал. Стандарт ISO/IEC 27001 містить у собі дисципліну по керуванню уразливостями.

Оскільки такі тести виконуються, що називається, « для галочки», часто завдання формулюється в мінімальному обсязі. Без надмірностей. Головне, щоб отримані результати були застосовні з метою compliance.

Такий підхід виправданий тільки в тому випадку, якщо організація сама на винному рівні займається аналізом захищеності своєї обчислювальної інфраструктури. Якщо ж об'єктивної інформації про рівень захищеності ні, то має сенс трохи розширити стандартне compliance-тестування, поставивши перед підрядником ціль не стільки скласти звіт, скільки провести практичну роботу з дослідження захищеності. І на цьому варто загострити увага підрядника, оскільки найчастіше самі вони до робіт з compliance відносяться досить формально.

3.2 Розробка структурної схеми

Тестування на проникнення

Аналіз захищеності допомагає відповісти на важливі питання:

1. Який поточний рівень захищеності?

Наскільки захищений периметр мережі? Що може почати зовнішній або внутрішній порушник? Яка шкода може бути нанесений критичним інформаційним системам і даним? Як будуть розвиватися атаки?

2. Яка реакція на таргетовану (цілеспрямовану) атаку?

Наскільки оперативно і як ефективно виявляються комп'ютерні атаки? Чи здатні співробітники служб IT та ІБ протистояти хакерам Працюють чи в дійсності заходу й засобу виявлення й попередження атак?

3. Що почати для посилення безпеки?

Чому діючі заходи щодо забезпечення безпеки не ефективні? Есть Есть ли системні проблеми и как с ними бороться На що слід звернути увагу в першу чергу, що почати в довгостроковій перспективі?

Вибирайте глибину й масштаб

Структура роботи визначається вашими цілями й об'єктом аналізу.

Залежно від цілей аналіз захищеності може виконуватися з різним охватом і глибиною. Акцент може бути зміщений убік практичної безпеки або убік оцінки відповідності певним вимогам. Аналіз може бути поверхневим або глибоким. Робота може вестися з імітацією дій зовнішнього або внутрішнього порушника.

Зовнішнє й внутрішнє тестування

Автоматизований і ручний пошук уразливостей, їх аналіз і експлуатація.

Тестування бездротових мереж

Оцінка безпеки, формування карти покриття бездротової мережі, пошуки несанкціонованих точок доступу.

Соціальна інженерія

Оцінка поінформованості співробітників компанії в області інформаційної безпеки.

Аналіз веб-застосунків

Аналіз на наявність уразливостей, викликаних помилками програмування, слабостями конфігурації, помилками бізнес-логіки й архітектури застосунків. Проводиться на основі посібника з тестування OWASP.

Аналіз мобільних застосунків

Оцінка уразливостей на рівні застосунків, проблем, пов'язаних з викликами API, компонентів клієнтських мобільних застосунків. Включає OWASP Mobile Security Testing Guide (MSTG).

Red Teaming

Повномасштабне багаторівневе моделювання атак. Містить у собі серії тестів, пов'язаних з тестуванням на проникнення й тестуванням безпеки застосунків.

Основні результати роботи

Якісно й інформативно – незалежно від типу дослідження.

Перелік уразливостей

За результатами випробувань ми надаємо перелік виявлених уразливостей, ранжируваний по критичності й об'єктам аналізу, а також рекомендації з усунення уразливостей.

Рекомендації

Поряд з рекомендаціями з усунення окремих уразливостей, ми даємо рекомендації більш загального характеру, дотримання яких спрямовано на усунення системних проблем безпеки.

Звітна презентація

При необхідності може бути проведена презентація за підсумками проекту. Проведення презентації дозволяє продемонструвати значимість і пріоритизувати питання захисту інформації.

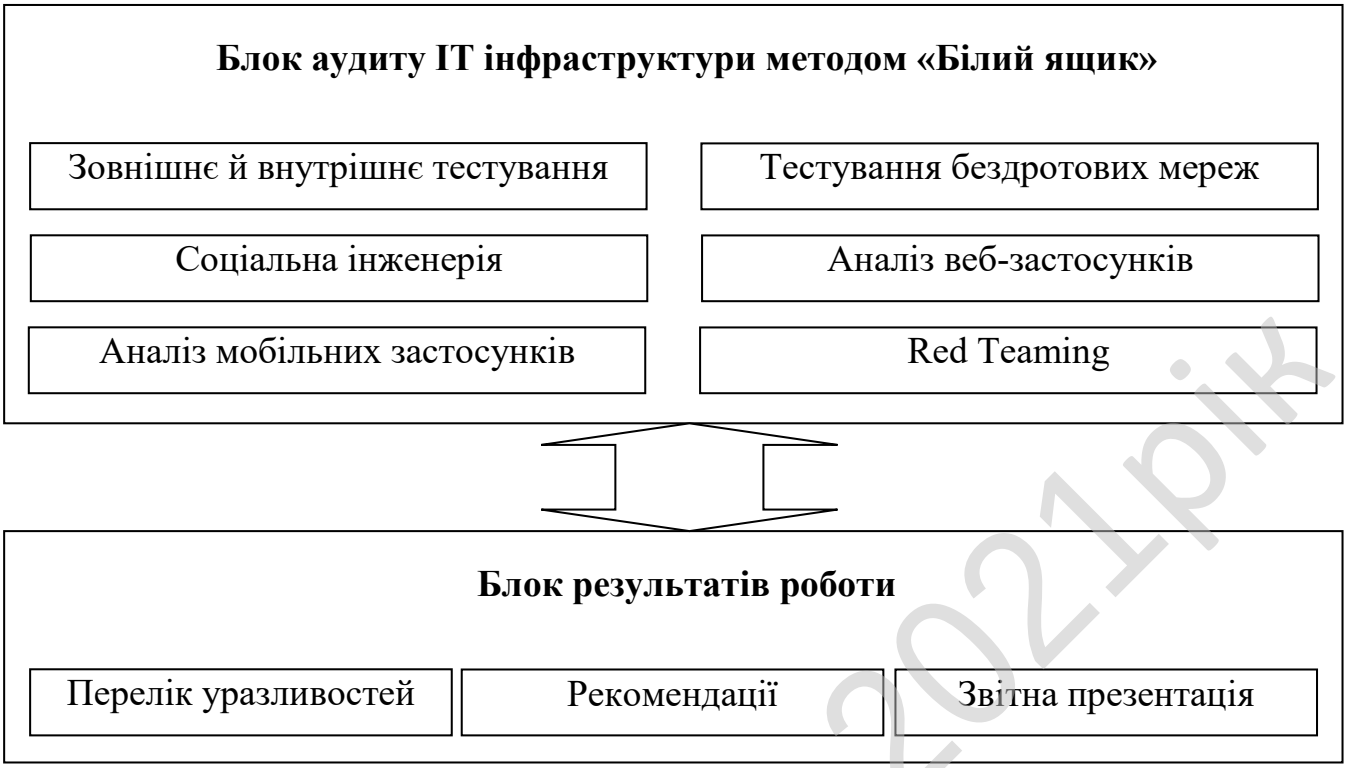

Рисунок 3.1 – Структурна схема системи

3.3 Розробка функціональної схеми

Як відомо, існує прямий зв'язок між розміром компанії з однієї сторони й ступенем нерозуміння як же функціонують IT сервіси, а головне як вони взаємодіють між собою з іншої сторони.

Організація постійно росте, бізнес вимагає все частіше нові IT інструменти й сервіси. Одні з них використовуються всією компанією, інші – одним або двома співробітниками, а про якісь тестові інструменти зовсім забувають, незважаючи на те, що вони як і раніше є живими частинами інфраструктури й вимагають уваги.

Як розплутати цей клубок взаємозв'язків?

На допомогу приходить програмне забезпечення, яке розроблене в даній роботі – WhiteBoxMethod.

WhiteBoxMethod являє собою потужний аналізатор трафіку.

Основний плюс розробленого розв'язку в тому, що крім «залізної» реалізації (окремі сервер колектор і аналізатор), він уміє працювати усередині віртуальної інфраструктури. Розвертається у вигляді appliance на продукти компанії Vmware без яких або спеціальних навичок і читання довгих мануалів.

Після установки, досить настроїти Port Mirroring на Vmware dvswitch, що робиться буквально в кілька кличів, і ми одержуємо копію всього трафіку з нашої віртуальної інфраструктури на колектор.

Далі досить зайти на WhiteBoxMethod і вказати адреса колектора Infinistream. Через 10 хвилин ми вже буде мати аналіз усього трафіку в легко читабельному й зрозумілому виді.

Хотілося б відразу розставити всі точки над «і». WhiteBoxMethod не є моніторингом серверів. Він є аналізатором взаємодії сервісів інфраструктури. Працює він на підставі даних, які максимально можна «вичавити» із трафіку. І повірте, це дуже багато. Ви навіть не представляєте, скільки віток взаємозв'язку побудує WhiteBoxMethod навколо кожного вашого сервісу.

Основні можливості для аудита

Функції WhiteBoxMethod представлені нижче:

1. Уміє сам визначати сервіси залежно від типу трафіку.

2. Покаже, які застосунки найчастіше використовуються в компанії, яку частку трафіку мають у мережі.

3. Будує візуальну карту взаємодії.

4. Виводить на dashboard процентне співвідношення позитивної роботи сервісів і операцій з помилками.

5. Уміє трактувати практично все коди помилок застосунків.

6. Дозволяє відразу, саме в точці, де відбулася помилка, постфактум зробити розбір пакета в читабельному виді.

7. Надає покроковий розбір транзакцій.

8. Допомагає швидко побачити нелегітимні з'єднання.

Цей список можна продовжувати нескінченно, якщо починати перераховувати можливості, якими користуються системні адміністратори. Нам в основному вистачає цих 8 пунктів, щоб зрозуміти загальну й, якщо буде потрібно, детальну «життя» інфраструктури.

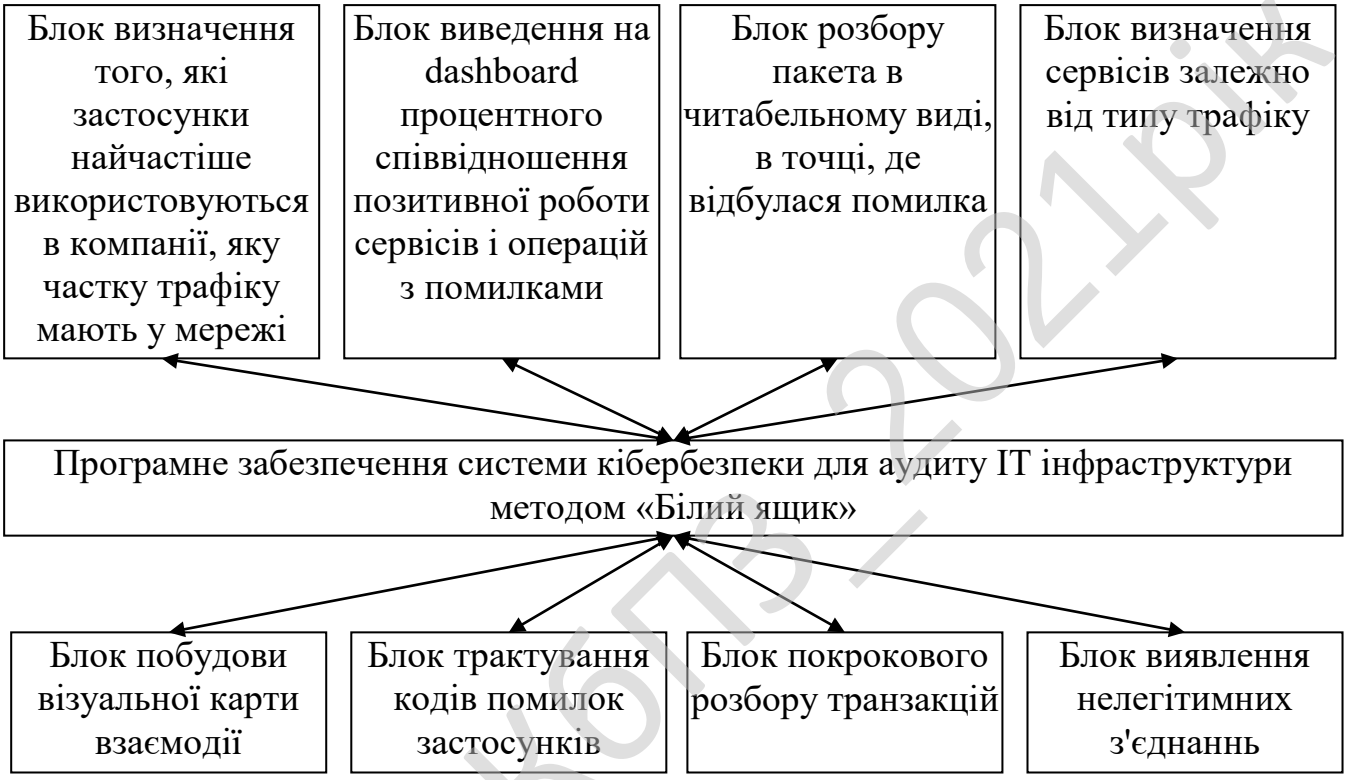

Рисунок 3.2 – Функціональна схема системи

3.4 Розробка діаграми процесів

Відповідно до методичних рекомендацій розроблення графічної частини кваліфікаційної бакалаврської роботи розглянемо розроблену діаграму процесів яка зображена на рисунку 3.3.

Розроблена діаграма взаємодії процесів використовується для представлення та візуалізації процесів обробки даних тобто структурного проектування бакалаврської роботи.

Основні складові елементи діаграми взаємодії процесів це потоки даних:

– Репозиторії, потік сховища даних.

– Потоки зовнішні по відношенню до системи сутності.

– Процеси які являють собою трансформацію даних в рамках описуваної системи.

– Потоки даних гібридні між елементами трьох попередніх типів.

Відповідно до документації основна будова діаграми процесів полягає у графічному представленні складу сукупностей даних, що характеризуються як співвідношення різних частин кожної з сукупностей. Склад статистичної сукупності графічно може бути представлений як за допомогою абсолютних, так і відносних показників. Графічне зображення складу сукупності по абсолютними і відносними показниками сприяє проведенню більш глибокого аналізу і дозволяє проводити аналіз системи.

Рисунок 3.3 – Діаграма взаємодії процесів

Для схематичного представлення системи що розробляється необхідно спочатку представити діаграму взаємодії процесів даних рівня контексту, завдяки чому буде показано взаємодію системи в цілому у подальшому. Використовується

модель проектування, графічне представлення «потоків» даних в інформаційній системі.

Розроблена діаграма взаємодії процесів системи в подальшому уточнюється шляхом деталізації процесів та потоків даних з метою показати систему що розробляється.

Таким чином у результаті після розгляду, вищеописаної системи, схеми структурної, функціональної, діаграми взаємодії процесів перейдемо до опису та розгляду блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

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

4.1 Блок-схеми та опис алгоритмів функціонування системи

Розглянемо алгоритм роботи основної програми. Його блок-схема зображена на рисунку 4.1. З рисунку видно, що після запуску програми спочатку відбувається ініціалізація динамічних бібліотек та перевірка наявності файлу існуючих типів атак. Потім здійснюється запит перевірки файлу та при наявності проходе підключення файлу атак.

Далі проходе послідовно запит перевірки всі модулі підключені системи чи ні та при наявності проводиться сканування та складання списку пристроїв в мережі з реалізацією черги чекання запиту перевірки аномального трафіку з послідуючим запитом прав адміністратора при реалізації якої відбувається виклик підпрограми моніторингу IT-інфраструктури що зображено на рисунку 4.2.

Під час роботи блоку аудиту крім основного функціоналу увагу необхідно також зосередити на оперативну пам’ять так як при її повному критичному застосуванні неминуче викривлення у остаточної звітної презентації. Розглянемо вихідний код:

```
unit ModuleNeedMemory; // модуль

interface

uses // бібліотеки
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, ExtCtrls, Gauges, StdCtrls, ShellAPI, Menus;

type
  TfrmMain = class(TForm) // клас
    TmrRefresh: TTimer;
    Panel9: TPanel;
```

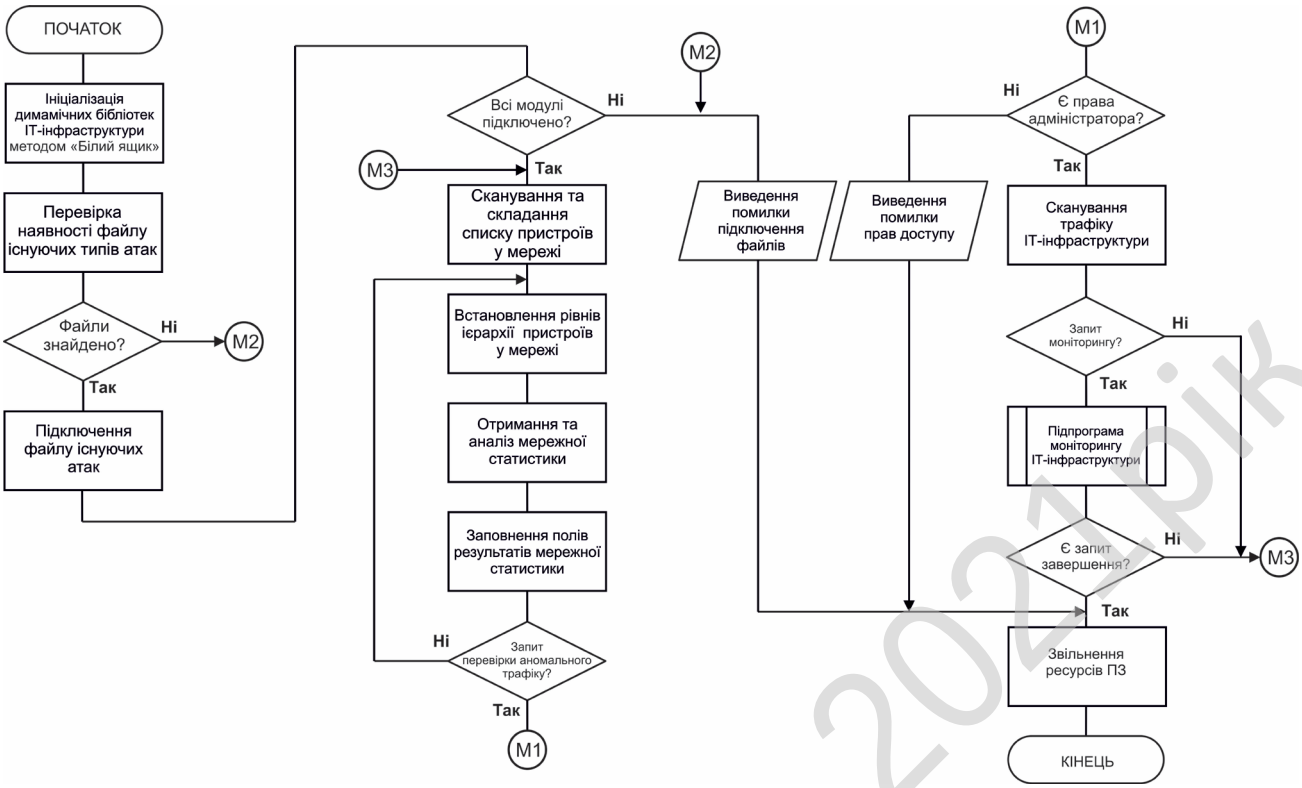

Рисунок 4.1 – Блок-схема основної програми

```
Panel10: TPanel;
Panel1: TPanel;
GgAllocatedMem: TGauge;
Panel6: TPanel;
GgAllocatedPhysicalMem: TGauge;
Panel7: TPanel;
GgAllocatedPageFileMem: TGauge;
Panel8: TPanel;
GgAllocatedVirtualMem: TGauge;
Panel2: TPanel;
GgFreeMem: TGauge;
Panel3: TPanel;
GgFreePhysicalMem: TGauge;
Panel4: TPanel;
GgFreePageFileMem: TGauge;
Panel5: TPanel;
GgFreeVirtualMem: TGauge;
Label1: TLabel;
Label2: TLabel;
Panel11: TPanel;
```

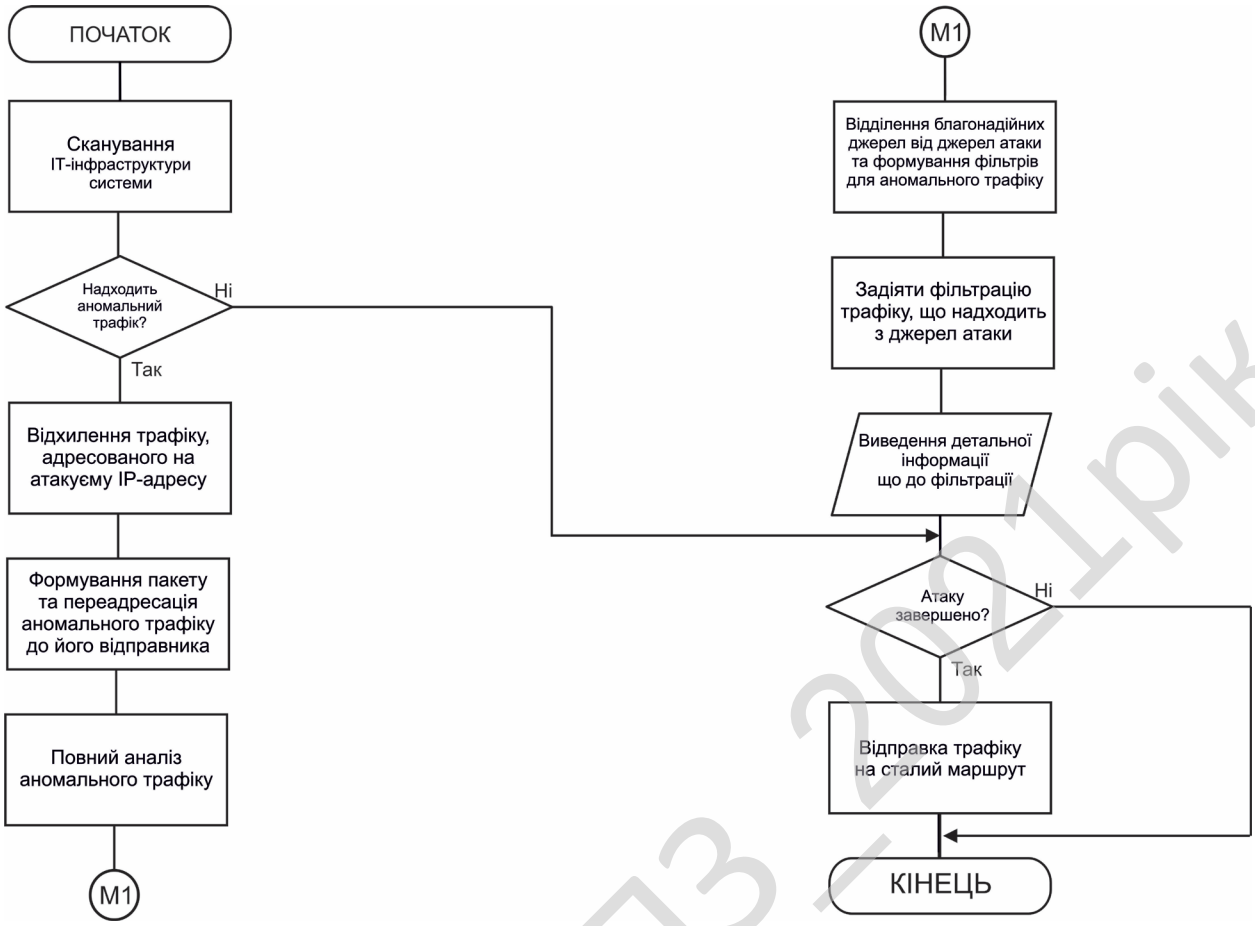

Рисунок 4.2 – Блок-схема роботи підпрограми

```
    MmResumo: TMemo;
    PppMnTray: TPopupMenu;
    Fechar1: TMenuItem;
    procedure TmrRefreshTimer(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure Fechar1Click(Sender: TObject);
  private
    DeixaSair: Boolean;
    IconData: TNotifyIconData;
    procedure MyMinimize(Sender: TObject);
    function UpRound(Value: Double): Integer;
    function DownRound(Value: Double): Integer;
  end;

var
  frmMain: TfrmMain; // об'єкт класу
```

```
implementation // реалізація об'яв

{$R *.dfm}

function TfrmMain.DownRound(Value: Double): Integer;
begin
  Result := Trunc(Value);
end;

procedure TfrmMain.TmrRefreshTimer(Sender: TObject);
const
  cBytesPorMb = 1024 * 1024;
var
  MemStatus: TMemoryStatus;
begin
  MemStatus.dwLength := SizeOf(MemStatus);
  GlobalMemoryStatus(MemStatus);

  MmResumo.Clear;
  with MmResumo.Lines do
   begin
Add(Format('Memory in use:                    %d%%',
           [MemStatus.dwMemoryLoad]));
Add('Total of physical memory:     ' + FormatFloat('0000.00 Mb',
          MemStatus.dwTotalPhys / cBytesPorMb));
Add('Maximum of pagefile:          ' + FormatFloat('0000.00 Mb',
          MemStatus.dwTotalPageFile / cBytesPorMb));
    Add('Total of virtual memory:    ' + FormatFloat('0000.00 Mb',
          MemStatus.dwTotalVirtual / cBytesPorMb));
   end;

  // Memorias alocadas
  with GgAllocatedMem do
  begin
    MaxValue := DownRound(MemStatus.dwTotalPhys / cBytesPorMb);
    Progress := UpRound(MemStatus.dwMemoryLoad);
  end;
  with GgAllocatedPhysicalMem do
  begin
    MaxValue := DownRound(MemStatus.dwTotalPhys / cBytesPorMb);
    Progress := UpRound((MemStatus.dwTotalPhys - MemStatus.dwAvailPhys) /
```

```
                cBytesPorMb);
  end;
with GgAllocatedPageFileMem do
begin
MaxValue := DownRound(MemStatus.dwTotalPageFile / cBytesPorMb);
Progress := UpRound((MemStatus.dwTotalPageFile -
            MemStatus.dwAvailPageFile) / cBytesPorMb);
  end;
  with GgAllocatedVirtualMem do
  begin
MaxValue := DownRound(MemStatus.dwTotalVirtual / cBytesPorMb);
Progress := UpRound((MemStatus.dwTotalVirtual -
            MemStatus.dwAvailVirtual) / cBytesPorMb);
  end;
with GgFreeMem do
  begin
    MaxValue := DownRound(MemStatus.dwTotalPhys / cBytesPorMb);
    Progress := DownRound(MemStatus.dwTotalPhys / cBytesPorMb) -
                          DownRound(MemStatus.dwMemoryLoad);
  end;
  with GgFreePhysicalMem do
  begin
    MaxValue := DownRound(MemStatus.dwTotalPhys / cBytesPorMb);
    Progress := DownRound(MemStatus.dwAvailPhys / cBytesPorMb);
  end;
  with GgFreePageFileMem do
  begin
    MaxValue := DownRound(MemStatus.dwTotalPageFile / cBytesPorMb);
    Progress := DownRound(MemStatus.dwAvailPageFile / cBytesPorMb);
  end;
  with GgFreeVirtualMem do
  begin
    MaxValue := DownRound(MemStatus.dwTotalVirtual / cBytesPorMb);
    Progress := DownRound(MemStatus.dwAvailVirtual / cBytesPorMb);
  end;
end;

function TfrmMain.UpRound(Value: Double): Integer;
begin
  Result := Trunc(Value);
  if Frac(Value) <> 0 then
    Result := Trunc(Value + 1);
```

```
end;

// конструктор
procedure TfrmMain.FormCreate(Sender: TObject);
begin
  TmrRefreshTimer(Self);
end;

// деконструктор
procedure TfrmMain.FormDestroy(Sender: TObject);
begin
  Shell_NotifyIcon(NIM_DELETE, @IconData);
end;
// приховання форми
procedure TfrmMain.MyMinimize(Sender: TObject);
begin
  ShowWindow(Application.Handle, SW_HIDE);
end;

procedure TfrmMain.Fechar1Click(Sender: TObject);
begin
  Close;
end;
end.
```

Під час роботи над програмою було створено блок-схеми. Перед їх розглядом необхідно провести роз’яснення який саме тип блок-схем використовується.

Блок-схема це представлення задачі для її аналізу або розв'язування за допомогою спеціальних символів (геометричних образів), які позначають такі елементи, як операції, потік, дані тощо. Блок вхідних та вихідних даних прийнято позначати паралелограмом, блок обчислень (обробки) даних – прямокутником, блок прийняття рішень – ромбом, еліпсом – початок та кінець алгоритму.

У інформаційних технологіях функціональна схема складається з функціональних блоків, які являють собою конструктивно відособлені частини (елементи або пристрої) автоматичних систем, які виконують певні функції. Функціональні блоки на схемі позначають прямокутниками, всередині яких надписують їх найменування відповідно до функцій, що виконуються. Зв'язки між

функціональними блоками (внутрішні впливи) позначаються лініями зі стрілками, які вказують напрям впливів.

Функціональні схеми можуть виконуватися в укрупненому і розгорненому вигляді. У першому випадку на схемі зображають найважливіші блоки системи і зв'язки між ними.

У другому варіанті схема відображається більш детально, що полегшує її читання та ілюструє принцип роботи.

Основні елементи схем алгоритму це термінатор, процес, рішення, зумовлений процес (підпрограма), дані та з'єднувач.

Термінатор це елемент відображає вхід із зовнішнього середовища або вихід з неї (найчастіше застосування – початок і кінець програми). Всередині фігури записується відповідна дія.

Процес це виконання однієї або кількох операцій, обробка даних будь-якого виду (зміна значення даних, форми подання, розташування). Всередині фігури записують безпосередньо самі операції.

Рішення це показує рішення або функцію перемикального типу з одним входом і двома або більше альтернативними виходами, з яких тільки один може бути обраний після обчислення умов, визначених всередині цього елементу. Вхід в елемент позначається лінією, що входить зазвичай у верхню вершину елементу. Якщо виходів два чи три то зазвичай кожен вихід позначаєт ься лінією, що виходить з решти вершин (бічних і нижній). Якщо виходів більше трьох, то їх слід показувати однією лінією, що виходить з вершини (частіше нижній) елемента, яка потім розгалужується. Відповідні результати обчислень можуть записуватися поруч з лініями, що відображають ці шляхи.

Зумовлений процес (підпрограма) це символ відображає виконання процесу, що складається з однієї або кількох операцій, що визначені в іншому місці програми (у підпрограмі, модулі). Всередині символу записується назва процесу і передані в нього дані.

Дані це перетворення у форму, придатну для обробки (введення) або відображення результатів обробки (виведення). Цей символ не визначає носія даних (для вказівки типу носія даних використовуються специфічні символи).

З'єднувач це символ відображає вихід в частину схеми і вхід з іншої частини цієї схеми. Використовується для обриву лінії та продовження її в іншому місці (приклад: поділ блок-схеми, що не поміщається на листі). Відповідні сполучні символи повинні мати одне (при тому унікальне) позначення.

Jira – була використана комерційна система відслідковування помилок, призначена для організації взаємодії з користувачами, хоча в деяких випадках використовується і для управління проектами. Розроблено компанією Atlassian, є одним з двох її основних продуктів (поряд з вікі-системою Confluence). Має веб-інтерфейс.

Назва системи отримано шляхом усічення слова «Gojira» – Японського імені монстра Годзилла, що, в свою чергу, є відсиланням до назви конкуруючого продукту – Bugzilla; створювалася в якості заміни Bugzilla і багато в чому повторює її архітектуру. Система дозволяє працювати з декількома проектами. Для кожного з проектів створює і веде схеми безпеки і схеми оповіщення.

До версії 3.13.5 (включно) розрізнялися редакції Enterprise, Professional і Standard, після – Залишилася тільки редакція Enterprise (для великих організацій).

Система заснована на Java EE і працює на кількох популярних системах управління базами даних і операційних системах.

Основний елемент обліку в системі – завдання (ticket або issue). Завдання містить назву проекту, тему, тип, пріоритет, компоненти і зміст. Завдання може бути розширена додатковими полями (також і нові призначені для користувача поля можуть бути визначені), додатками (наприклад – фотографіями, скриншотами) або коментарями. Завдання може редагуватися або просто змінювати статус, наприклад, з «відкритий» в «закритий». Які переходи між станами можливі, визначається через настроюється потік операцій. Будь-які зміни в задачі записуються в журнал.

Jira має велику кількість можливостей конфігурації: для кожної програми може бути визначений окремий тип завдання з власним workflow, набором статусів, одним або декількома видами уявлення (screens). Крім того, за допомогою так званих «схем» можна визначити для кожного індивідуального Jira-проекту власні права доступу, поведінку і видимість полів і багато іншого.

Завдяки універсальному підходу можна пристосувати Jira для багатьох непрофільних завдань, наприклад, керування вимогами, керування ризиками, аж до реалізації невеликої системи бронювання, автоматизації процесу рекрутингу.

Для інтеграції з зовнішніми системами підтримує інтерфейси SOAP, XML-RPC і REST. Поставляється із засобами інтеграції з такими системами управління версіями, як Subversion, CVS, Git, Clearcase, Team Foundation Server, Mercurial і Perforce.

Існують доповнення, що дозволяють вбудувати Jira в інтегровані середовища розробки, в тому числі Eclipse і IntelliJ IDEA. Перекладена багатьма мовами, включаючи російську, англійську, японську, німецьку, французьку, іспанську.

Для сторонніх розробників надаються кошти розробки розширень системи – плагінів. Розробники розширень можуть викладати плагіни для продажу на спеціальний розділ сайту Atlassian.

Є комерційним продуктом, який може бути ліцензований для роботи на локальному сервері або доступний в якості віддаленого додатки. Ціноутворення залежить від максимального числа користувачів, при цьому близько $50 за користувача для локального і $7 на місяць за користувача для віддаленого доступу є типовими цінами.

Для академічних і комерційних клієнтів доступний повний вихідний код під ліцензією розробника.

Для проектів з відкритим вихідним кодом Atlassian надає спеціальну безкоштовну ліцензію при дотриманні наступних правил:

– проект використовує ліцензії, схвалені Open Source Initiative;

– Вихідний код проекту доступний для скачування;

– у проекту є публічно доступна веб-сайт;

– програмне забезпечення від Atlassian є на веб-сайті проекту.

Була використана водоспадна (каскадна) модель життєвого циклу ПЗ (waterfall model) – послідовний метод розробки програмного забезпечення, названий так через діаграму схожу на водоспад.

Ця модель розробки запозичена з системної інженерії у виробництві та будівництві – областях, в яких зміни на пізніх етапах дуже дорогі, або неможливі. Наприклад, для створення складних інженерних конструкцій (споруд, літаків, мостів і т.п.). Зміни в проекті фундаменту будинку після того, як покладений дах коштують дуже дорого, тому перфекціонізм на початкових етапах проектування просто необхідний. Інженери, які починали займатись розробкою програмного забезпечення перейшовши з інших галузей, просто адаптували звичну модель, тому що на ранніх етапах розвитку комп'ютерної техніки не було методологій створених саме для програмування. Проте, схожі методології застосовуються для програмного забезпечення й далі, у випадках коли вимоги фіксовані, і вимагається висока якість та надійність, наприклад в системах для військових чи медичних потреб.

Перший формальний опис водоспадної моделі, після якої вона стала популярною був здійснений В. В. Ройсом у 1970. Попри те, що стаття містить переважно критику методу, на неї часто посилаються.

Переваги методу:

– Ніяких переробок.

– Гарна специфікація перетікає в гарну документацію.

– Зрозуміла модель.

– Розробники можуть мати низьку кваліфікацію.

Недоліки:

– Необхідний перфекціонізм на кожному етапі.

– Важко вносити зміни (якщо взагалі можливо).

– Надлишкове проектування.

– Поділ розробників на "perfect" та "code monkeys".

Модифікації. Через те що цей метод погано підходить для розробки саме ПЗ, частіше використовують його модифікації.

Найвідоміша модифікація – Sashimi. Названа так через японську страву сашимі (суші нарізане і сервіроване так, що складені рядочком шматочки накладаються один на одного). В моделі розробки Сашимі фази життєвого циклу йдуть одна за одною, але при цьому перекриваються одна з одною в часі.

Розглянемо формат що використовується – **JSON** (JavaScript Object Notation, укр. запис об'єктів JavaScript, вимовляється джейсон) – це текстовий формат обміну даними між комп'ютерами.

JSON базується на тексті, може бути прочитаним людиною. Формат дозволяє описувати об'єкти та інші структури даних. Цей формат головним чином використовується для передачі структурованої інформації через мережу (завдяки процесу, що називають серіалізацією). Розробив і популяризував формат Дуглас Крокфорд.

JSON знайшов своє головне призначення у написанні веб-програм, а саме при використанні технології AJAX. JSON, що використовується в AJAX, виступає як заміна XML (використовується в AJAX) під час асинхронної передачі структурованої інформації між клієнтом та сервером. При цьому перевагою JSON перед XML є те, що він дозволяє складні структури в атрибутах, займає менше місця і прямо інтерпретується за допомогою JavaScript в об'єкти.

JSON з'явився через необхідність обміну даними з сервером у реальному часі без використання плагінів для браузерів, flash-додатків або Java-аплетів, які використовувались скрізь на початку 2000-х років. Дуглас Крокфорд був тим, хто активно просував новий на той час формат. Він з колегами хотів створити технологію, яка використовувала б можливості звичайного браузера давала б веб-розробникам можливість створювати веб-додатки із постійним двостороннім зв'язком із веб-сервером.

JSON вперше був використаний в проекті в Communities.com для Cartoon Network, він дозволяв обмінюватись повідомленнями і одночасно маніпулювати DHTML-елементами.

Веб-сайт JSON.org було запущено 2002 року. У грудні 2005-го року Yahoo! почав переводити деякі зі своїх веб-сервісів на роботу з JSON. Google взялася до роботи з технологією у своєму веб-протоколі GData у грудні 2006-го року.

Використання

За рахунок своєї лаконічності в порівнянні з XML, формат JSON може бути більш придатним для серіалізації складних структур.

Якщо говорити про веб-застосунки, в такому ключі він доречний в задачах обміну даними як між браузером і сервером (AJAX), так і між самими серверами (програмні HTTP-інтерфейси).

Формат JSON так само добре підходить для зберігання складних динамічних структур в реляційних базах даних або файловому кеші.

Синтаксис

JSON будується на двох структурах:

1. Набір пар назва/значення. У різних мовах це реалізовано як об'єкт, запис, структура, словник, хеш-таблиця, список з ключем або асоціативним масивом.

2. Впорядкований список значень. У багатьох мовах це реалізовано як масив, вектор, список, або послідовність.

Це універсальні структури даних. Теоретично всі сучасні мови програмування підтримують їх у тій чи іншій формі. Оскільки JSON використовується для обміну даними між різними мовами програмування, то є сенс будувати його на цих структурах.

У JSON використовуються такі їхні форми:

1. Об'єкт – це послідовність пар назва/значення. Об'єкт починається з символу { і закінчується символом } . Кожне значення слідує за : і пари назва/значення відділяються комами.

2. Масив – це послідовність значень. Масив починається символом [і закінчується символом]. Значення відділяються комами.

3. Значення може бути рядком в подвійних лапках, або числом, або логічними true чи false, або null, або об'єктом, або масивом. Ці структури можуть бути вкладені одна в одну.

4. Рядок – це послідовність з нуля або більше символів юнікода, обмежена подвійними лапками, з використанням escape-послідовностей, що починаються зі зворотної косої риски \. Символи представляються простим рядком. Тип Рядок (String) дуже схожий на String в мовах C і Java. Число теж дуже схоже на C- або Java-число, за винятком того, що вісімкові та шістнадцяткові формати не використовуються. Пропуски можуть бути вставлені між будь-якими двома лексемами.

4.2 Захист розробленого програмного забезпечення

Для захисту розробленого програмного забезпечення запропоновано використовувати алгоритм Khufu. Khufu – це 64-бітовий блоковий шифр. 64-бітовий відкритий тест спочатку розщеплюється на дві 32-бітові половини, *L* і *R*.

Над обома половинами й певними частинами ключа виконується операція XOR. Потім, аналогічно DES, результати проходять деяку послідовність раундів. У кожному раунді молодший значущий байт *L* використовується як вхід S-блоку. У кожного S-блоку 8 вхідних біт і 32 вихідних біта.

Далі обраний в S -блоці 32-бітовий елемент піддається операції XOR з *R*. Потім *L* циклічно зрушується на число, кратним восьми биткам, *L* і *R* міняються місцями, і раунд завершується.

Сам S-блок не статичний, він міняється кожні вісім раундів. Нарешті, по закінченні останнього раунду, над *L* і *R* виконується операція XOR з іншими частинами ключа, і половини поєднуються, утворюючи блок шифртексту.

Хоча частини ключа використовуються для операції XOR із блоком шифрування на початку й кінці виконання алгоритму, головне призначення ключа – генерація S-блоків.

Ці S-блоки секретні, по суті, це частина ключа. Повний розмір ключа алгоритму Khufu дорівнює 512 біт (64 байт), алгоритм надає спосіб генерації S-блоків по ключу.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Програма має простий та інтуїтивно зрозумілий інтерфейс, який зображений на рисунку 5.1. З рисунку головного вікна можна побачити що інтерфейс головного вікна розподілено на наступні функціональні розділи:

– Функціональних кнопок ПЗ: Сканувати; Побудова мережі; Блокування; Встановлення фільтру; Налаштування; Виявлення.

– Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші: Довідка.

– Верхнього меню: Файл; Дані; Фільтри; Налаштування; Довідка.

– Розділу виведення результату роботи системи: Поточний стан; Дані мережних адрес.

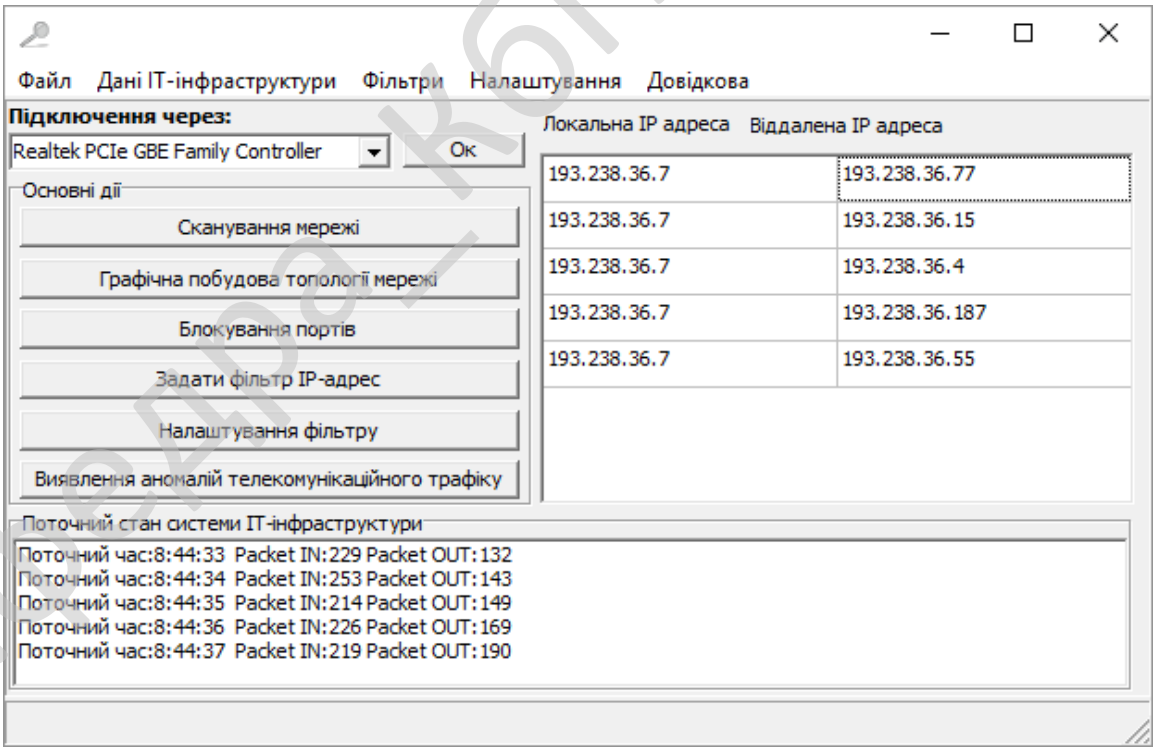

Рисунок 5.1 – Головне вікно ПЗ

На рисунку 5.2 зображено авторські дані розробленого програмного забезпечення. Розроблена програма має дуже простий і зрозумілий інтерфейс з користувачем. Кожен, хто в достатньому обсязі володіє операційним середовищем Windows без особливих складностей освоїть і цю програму, оскільки її інтерфейс інтуїтивно зрозумілий. Якщо програма не видала ніяких помилок, і працює, то можна використовувати, інакше слід слідувати інструкціям, які пропонує програма.

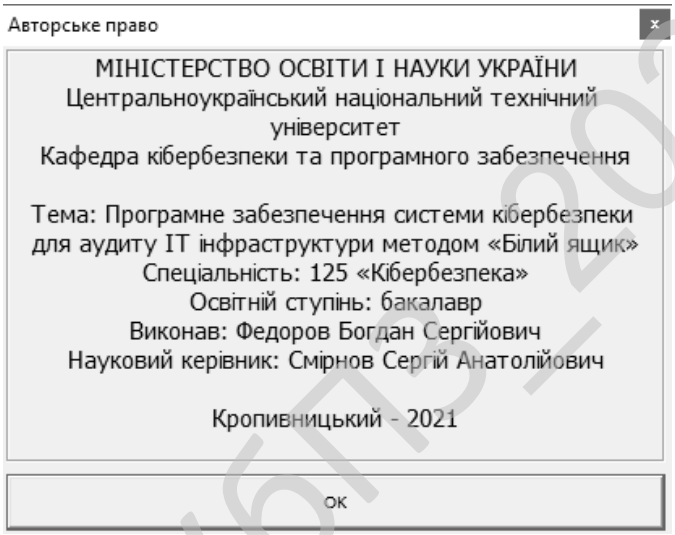

Рисунок 5.2 – Авторське право

Процес впровадження програмного забезпечення, це процес налаштування програмного забезпечення під певні умови використання, а також навчання користувачів роботі з програмним продуктом. Впровадження програмного забезпечення це усі дії, що роблять розроблену програмну систему готовою до використання. Даний процес є частинною життєвого циклу програмного забезпечення. Таким чином у результаті вищерозглянутого можна стверджувати що розроблено інтерфейс системи у відповідності з вибраною метою роботи. Система містить максимальний необхідний набір функцій придатних для виконання будь-яких дій для забезпечення повноцінної роботи програми. Далі розглянемо висновки та використані літературні джерела.

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання кваліфікаційної бакалаврської роботи, призначено для системи кібербезпеки для аудиту IT інфраструктури методом «Білий ящик».

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем кібербезпеки для аудиту IT інфраструктури методом «Білий ящик».

– Досліджена система кібербезпеки для аудиту IT інфраструктури методом «Білий ящик».

– На основі отриманих результатів досліджень створена програмна реалізація системи кібербезпеки для аудиту IT інфраструктури методом «Білий ящик».

Розроблені під час виконання кваліфікаційної бакалаврської роботи алгоритми дозволяють успішно вирішувати завдання для аудиту IT інфраструктури методом «Білий ящик».

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об’єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня RAD Studio Delphi 10.4.1. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для системи кібербезпеки для аудиту IT інфраструктури методом «Білий ящик». Це дозволило мінімізувати строк розробки програмного

забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 10.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм Khufu.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Kovalenko O., Popereshnyak S., Grinenko S., Grinenko O., Radivilova T. «Methods for Assessing the Maturity Levels of Software Ecosystems». *CEUR Workshop Proceedings* Volume 2654, 2019, Pages 251-261. Режим доступу: https://www.scopus.com/record/display.uri?eid=2-s2.0-85083214331&origin=AuthorNamesList&txGid=5633fba897776a6e0f3d5633fbcd3fbe **(Scopus)**.

2. Kovalenko O., Drieieva H., Simakhin V., Bondar S., Drieiev O., Zhumadilova M. «Multifractal Properties of Traffic Generator Based on Markov Chains ». *CEUR Workshop Proceedings* Volume 2588, 2019, Pages 567-579. Режим доступу: https://www.scopus.com/record/display.uri?eid=2-s2.0-85083214331&origin=AuthorNamesList&txGid=176e2cada8976a6e0f3d5633fbcd3fbe **(Scopus)**.

3. Kovalenko Oleksandr Qualitative risk analysis of software development / Oleksandr Kovalenko, Jamil Al-Azzeh, Oleksii Smirnov, Anna Kovalenko, Serhii Smirnov // Asian Journal of Information Technology. – Volume 17 Issue 3. – Medwell Journals. – 2018. – P. 218-230. ISSN: 1682-3915. URL: http://medwelljournals.com/abstract/?doi=ajit.2018.218.230 Doi: ajit.2018.218.230

4. Kovalenko Oleksandr, The mathematical model of the testing technology for DOM XSS vulnerabilities / O. Kovalenko, O. Smirnov, A.Kovalenko, S. Smirnov, V. Vialkova // Scientific & practical cyber security journal (SPCSJ) Volume 2 Issue 1, P. 22-28. Georgia. Tbilisi. Scientific Cyber Security Association (SCSA), 2018 ISSN: 2587-4667. URL: https://journal.scsa.ge/wp-content/uploads/2018/12/04-3-o.kovalenko-a.kovalenko-o.smirnov-s.smirnov-v.vialkova.pdf

5. Коваленко А.В. Технология тестирования DOM XSS уязвимости / А.В. Коваленко, А.С. Коваленко, А.А. Смирнов, С.А. Смирнов // Scientific & practical cyber security journal (SPCSJ) Volume 1. Issue 1. P. 38-45 Georgia. Tbilisi. Scientific Cyber Security Association (SCSA), 2017 ISSN: 2587-

4667. URL: https://journal.scsa.ge/wp-content/uploads/2018/12/8-dom-xss-testing-technology-vulnerabilities.pdf

6. Коваленко О.В. Моделі та методи розроблення програмного забезпечення комп'ютерних систем для підвищення безпеки даних: монографія / О.В. Коваленко // К.: Вид. «КОД» – 2019. – 305 с.

7. Коваленко А.В. Методы качественного анализа и количественной оценки рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Информационные технологии в управлении, образовании, науке и промышленности: монография / Под редакцией профессора В.С. Пономаренко. – Х.: Видавець Рожко С.Г., 2016. – 566 с.

8. Коваленко А.В. Разработка метода управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Інформаційні технології: проблеми та перспективи: монографія / За загальною редакцією В.С. Пономаренка. – Х.: Видавець Рожко С.Г., 2017. – 447 с.

9. Коваленко А.В. Комплекс математических моделей технологии тестирования web-приложений / А.В. Коваленко, А.А. Смирнов // Інформаційні технології: сучасний стан та перспективи: монографія / За загальною редакцією В.С. Пономаренка. – Х.: ТОВ «ДІСА ПЛЮС», 2018. – 461 с.

10. Коваленко А.В. Задачи распознавания ситуаций в ERP системах / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник наукових праць "Системи обробки інформації". – Випуск 4(120). – Х.: ХУПС – 2014. – С. 161-164.

11. Коваленко А.В. Методы качественного анализа и количественной оценки рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник наукових праць "Системи обробки інформації". – Випуск 5(142). – Х.: ХУПС – 2016. – С. 153-157.

12. Коваленко А.В. Проблемы анализа и оценки рисков информационной деятельности / А.В. Коваленко, А.А. Смирнов, Н.Н. Якименко, А.П. Доренский // Збірник наукових праць "Системи обробки інформації". –

Випуск 3(140). – Х.: ХУПС – 2016. – С. 40-42.

13. Коваленко А.В. Метод качественного анализа рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, Н.Н. Якименко, А.П. Доренский // Наука і техніка Повітряних Сил Збройних Сил України. – Випуск 2(23). – Харків: ХУПС. – 2016. – С. 150-158.

14. Коваленко А.В. Метод количественной оценки рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, Н.Н. Якименко, А.П. Доренский // Збірник наукових праць Харківського університету Повітряних Сил. Випуск 2 (47). – Харків: ХУПС. – 2016. – С. 128-133.

15. Коваленко А.В. Использование псевдобулевых методов бивалентного программирования для управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Системи управління, навігації та зв'язку. – Випуск 1 (37). – Полтава: ПолтНТУ. – 2016. – С. 98-103.

16. Коваленко А.В. Метод управления рисками разработки программного обеспечения / А.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 2 (38). – Полтава: ПолтНТУ. – 2016. – С. 93-100.

17. Коваленко А.В. Технология тестирования уязвимости к SQL инъекциям / А.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 5 (45). – Полтава: ПолтНТУ. – 2017. – С. 66-71.

18. Коваленко А.В. Масштабирование имитационной модели технологии тестирования безопасности / А.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 6 (46). – Полтава: ПолтНТУ. – 2017. – С. 181-184.

19. Коваленко А.В. Имитационная модель технологии тестирования безопасности Web-приложений / А.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 1 (47). – Полтава: ПолтНТУ. – 2018. – С. 114-123.

20. Коваленко О.В. Методи якісного аналізу та кількісної оцінки ризиків розроблення програмного забезпечення/ О.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 3 (49). – Полтава: ПолтНТУ. – 2018. – С. 116-125.

21. Коваленко О.В. Управління ризиками розроблення програмного

забезпечення за умови обмеженості коштів виділених на усунення помилок безпеки/ О.В. Коваленко // Техніка в сільськогосподарському виробництві, галузеве машинобудування, автоматизація. – Випуск 31. – Кропивницький: ЦНТУ. – 2018. – С. 128-140.

22. Коваленко О.В. GERT-мережевий синтез технології тестування на вразливість WEB-додатків/ О.В. Коваленко // Захист інформації. – Випуск 20(2) – К.: НАУ. – 2018. – С. 89-94.

23. Коваленко О.В. Імітаційна модель технології тестування безпеки на основі положень теорії масштабування / О.В. Коваленко // Безпека інформації. – Випуск 24 (2). – К.: НАУ. – 2018. – С. 110-117.

24. Коваленко О.В. Оцінка ефективності технології тестування безпеки / О.В. Коваленко // Вчені записки Таврійського національного університету імені В.І. Вернадського. Серія: Технічні науки. Том 29 (68) № 2, 2018. – С. 137-141

25. Коваленко О.В. Методи та засоби управління безпекою додатків / О.В. Коваленко // Інформаційно-керуючі системи на залізничному транспорті. №4, 2018. – С. 41-44.

26. Коваленко О.В. Розробка інформаційної технології передтестової компіляції та розподілу доступу / О.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 4 (50). – Полтава: ПолтНТУ. – 2018. – С. 115-119.

27. Коваленко О.В. Аналіз та дослідження інформаційних технологій розробки програмного забезпечення/ О.В. Коваленко // Вчені записки Таврійського національного університету імені В.І. Вернадського. Серія: Технічні науки. Том 29 (68) № 5, 2018. – С. 131-137.

28. Коваленко О.В. Удосконалений метод управління ризиками розроблення програмного забезпечення на основі напівмарковської моделі прийняття рішень/ О.В. Коваленко // Сучасні інформаційні системи. – Випуск 2(3). – Харків. – 2018. – С. 41-48.

29. Коваленко О.В. Математичні моделі технології тестування DOM XSS

вразливості та вразливості до SQL ін'єкцій / О.В. Коваленко // Вісник Черкаського державного технологічного університету. Серія : Технічні науки №4, 2018. – С. 29-36.

30. Коваленко О.В. Математична модель технології тестування вразливості до SQL ін’єкцій / О.В. Коваленко // Системи управління, навігації та зв’язку. – Випуск 6 (58). – Полтава: ПолтНТУ. – 2019. – С. 43-47.

31. Коваленко О.В. Математична модель технології тестування комплексу DOM XSS вразливостей для аналітичної оцінки часових витрат / О.В. Коваленко // Центральноукраїнський науковий вісник. Технічні науки. № 2(33). с. 173-180, 2019.

32. Коваленко А.В. Проблемы анализа и оценки рисков информационной деятельности / А.В. Коваленко, А.А. Смирнов // Збірник наукових праць II міжнародної науково-практичної конференції «Актуальні питання забезпечення кібернетичної безпеки та захисту інформації». м. Київ. 24-27 лютого 2016 р. – Київ: Європейський університет. – 2016. – С. 138-139.

33. Коваленко А.В. Анализ и оценка рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез «Securitea informationala 2015-2016». Conferenta internationala (editia a XII-a). Chisinau. Moldova. 3 martie 2016. – Chisinau: ADSEM. – 2016. – P. 96-102.

34. Коваленко А.В. Исследование источников и причин риска разработки программного обеспечения, этапов и работ, при выполнении которых возникает риск / А.В. Коваленко, А.А. Смирнов // Збірник тез VII всеукраїнської науково-практичної конференції "Інформатика та системні науки (ІСН-2016)". м. Полтава. 10-12 березня 2016 р. – Полтава.: ПУЕТ – 2016. – С. 264-266.

35. Коваленко А.В. Оценка показателя чистой приведенной стоимости для количественной оценки рисков проекта разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез науково-практичної конференції “Проблеми кібербезпеки інформаційно-телекомунікаційних

систем”. м. Київ. 10-11 березня 2016 р. – Київ: КНУ ім. Тараса Шевченко – 2016. – С. 81-82.

36. Коваленко А.В. Методика структурной идентификации рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез Міжнародної науково-практичної конференції «Інформаційна безпека та комп'ютерні технології» (IS&CT). м. Кіровоград. 24-25 березня 2016 р. – Кіровоград: КНТУ. – 2016. – С. 71-72.

37. Коваленко А.В. Методы качественного анализа рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез першої міжнародної науково-практичної конференції «Проблеми науково-технічного та правового забезпечення кібербезпеки у сучасному світі» (ПНПЗК-2016). м. Харків. 30 березня – 1 квітня 2016 р. – Харків: НТУ «ХПІ». – 2016. – С. 6-7.

38. Коваленко А.В. Структурная идентификация рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез XVIII міжнародного науково-практичного семінару «Комбінаторні конфігурації та їх застосування». м. Кіровоград. 15-16 квітня 2016 р. – Кіровоград: КНТУ. – 2016. – С. 175-182.

39. Коваленко А.В. Исследование разработанной методики структурной идентификации рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез VIII міжнародної науково-практичної конференції “Проблеми і перспективи розвитку IT-індустрії”. м. Харків. 28-29 квітня 2016 р. – Харків: ХНЕУ. – 2016. – С. 49.

40. Коваленко А.В. Исследование дерева рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез III міжнародної науково-практичної конференції «Інформаційна та економічна безпека» (INFECO-2016)». м. Харків. 28-30 квітня 2016 р. – Харків: ХННІ ДВНЗ «УБС». – 2016. – С. 174-178.

41. Коваленко А.В. Методы качественного анализа и количественной

оценки рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Сборник тезисов XII международной конференции "Стратегия качества в промышленности и образовании". г. Варна. Болгария. 30 мая – 02 июня 2016 г – Варна. ТУВ. – 2016. – С. 585-589.

42. Коваленко А.В. Разработка метода управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Матеріали Всеукраїнської науково-практичної конференції «Кібербезпека в Україні: правові та організаційні питання». м. Одеса, 21 жовтня 2016 р. – Одеса : ОДУВС, 2016. – С.146-148.

43. Коваленко А.В. Метод управления рисками разработки программного обеспечения с использованием псевдобулевых методов бивалентного программирования / А.В. Коваленко, А.А. Смирнов // Матеріали Всеукраїнської науково-практичної конференції «Актуальні задачі та досягнення у галузі кібербезпеки». м. Кропивницький, 23-25 листопада 2016 року – Кропивницький: ЦНТУ, 2016. – С. 162.

44. Коваленко А.В. Псевдобулевые методы бивалентного программирования для управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко, С.А. Смирнов // Збірник наукових праць III міжнародної науково-практичної конференції «Актуальні питання забезпечення кібернетичної безпеки та захисту інформації». м. Київ. 22-25 лютого 2017 р. – Київ: Європейський університет. – 2017. – С. 158-162.

45. Коваленко А.В. Метод управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез II науково-практичної конференції “Проблеми кібербезпеки інформаційно-телекомунікаційних систем”. м. Київ. 23-24 березня 2017 р. – Київ: КНУ ім. Тараса Шевченко – 2017. – С. 203-205.

46. Коваленко А.В. Алгоритмы анализа уязвимостей при управлении рисками разработки программного обеспечения / А.В. Коваленко,

А.А. Смирнов, А.С. Коваленко // Conferenta internationala (editia a XIII-a). «Securitea informationala 2017». Chisinau. Republic of Moldova. 4-5 aprilie 2017. – Chisinau: ADSEM. – 2017. – P. 19-22.

47. Коваленко А.В. Алгоритм анализа DOM XSS уязвимости при управлении рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник тез дев'ятнадцятого міжнародного науково-практичного семінару «Комбінаторні конфігурації та їх застосування». м. Кропивницький 7-8 квітня 2017 р. – Кропивницький: ГЛА НАУ. – 2017. – С. 125-127.

48. Коваленко А.В. Алгоритм анализа уязвимости SQL Injection для управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник тез другої міжнародної науково-технічної конференції «Проблеми науково-технічного та правового забезпечення кібербезпеки у сучасному світі» (ПНПЗК-2017). м. Харків.10-12 квітня 2017 р. – Харків: НТУ «ХПІ». – 2017. – С. 27.

49. Коваленко А.В. Метод управления рисками разработки программного обеспечения на основе алгоритмов анализа уязвимостей / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник тез Міжнародної науково-практичної конференції «Інформаційна безпека та комп'ютерні технології» (IS&CT). м. Кіровоград. 20-22 квітня 2017 р. Кіровоград: КНТУ. – 2017. – С. 92.

50. Коваленко А.В. Алгоритмы анализа DOM XSS уязвимости и уязвимости SQL Injection при управлении рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник тез IX міжнародної науково-практичної конференції "Проблеми і перспективи розвитку IT-індустрії". м. Харків. 20-21 квітня 2017 р. – Харків: ХНЕУ. – 2017. – С. 61.

51. Kovalenko O.V. Method of testing the dom xss vulnerability / Kovalenko Oleksandr, Kovalenko Anna, Smirnov Oleksii, Smirnov Serhii // International

Conference «information technologies, systems and networks ITSN-2017». Chisinau, Republic of Moldova. 17 – 18 October 2017. – Chisinau: Academy of Sciences of Moldova, Military Academy of Armed Forces “Alexandru cel Bun”. – 2017. – P. 7.

52. Коваленко О.В. Метод тестування DOM XSS уразливості / О.В. Коваленко, О.А. Смірнов, А.С. Коваленко, С.А. Смірнов // Збірник тез всеукраїнської науково-практичної інтернет-конференції «Автоматика та комп’ютерно-інтегровані технології у промисловості, телекомунікаціях, енергетиці та транспорті». м. Кропивницький. 16-17 листопада 2017 р. – Кропивницький: ЦНТУ. – 2017. – С. 198-199.

53. Коваленко О.В. GERT-модель технології тестування DOM XSS уразливості / О.В. Коваленко, А.С. Коваленко, О.А. Смірнов, С.А. Смірнов // Збірник наукових праць IV міжнародної науково-практичної конференції «Актуальні питання забезпечення кібернетичної безпеки та захисту інформації». м. Київ. 21-24 лютого 2018 р. – Київ: Європейський університет. – 2018. – С. 65-70.

54. Коваленко О.В. Технології тестування уразливостей Web-застосунків з використанням GERT-моделі / О.В. Коваленко, А.С. Коваленко, О.А. Смірнов, С.А. Смірнов // Збірник тез всеукраїнської науково-практичної конференції "Комп'ютерні інтелектуальні системи та мережі (КІСМ-2018)". м. Кривий Ріг. 21-23 березня 2018 р. – Кривий Ріг.: ДВНЗ КНУ – 2018. – С. 227-230.

55. Коваленко А.В. Тестирование уязвимости Web-приложений к атаке вида межсайтовый скриптинг / А.В. Коваленко, А.С. Коваленко, А.А. Смирнов, С.А. Смирнов // Збірник тез «Securitea informationala 2018». Conferenta internationala (editia a XIV-a). Chisinau. Moldova. 20-21 martie 2018. – Chisinau: ADSEM. – 2018. – P. 54-56.

56. Коваленко А.В. Комплекс математических моделей технологии тестирования web-приложений / А.В. Коваленко, А.С. Коваленко, А.А. Смирнов, С.А. Смирнов // Збірник тез X міжнародної науково-практичної

конференції “Проблеми і перспективи розвитку ІТ-індустрії”. м. Харків. 19-20 квітня 2018 р. – Харків: ХНЕУ. – 2018. – С. 38.

57. Коваленко О.В. Розробка методу передтестової компіляції й розподілу доступу / О.В. Коваленко, А.С. Коваленко, О.А. Смірнов, С.А. Смірнов // Збірник наукових праць III міжнародної науково-практичної конференції “Інформаційна безпека та комп’ютерні технології”, м. Кропивницькй. 19-20 квітня 2018 р. – Кропивницький: ЦНТУ. – 2018. – С. 214-215.

58. Коваленко О.В. Оцінка ефективності технологій тестування безпеки уразливостей DOM XSS й SQL-ін'єкцій / О.В. Коваленко, А.С. Коваленко, О.А. Смірнов, С.А. Смірнов // Сборник тезисов XIV международной конференции "Стратегия качества в промышленности и образовании", Варна, Болгария. 04-07 июня 2018 г – Варна. ТУВ. – 2018. – С. 271-274.

59. Коваленко О.В. Аналіз основних підходів математичного моделювання та методологій для забезпечення максимальних показників безпеки програмного забезпечення / О.В. Коваленко, А.С. Коваленко // Збірник наукових праць всеукр. наук.-практ. конф. здобувачів вищої освіти й молодих учених «Комп’ютерна інженерія і кібербезпека : досягнення та інновації, м. Кропивницькй. 27-29 листопада 2018

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування.....2
2 Підстава для розробки.....2
3 Мета та призначення розробки.....2
4 Джерела розробки.....2
5 Технічні вимоги.....2
5.1 Вміст проекту.....2
5.2 Показники призначення.....3
5.3 Вимоги до функціональних характеристик.....3
5.4 Вимоги до архітектури.....3
5.5 Вимоги до надійності.....3
5.6 Умови експлуатації.....4
5.7 Вимоги до складу та параметрів технічних засобів.....4
5.8 Вимоги до інформаційної і програмної сумісності.....4
5.8.1 Обладнання.....4
5.8.2 Мова програмування.....4
5.8.3 Вхідні дані.....5
5.8.4 Вихідні дані.....5
6 Вимоги до програмної документації.....5
7 Перелік документів, що розробляються.....5
8 Етапи розробки.....6
9 Порядок контролю та приймання.....6

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи кібербезпеки для аудиту IT інфраструктури методом «Білий ящик».

2 Підстава для розробки

Підставою для розробки служить завдання на кваліфікаційну бакалаврську роботу, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 185-02 від 28.12.2020 року).

3 Мета та призначення розробки

Метою кваліфікаційної бакалаврської роботи є розробка програмного забезпечення системи кібербезпеки для аудиту IT інфраструктури методом «Білий ящик».

4 Джерела розробки

Джерелом цієї кваліфікаційної бакалаврської роботи є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи кібербезпеки для аудиту IT інфраструктури методом «Білий ящик»;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows XP/Vista/7/8/10 і з с умісними з цією платформою пристроями і прикладним програмним забезпечення.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows XP/Vista/7/8/10.

5.8.1 Обладнання

Комп'ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14" 1Mb або сумісні з ним.

5.8.2 Мова програмування

Середовище RAD Studio Delphi 10.4.1.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Перелік документів, що розробляються

– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Пояснювальна записка – 80 аркушів.

8 Етапи розробки

8.1 Збір і обробка інформації по темі кваліфікаційної бакалаврської роботи. Постановка задачі на виконання кваліфікаційної бакалаврської роботи (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень кваліфікаційної бакалаврської роботи.

8.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання кваліфікаційної бакалаврської роботи на попередній захист 22.05.2021 р.

11.2 Подання кваліфікаційної бакалаврської роботи на захист 7.06.2021 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник кваліфікаційної бакалаврської роботи

_____ Смірнов С.А.

Програмне забезпечення системи кібербезпеки для аудиту IT інфраструктури методом «Білий ящик»

Лістинг програми

Код документу 12

Носій: CD/DVD-диск

Загальна кількість аркушів: 49

Літера: РП

Кропивницький – 2021 року

Файл TCP_IP.pas- монітор TCP/IP з'єднань системи кібербезпеки для аудиту IT інфраструктури методом «Білий ящик»

```
unit TCP_IP;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls, IPHelper, IpHlpApi, Buttons;

type
  TForm2 = class(TForm)
    StaticText2: TStaticText;
    StaticText3: TStaticText;
    TCPMemo: TMemo;
    UDPMemo: TMemo;
    Timer1: TTimer;
    cbTimer: TCheckBox;
    btRTTI: TSpeedButton;
    SpeedButton1: TSpeedButton;
    edtRTTI: TEdit;
    procedure Timer1Timer(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure btRTTIClick(Sender: TObject);
    procedure cbRecentIPsClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
      procedure DOIpStuff;
  public
    { Public declarations }
  end;

var
  Form2: TForm2;

implementation

{$R *.dfm}

procedure TForm2.Timer1Timer(Sender: TObject);
begin
  if cbTimer.State = cbCHECKED then
 begin
   Timer1.Enabled := false;
   DoIPStuff;
   Timer1.Enabled := true;
 end;
end;

procedure TForm2.DOIpStuff;
begin

  Get_TCPTable( TCPMemo.Lines );
  Get_UDPTable( UDPMemo.Lines );

end;

procedure TForm2.SpeedButton1Click(Sender: TObject);
begin
  Speedbutton1.Enabled := false;
  DoIPStuff;
  Speedbutton1.Enabled := true;
end;
```

```
procedure TForm2.btRTTIClick(Sender: TObject);
var
  IPadr         : dword;
  Rtt, HopCount : longint;
  Res           : integer;
begin
  btRTTI.Enabled := false;
  Screen.Cursor := crHOURGLASS;
  IPadr := Str2IPAddr( edtRTTI.Text );
  Res := Get_RTTAndHopCount( IPadr, 128, RTT, HopCount );
  if Res = NO_ERROR then
    ShowMessage( ' Час запиту '
      + inttostr( rtt ) + ' ms, '
      + inttostr( HopCount )
      + ' hops to : ' + edtRTTI.Text
      )
  else
    ShowMessage( ' Відбулася помилка:' + #13
                 + ICMPErr2Str( Res ) ) ;
  btRTTI.Enabled := true;
  Screen.Cursor := crDEFAULT;

end;

procedure TForm2.cbRecentIPsClick(Sender: TObject);
begin
//edtRTTI.Text := cbRecentIPs.Items[cbRecentIPs.ItemIndex];
end;

procedure TForm2.FormCreate(Sender: TObject);
begin

  if LoadIpHlp then
  begin
      DOIpStuff;
      Timer1.Enabled := true;
  end
  else
      ShowMessage( ' Інтернет помічник DLL не є доступним, або не підтримується'
) ;
end;

end.
```

Файл Stat.pas- статистика мережі системи кібербезпеки для аудиту IT інфраструктури методом «Білий ящик»

```
unit Stat;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls, IPHelper, IpHlpApi, Buttons;

type
  TForm3 = class(TForm)
    StaticText7: TStaticText;
    TCPStatMemo: TMemo;
    StaticText5: TStaticText;
    IPStatsMemo: TMemo;
    StaticText12: TStaticText;
    ICMPInMemo: TMemo;
    ICMPOutMemo: TMemo;
    StaticText4: TStaticText;
    UDPStatsMemo: TMemo;
    Timer1: TTimer;
    cbTimer: TCheckBox;
    btRTTI: TSpeedButton;
    edtRTTI: TEdit;
    procedure Timer1Timer(Sender: TObject);
    procedure btRTTIClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
     procedure DOIpStuff;
  public
    { Public declarations }
  end;

var
  Form3: TForm3;

implementation

{$R *.dfm}

procedure TForm3.DOIpStuff;
begin

  Get_TCPStatistics( TCPStatMemo.Lines );
  Get_IPStatistics( IPStatsMemo.Lines );
  Get_UDPStatistics( UDPStatsMemo.Lines );
  Get_ICMPStats( ICMPInMemo.Lines, ICMPOutMemo.Lines );

end;

procedure TForm3.Timer1Timer(Sender: TObject);
begin
  if cbTimer.State = cbCHECKED then
 begin
   Timer1.Enabled := false;
   DoIPStuff;
   Timer1.Enabled := true;
 end;
end;

procedure TForm3.btRTTIClick(Sender: TObject);
var
  IPadr         : dword;
  Rtt, HopCount : longint;
```

```
  Res           : integer;
begin
  btRTTI.Enabled := false;
  Screen.Cursor := crHOURGLASS;
  IPadr := Str2IPAddr( edtRTTI.Text );
  Res := Get_RTTAndHopCount( IPadr, 128, RTT, HopCount );
  if Res = NO_ERROR then
    ShowMessage( ' Час запиту '
      + inttostr( rtt ) + ' ms, '
      + inttostr( HopCount )
      + ' hops to : ' + edtRTTI.Text
      )
  else
    ShowMessage( ' Помилка:' + #13
                 + ICMPErr2Str( Res ) ) ;
  btRTTI.Enabled := true;
  Screen.Cursor := crDEFAULT;

end;

procedure TForm3.FormCreate(Sender: TObject);
begin

  if LoadIpHlp then
  begin
      DOIpStuff;
      Timer1.Enabled := true;
  end
  else
      ShowMessage( 'Інтернет помічник DLL не є доступним, або не підтримується'
) ;
end;

end.
```

Основна програма

Файл WhiteBoxMethod. dpr основної програми

```
program WhiteBoxMethod;

uses
  Forms,
  Main in ' Main.pas'  {MainForm},
  About in ' About.pas'  {Form1},
  TCP_IP in ' TCP_IP.pas'  {Form2},
  Stat in ' Stat.pas'  {Form3};

{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TMainForm, MainForm);
  Application.CreateForm(TForm1, Form1);
  Application.CreateForm(TForm2, Form2);
  Application.CreateForm(TForm3, Form3);
  Application.Run;
end.
```

Файл Main.pas основної програми

```
unit Main;

interface

// опис бібліотек

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls, ComCtrls, Stat,
  ShellAPI, ShlObj, ImgList, TCP_IP, About;

//опис типів

type
  TMainForm = class(TForm)
    gbxShares: TGroupBox;
    lbxShares: TListBox;
    gbxSessions: TGroupBox;
    lvSessions: TListView;
    bvlSessions: TBevel;
    gbxFiles: TGroupBox;
    btnGetShares: TButton;
    btnCloseShares: TButton;
    btnAddShares: TButton;
    btnCloseSession: TButton;
    btnGetSessions: TButton;
    bvlTopSessions: TBevel;
    plButtonFiles: TPanel;
    btnGetFiles: TButton;
    btnCloseFile: TButton;
    bvlLeftFiles: TBevel;
    plFiles: TPanel;
    lvFiles: TListView;
    bvlTopFiles: TBevel;
    gbxTraffic: TGroupBox;
    lvTraffic: TListView;
    bvlTraffic: TBevel;
    tmrTraffic: TTimer;
    Button1: TButton;
    rgScope: TRadioGroup;
    GroupBox1: TGroupBox;
    cbUsageAll: TCheckBox;
    cbUsageConnectable: TCheckBox;
    cbUsageContainer: TCheckBox;
    GroupBox2: TGroupBox;
    cbTypeAny: TCheckBox;
    cbTypeDisk: TCheckBox;
    cbTypePrint: TCheckBox;
    NetTree: TTreeView;
    ImageList1: TImageList;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    function  IsNT(var Value: Boolean): Boolean;
    procedure btnGetSharesClick(Sender: TObject);
    procedure btnCloseSharesClick(Sender: TObject);
    function  SelectDirectory: String;
    procedure btnAddSharesClick(Sender: TObject);
    function  CardinalToTimeStr(Value:Cardinal):String;
    procedure btnGetSessionsClick(Sender: TObject);
    procedure btnCloseSessionClick(Sender: TObject);
    procedure btnGetFilesClick(Sender: TObject);
    procedure btnCloseFileClick(Sender: TObject);
    procedure tmrTrafficTimer(Sender: TObject);
    procedure Button1Click(Sender: TObject);
```

```
    procedure NetTreeCustomDrawItem(Sender: TCustomTreeView;
      Node: TTreeNode; State: TCustomDrawState; var DefaultDraw: Boolean);
    procedure NetTreeDblClick(Sender: TObject);
    procedure NetTreeGetImageIndex(Sender: TObject; Node: TTreeNode);
    procedure Button4Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);

//опис типів та записів


  private
    { Private declarations }
  public
    { Public declarations }
    SessionCloseKey: array [0..512] of SmallInt;
    procedure Open_Do_Close_Enum(const ParentNode: TTreeNode;
        ResScope, ResType, ResUsage: DWORD; const NetContainerToOpen:
PNetResource);
  //  function OpenEnum(const NetContainerToOpen: PNetResource;
 //        ResScope, ResType, ResUsage: DWORD): THandle;
 //   function EnumResources(const ParentNode: TTreeNode;
  //       ResScope, ResType, ResUsage: DWORD; hNetEnum: THandle): UINT;
  end;

type
 TShareInfo2 = packed record
  shi2_netname : PWChar;
  shi2_type: DWORD;
  shi2_remark :PWChar;
  shi2_permissions: DWORD;
  shi2_max_uses : DWORD;
  shi2_current_uses : DWORD;
  shi2_path : PWChar;
  shi2_passwd : PWChar;
 end;
 PShareInfo2 = ^ TShareInfo2;
 TShareInfo2Array = array [0..512] of TShareInfo2;
 PShareInfo2Array = ^ TShareInfo2Array;

type
 TShareInfo50 = packed record
  shi50_netname : array [0..12] of Char;
  shi50_type : Byte;
  shi50_flags : Word;
  shi50_remark : PChar;
  shi50_path : PChar;
  shi50_rw_password : array [0..8] of Char;
  shi50_ro_password : array [0..8] of Char;
 end;

type
 TSessionInfo502 = packed record
   Sesi502_cname: PWideChar;
   Sesi502_username: PWideChar;
   Sesi502_num_opens: DWORD;
   Sesi502_time: DWORD;
   Sesi502_idle_time: DWORD;
   Sesi502_user_flags: DWORD;
   Sesi502_cltype_name: PWideChar;
   Sesi502_transport: PWideChar;
 End;
 PSessionInfo502 = ^TSessionInfo502;
 TSessionInfo502Array = array[0..512] of TSessionInfo502;
 PSessionInfo502Array = ^TSessionInfo502Array;

type
 TSessionInfo50 = packed record
   Sesi50_cname       : PChar;
```

```
    Sesi50_username    : PChar;
    sesi50_key         : Cardinal;
    sesi50_num_conns   : Word;
    sesi50_num_opens   : Word;
    sesi50_time        : Cardinal;
    sesi50_idle_time   : Cardinal;
    sesi50_protocol    : Byte;
    pad1               : Byte;
  end;

type
  TFileInfo3 = packed record
    fi3_id          : DWORD;
    fi3_permissions : DWORD;
    fi3_num_locks   : DWORD;
    fi3_pathname    : PWChar;
    fi3_username    : PWChar;
  end;
  PFileInfo3 = ^TFileInfo3;
  TFileInfo3Array = array[0..512] of TFileInfo3;
  PFileInfo3Array = ^TFileInfo3Array;

type
  TFileInfo50 = packed record
    fi50_id          : Cardinal;
    fi50_permissions : WORD;
    fi50_num_locks   : WORD;
    fi50_pathname    : PChar;
    fi50_username    : PChar;
    fi50_sharename   : PChar;
  end;

type
   TMibIfRow = packed record
     wszName            : array[0..255] of WideChar;
     dwIndex            : DWORD;
     dwType             : DWORD;
     dwMtu              : DWORD;
     dwSpeed            : DWORD;
     dwPhysAddrLen      : DWORD;
     bPhysAddr          : array[0..7] of Byte;
     dwAdminStatus      : DWORD;
     dwOperStatus       : DWORD;
     dwLastChange       : DWORD;
     dwInOctets         : DWORD;
     dwInUcastPkts      : DWORD;
     dwInNUCastPkts     : DWORD;
     dwInDiscards       : DWORD;
     dwInErrors         : DWORD;
     dwInUnknownProtos  : DWORD;
     dwOutOctets        : DWORD;
     dwOutUCastPkts     : DWORD;
     dwOutNUCastPkts    : DWORD;
     dwOutDiscards      : DWORD;
     dwOutErrors        : DWORD;
     dwOutQLen          : DWORD;
     dwDescrLen         : DWORD;
     bDescr             : array[0..255] of Char;
   end;
   TMibIfArray = array [0..512] of TMibIfRow;
   PMibIfRow = ^TMibIfRow;
   PMibIfArray = ^TMibIfArray;

type
  TMibIfTable = packed record
    dwNumEntries     : DWORD;
    Table            : TMibIfArray;
  end;
  PMibIfTable = ^TMibIfTable;
```

```
var
NetShareEnumNT:function (     servername:PWChar;
                          level:DWORD;
                          bufptr:Pointer;
                          prefmaxlen:DWORD;
                          entriesread,
                          totalentries,
                          resume_handle:LPDWORD): DWORD; stdcall;

var
NetShareEnum:function ( pszServer   : PChar;
                        sLevel           : Cardinal;
                        pbBuffer    : Pchar;
                        cbBuffer    : Cardinal;
                        pcEntriesRead,
                        pcTotalAvail: Pointer):DWORD; stdcall;

var
NetShareDelNT:function (servername: PWideChar;
                        netname: PWideChar;
                        reserved: DWORD): LongInt; stdcall;

var
NetShareDel:function (  pszServer,
                        pszNetName:PChar;
                        usReserved:Word): DWORD; stdcall;

var
NetShareAddNT: function(servername: PWideChar;
                        level: DWORD;
                        buf: Pointer;
                        parm_err: LPDWORD): DWORD; stdcall;

var
NetShareAdd: function ( pszServer:Pchar;
                        sLevel:Cardinal;
                        pbBuffer:PChar;
                        cbBuffer:Word):DWORD; stdcall;

Var
NetSessionEnumNT:function(servername,
                        UncClientName,
                        username:PWChar;
                        level:DWORD;
                        bufptr:Pointer;
                        prefmaxlen:DWORD;
                        entriesread,
                        totalentries,
                        resume_handle:LPDWORD):DWORD; stdcall;

var
NetSessionEnum:function(pszServer:PChar;
                        sLevel: DWORD;
                        pbBuffer:Pointer;
                        cbBuffer:DWORD;
                        pcEntriesRead,
                        pcTotalAvial:Pointer):integer; stdcall;

var
NetSessionDelNT:function(ServerName,
                        UncClientName,
                        username:PWChar):DWORD; stdcall;

var
NetSessionDel:function( pszServer:PChar;
                        pszClientName: PChar;
```

```
                        sReserved: SmallInt):DWORD; stdcall;

var
NetFileEnumNT:function( servername,
                        basepath,
                        username:PWChar;
                        level:DWORD;
                        bufptr:Pointer;
                        prefmaxlen:DWORD;
                        entriesread,
                        totalentries,
                        resume_handle:LPDWORD):DWORD; stdcall;

var
NetFileEnum:function(     pszServer,
                        pszBasePath:PChar;
                        sLevel:DWORD;
                        pbBuffer:Pointer;
                        cbBuffer:DWORD;
                        pcEntriesRead,
                        pcTotalAvail:pointer):integer; stdcall;

var
NetFileClose:function(  ServerName:PWideChar;
                        FileId:DWORD):DWORD; stdcall;

var
NetFileClose2:function(pszServer:PChar;
                        ulFileId:LongWord):DWORD; stdcall;

var
GetIfTable:function(    pIfTable    : PMibIfTable;
                        pdwSize     : PULONG;
                        bOrder      : Boolean ): DWORD; stdcall;

var
  MainForm: TMainForm;

implementation

{$R *.dfm}

{ TMainForm }

//////////////////////////
//
//  Спочатку нам потрібно визначиться, під якою системою ми працюємо,
// щоб довідатися яку частину коду (для NT чи ні) використовувати в цей момент.
// Для цього напишемо невелику функцію, що і буде визначати тип системи.
//

function TMainForm.IsNT(var Value: Boolean): Boolean;
var Ver: TOSVersionInfo;
    BRes: Boolean;
begin
  Ver.dwOSVersionInfoSize := SizeOf(TOSVersionInfo);
  BRes := GetVersionEx(Ver);
  if not BRes then   //Перевірка
  begin
    Result := False; //Інформація не отримана
    Exit;            //ідемо
  end else
    Result := True;  //Інформація отримана

  case Ver.dwPlatformId of //визначаємося
    VER_PLATFORM_WIN32_NT       : Value := True;  //Windows NT- підходить
    VER_PLATFORM_WIN32_WINDOWS  : Value := False; //Windows 9 x-Me- підходить
    VER_PLATFORM_WIN32s         : Result := False //Windows 3.x- не підходить
  end;
```

```
end;

////////////////////////////
//
//  Одержання всіх відкритих загальних ресурсів
//

procedure TMainForm.btnGetSharesClick(Sender: TObject);
var
  i:Integer;
  FLibHandle : THandle;
  ShareNT : PShareInfo2Array;     //<= Змінні
  entriesread,totalentries:DWORD; //<= для Windows NT
  Share : array [0..512] of TShareInfo50; //<= Змінні
  pcEntriesRead,pcTotalAvail:Word;        //<= для Windows 9 x-Me
  OS: Boolean;
begin
  lbxShares.Items.Clear;
  if not IsNT(OS) then Close; //Визначаємо тип системи

  if OS then begin  //Код для NT
    FLibHandle := LoadLibrary(' NETAPI32.DLL' ); //Завантажуємо бібліотеку
    if FLibHandle = 0 then Exit;
    //Зв' язуємо функцію
    @NetShareEnumNT := GetProcAddress(FLibHandle,' NetShareEnum' );
    if not Assigned(NetShareEnumNT) then //Перевірка
    begin
      FreeLibrary(FLibHandle);
      Exit;
    end;
    ShareNT := nil; //Очищаємо покажчик на масив структур
    //Виклик функції
    if NetShareEnumNT(nil,2,@ShareNT,DWORD(-1),
                      @entriesread,@totalentries,nil) <> 0 then
    begin  //Якщо виклик невдалий вивантажуємо бібліотеку
      FreeLibrary(FLibHandle);
      Exit;
    end;
    if entriesread > 0 then //Обробка результатів
    for i:= 0 to entriesread- 1 do
      lbxShares.Items.Add(String(ShareNT^[i].shi2_netname));
  end else begin //Код для 9 x-me
    FLibHandle := LoadLibrary(' SVRAPI.DLL' ); //Завантажуємо бібліотеку
    if FLibHandle = 0 then Exit;
    //Зв' язуємо функцію
    @NetShareEnum := GetProcAddress(FLibHandle,' NetShareEnum' );
    if not Assigned(NetShareEnum) then //Перевірка
    begin
      FreeLibrary(FLibHandle);
      Exit;
    end;
    //Виклик функції
    if NetShareEnum(nil,50,@Share,SizeOf(Share),
                @pcEntriesRead,@pcTotalAvail)<> 0 then
    begin  //Якщо виклик невдалий вивантажуємо бібліотеку
      FreeLibrary(FLibHandle);
      Exit;
    end;
    if pcEntriesRead > 0 then //Обробка результатів
    for i:= 0 to pcEntriesRead- 1 do
      lbxShares.Items.Add(String(Share[i].shi50_netname));
  end;
  FreeLibrary(FLibHandle); //Не забуваємо вивантажити бібліотеку
end;

////////////////////////////
//
//  Закриття загального ресурсу
//
```

```
procedure TMainForm.btnCloseSharesClick(Sender: TObject);
var
  OS:Boolean;
  FLibHandle : THandle;
  Name9x:array [0..12] of Char;
  NameNT:PWChar;
  i:Integer;
  ShareName: String;
begin
  if not IsNT(OS) then Close; //Визначаємо тип системи

  if lbxShares.Items.Count = 0 then Exit;
  for i:= 0 to lbxShares.Items.Count-1 do
    if lbxShares.Selected[i] then Break;  //Шукаємо обраний елемент
  if not lbxShares.Selected[i] then Exit; //Якщо не знайдений ідемо
  ShareName := lbxShares.Items.Strings[i];

  if OS then begin //Код для NT
    FLibHandle := LoadLibrary(' NETAPI32.DLL' );
    if FLibHandle = 0 then Exit;
    @NetShareDelNT := GetProcAddress(FLibHandle,' NetShareDel' );
    if not Assigned(NetShareDelNT) then //Перевірка
    begin
      FreeLibrary(FLibHandle);
      Exit;
    end;
    i:= SizeOf(WideChar)*256;
    GetMem(NameNT,i);  //Виділяємо пам' ять під змінну
    StringToWideChar(ShareName,NameNT,i); //Перетворимо в PWideChar
    NetShareDelNT(nil,NameNT,0);   //Видаляємо ресурс
    FreeMem(NameNT);  //Звільняємо пам' ять
  end else begin  //Код для 9 х-ме
    FLibHandle := LoadLibrary(' SVRAPI.DLL' );
    if FLibHandle = 0 then Exit;
    @NetShareDel := GetProcAddress(FLibHandle,' NetShareDel' );
    if not Assigned(NetShareDel) then //Перевірка
    begin
      FreeLibrary(FLibHandle);
      Exit;
    end;
    FillChar(Name9x, SizeOf(Name9x), #0); //Очищаємо масив
    move(ShareName[1],Name9x[0],Length(ShareName)); //Заповнюємо масив
    NetShareDel(nil,@Name9x,0); //Видаляємо ресурс
  end;
  FreeLibrary(FLibHandle);
end;

//////////////////////////
//
//  Показу діалогу вибору директорії
//

function TMainForm.SelectDirectory: String;
var
  lpItemID : PItemIDList;
  BrowseInfo : TBrowseInfo;
  DisplayName : array[0..MAX_PATH] of Char;
  TempPath : array[0..MAX_PATH] of Char;
begin
  FillChar(BrowseInfo, sizeof(TBrowseInfo), #0);
  BrowseInfo.hwndOwner := Handle;
  BrowseInfo.pszDisplayName := @DisplayName;
  BrowseInfo.lpszTitle := ' Specify a directory' ;
  BrowseInfo.ulFlags := BIF_RETURNONLYFSDIRS;
  lpItemID := SHBrowseForFolder(BrowseInfo);
  if Assigned(lpItemId) then begin
    SHGetPathFromIDList(lpItemID, TempPath);
    GlobalFreePtr(lpItemID);
```

```
  end else Result := ' ' ;
  Result := String(TempPath);
end;

////////////////////////////
//
//  Додавання загального ресурсу
//

procedure TMainForm.btnAddSharesClick(Sender: TObject);
const
  STYPE_DISKTREE = 0;
  ACCESS_ALL =  258;
  SHI50F_FULL = 258;
var
  FLibHandle : THandle;
  Share9x : TShareInfo50;
  ShareNT : TShareInfo2;
  TmpDir, TmpName: String;
  TmpDirNT, TmpNameNT: PWChar;
  OS: Boolean;
  TmpLength: Integer;
begin
  TmpDir := SelectDirectory; //Визначаємо шлях до наступного ресурсу
  TmpName := InputBox(' Share name' ,' Enter name' ,' Test' ); //Визначаємо ім'
я під яким він буде видний у мережі системи кібербезпеки для аудиту IT
інфраструктури методом «Білий ящик»
  if TmpDir = ' '  then Exit;

  if not IsNT(OS) then Close; //З' ясовуємо тип системи

  if OS then begin //Код для NT
    FLibHandle := LoadLibrary(' NETAPI32.DLL' );
    if FLibHandle = 0 then Exit;
    @NetShareAddNT := GetProcAddress(FLibHandle,' NetShareAdd' );
    if not Assigned(NetShareAddNT) then
    begin
      FreeLibrary(FLibHandle);
      Exit;
    end;
    TmpLength := SizeOF(WideChar)*256; //Визначаємо необхідний розмір

    GetMem(TmpNameNT, TmpLength); //Конвертуємо в PWChar
    StringToWideChar(TmpName, TmpNameNT, TmpLength);
    ShareNT.shi2_netname := TmpNameNT; //Ім' я

    ShareNT.shi2_type := STYPE_DISKTREE; //Тип ресурсу
    ShareNT.shi2_remark := ' '; //Коментар
    ShareNT.shi2_permissions := ACCESS_ALL; //Доступ
    ShareNT.shi2_max_uses := DWORD(-1); // Кіл-У максим. підключ.
    ShareNT.shi2_current_uses := 0; // Кіл-У тік підкл.

    GetMem(TmpDirNT, TmpLength);
    StringToWideChar(TmpDir, TmpDirNT, TmpLength);
    ShareNT.shi2_path := TmpDirNT; //Шлях до ресурсу

    ShareNT.shi2_passwd := ' ' ; //Пароль

    NetShareAddNT(nil,2,@ShareNT, nil); //Додаємо ресурс
    FreeMem (TmpNameNT); //звільняємо пам' ять
    FreeMem (TmpDirNT);
  end else begin
    FLibHandle := LoadLibrary(' SVRAPI.DLL' );
    if FLibHandle = 0 then Exit;
    @NetShareAdd := GetProcAddress(FLibHandle,' NetShareAdd' );
    if not Assigned(NetShareAdd) then
    begin
      FreeLibrary(FLibHandle);
      Exit;
```

```
    end;
    FillChar(Share9x.shi50_netname, SizeOf(Share9x.shi50_netname), #0);
    move(TmpName[1],Share9x.shi50_netname[0],Length(TmpName)); //Ім' я
    Share9x.shi50_type := STYPE_DISKTREE; //Тип ресурсу
    Share9x.shi50_flags := SHI50F_FULL; //Доступ
    FillChar(Share9x.shi50_remark,
      SizeOf(Share9x.shi50_remark), #0); //Коментар
    FillChar(Share9x.shi50_path,
      SizeOf(Share9x.shi50_path), #0);
    Share9x.shi50_path := PAnsiChar(TmpDir); //Шлях до ресурсу
    FillChar(Share9x.shi50_rw_password,
      SizeOf(Share9x.shi50_rw_password), #0); //Пароль повного доступу
    FillChar(Share9x.shi50_ro_password,
      SizeOf(Share9x.shi50_ro_password), #0); //Пароль для читання
    NetShareAdd(nil,50,@Share9x,SizeOf(Share9x));
  end;
  FreeLibrary(FLibHandle);
end;

////////////////////////////
//
//  Помітьте що активний і неактивний час сесій буде даватися нам
//  у вигляді кіл-ті секунд (тип Cardinal). Напишемо невелику
//  функцію, задача якої буде перетворювати кіл-у секунд у більше
//  звичну форму відображення.
//

function TMainForm.CardinalToTimeStr(Value: Cardinal): String;
var d,h,m,s: Real;
begin
    d:=0;
    h:=0;
    m:=0;
    s:=Value;
    if s > 59 then begin
        m:=int(s / 60);
        s:= s-s-(m*60);
    end;
    if m > 59 then begin
        h:=int(m/60);
        m:= m-m-(h*60);
    end;
    if h > 23 then begin
        d:=int(h/24);
        h:= h-h-(d*24);
    end;
    Result:=' ' ;
    if (d>0) then Result:=Result+floattostr(d)+'  d. ' ;
    if (h<9) then Result:=Result+' 0' +floattostr(h)+' :'  else
Result:=Result+floattostr(h)+' :' ;
    if (m<9) then Result:=Result+' 0' +floattostr(m)+' :'  else
Result:=Result+floattostr(m)+' :' ;
    if (s<9) then Result:=Result+' 0' +floattostr(s)     else
Result:=Result+floattostr(s);
end;

////////////////////////////
//
//  Одержання списку сесій
//

procedure TMainForm.btnGetSessionsClick(Sender: TObject);
var
  OS: Boolean;
  FLibHandle : THandle;
  SessionInfo50: array [0..512] of TSessionInfo50;
  SessionInfo502 : PSessionInfo502Array;
  TotalEntries,EntriesReadNT: DWORD;
  EntriesRead,TotalAvial: Word;
```

```
  i:integer;
begin
  lvSessions.Items.Clear;

  if not IsNT(OS) then Close; //З' ясовуємо тип системи

  if OS then begin  //Код для NT
    FLibHandle := LoadLibrary(' NETAPI32.DLL' );
    if FLibHandle = 0 then Exit;
    @NetSessionEnumNT := GetProcAddress(FLibHandle, ' NetSessionEnum' );
    if not Assigned(NetSessionEnumNT) then
    begin
      FreeLibrary(FLibHandle);
      Exit;
    end;
    SessionInfo502 := nil;
    if NetSessionEnumNT(nil,nil,nil,502,@SessionInfo502,DWORD(-
1),@entriesreadNT, @totalentries, nil)=0 then
    for i:=0 to EntriesReadNT-1 do
    begin
      with lvSessions.Items.Add do  //Заповнення даними зі структури
      begin
        Caption := string(SessionInfo502^[i].sesi502_cname); //Ім' я комп' ютера
        SubItems.Add(SessionInfo502^[i].sesi502_username);   //Ім' я користувача
        SubItems.Add(IntToStr(SessionInfo502^[i].sesi502_num_opens));
//Відкритих ресурсів
        SubItems.Add(CardinalToTimeStr(SessionInfo502^[i].Sesi502_Time)); //Час
активний
        SubItems.Add(CardinalToTimeStr(SessionInfo502^[i].sesi502_idle_time));
//Час не активний
      end;
    end;
  end else begin //Код для Windows 9 x-Me
   FLibHandle := LoadLibrary(' SVRAPI.DLL' );
   if FLibHandle = 0 then Exit;
   @NetSessionEnum := GetProcAddress(FLibHandle, ' NetSessionEnum' );
    if not Assigned(NetSessionEnum) then
    begin
      FreeLibrary(FLibHandle);
      Exit;
    end;
   if NetSessionEnum
(nil,50,@SessionInfo50,SizeOf(SessionInfo50),@EntriesRead,@TotalAvial) = 0 then
   for i:=0 to EntriesRead-1 do
   begin
      with lvSessions.Items.Add do  //Заповнення даними зі структури
      begin
        Caption := string(SessionInfo50[i].Sesi50_cname); //Ім' я комп' ютера
        SubItems.Add(SessionInfo50[i].Sesi50_username);   //Ім' я користувача
        SubItems.Add(IntToStr(SessionInfo50[i].sesi50_num_opens)); //Відкритих
ресурсів
        SubItems.Add(CardinalToTimeStr(SessionInfo50[i].Sesi50_Time)); //Час
активний
        SubItems.Add(CardinalToTimeStr(SessionInfo50[i].sesi50_idle_time));
//Час не активний
        SessionCloseKey[i]:= SessionInfo50[i].sesi50_key; //Унікальний
ідентифікатор для закриття
      end;
   end;
 end;
 FreeLibrary(FLibHandle);
end;

//////////////////////////
//
//  Завершення обраної сесії
//

procedure TMainForm.btnCloseSessionClick(Sender: TObject);
```

```
var
  OS: Boolean;
  FLibHandle : THandle;
  CNameNT: PWideChar;
  CName9x: PAnsiChar;
  Key:SmallInt;
  i: Integer;
begin
  if not IsNT(OS) then Close; //З' ясовуємо тип системи

  if not Assigned(lvSessions.Selected) then Exit;
  i:= lvSessions.Selected.Index; //Визначаємо номер обраної сесії

  if OS then begin
    FLibHandle := LoadLibrary(' NETAPI32.DLL' );
    if FLibHandle = 0 then Exit;
    @NetSessionDelNT := GetProcAddress(FLibHandle, ' NetSessionDel' );
    if not Assigned(NetSessionDelNT) then
    begin
      FreeLibrary(FLibHandle);
      Exit;
    end;
    //Перетворимо дані в необхідний вид
    CNameNT := PWChar(WideString(' \\' +lvSessions.Items.Item[i].Caption));
    NetSessionDelNT(nil,CNameNT,nil);
  end else begin
    FLibHandle := LoadLibrary(' SVRAPI.DLL' );
    if FLibHandle = 0 then Exit;
    @NetSessionDel := GetProcAddress(FLibHandle, ' NetSessionDel' );
    if not Assigned(NetSessionDel) then
    begin
      FreeLibrary(FLibHandle);
      Exit;
    end;
    //Перетворимо дані в необхідний вид
    CName9x := PAnsiChar(lvSessions.Items.Item[i].Caption);
    key := SessionCloseKey[i]; //Беремо ключ із масиву
    NetSessionDel(nil,CName9x,Key);
  end;
  FreeLibrary(FLibHandle);
end;

//////////////////////////
//
//  Одержання списку відкритих файлів
//

procedure TMainForm.btnGetFilesClick(Sender: TObject);
var
  OS: Boolean;
  FLibHandle : THandle;
  FileInfoNT: PFileInfo3Array;
  FileInfo9x: array [0..512] of TFileInfo50;
  TotalEntries,EntriesReadNT: DWORD;
  EntriesRead,TotalAvial: Word;
  i:integer;
begin
  lvfiles.Items.Clear;

  if not IsNT(OS) then Close; //З' ясовуємо тип системи

  if OS then begin  //Код для NT
    FLibHandle := LoadLibrary(' NETAPI32.DLL' );
    if FLibHandle = 0 then Exit;
    @NetFileEnumNT := GetProcAddress(FLibHandle, ' NetFileEnum' );
    if not Assigned(NetFileEnumNT) then
    begin
      FreeLibrary(FLibHandle);
      Exit;
```

```
    end;
    FileInfoNT := nil;
    if NetFileEnumNT(nil,nil,nil,3,@FileInfoNT,DWORD(-1),@entriesreadNT,
@totalentries, nil)=0 then
    for i:=0 to EntriesReadNT-1 do
    begin
      with lvFiles.Items.Add do  //Заповнення даними зі структури
      begin
        Caption := string(IntToStr(FileInfoNT^[i].fi3_id)); //Ідентифікатор
        SubItems.Add(FileInfoNT^[i].fi3_pathname);   //Шлях до файлу
        SubItems.Add(FileInfoNT^[i].fi3_username);   //Ім' я користувача
      end;
    end;
  end else begin //Код для Windows 9 x-Me
   FLibHandle := LoadLibrary(' SVRAPI.DLL' );
   if FLibHandle = 0 then Exit;
   @NetFileEnum := GetProcAddress(FLibHandle, ' NetFileEnum' );
    if not Assigned(NetFileEnum) then
    begin
      FreeLibrary(FLibHandle);
      Exit;
    end;
   if NetFileEnum (nil,
nil,50,@FileInfo9x,SizeOf(FileInfo9x),@EntriesRead,@TotalAvial)= 0 then
   for i:=0 to EntriesRead-1 do
   begin
      with lvFiles.Items.Add do  //Заповнення даними зі структури
      begin
        Caption := string(IntToStr(FileInfo9x[i].fi50_id)); //Ідентифікатор
        SubItems.Add(FileInfo9x[i].fi50_pathname); //Шлях до файлу
        SubItems.Add(FileInfo9x[i].fi50_username); //Ім' я користувача
      end;
   end;
 end;
 FreeLibrary(FLibHandle);
end;

//////////////////////////
//
//  Закриття файлу
//

procedure TMainForm.btnCloseFileClick(Sender: TObject);
var
  OS: Boolean;
  FLibHandle : THandle;
  i: Integer;
begin
  if not IsNT(OS) then Close; //З' ясовуємо тип системи

  if not Assigned(lvFiles.Selected) then Exit;
  i:= lvFiles.Selected.Index; //Визначаємо номер обраного файлу

  if OS then begin //Код для NT
    FLibHandle := LoadLibrary(' NETAPI32.DLL' );
    if FLibHandle = 0 then Exit;
    @NetFileClose := GetProcAddress(FLibHandle, ' NetFileClose' );
    if not Assigned(NetFileClose) then
    begin
      FreeLibrary(FLibHandle);
      Close;
    end;
    NetFileClose(nil,StrToInt(lvFiles.Items.Item[i].Caption));//Закриваємо файл
  end else begin //Код для Windows 9 x-Me
    FLibHandle := LoadLibrary(' SVRAPI.DLL' );
    if FLibHandle = 0 then Exit;
    @NetFileClose2 := GetProcAddress(FLibHandle, ' NetFileClose2' );
    if not Assigned(NetFileClose2) then
    begin
```

```
      FreeLibrary(FLibHandle);
      Close;
    end;
    NetFileClose2(nil,StrToInt(lvFiles.Items.Item[i].Caption));
  end;
  FreeLibrary(FLibHandle);
end;

////////////////////////////
//
//  Визначаємо вхідний- вихідний трафік
//

procedure TMainForm.tmrTrafficTimer(Sender: TObject);
  // Допоміжна функція, що перетворить MAC адресу до "нормального" виду
  //Визначаємо спеціальний тип, щоб можна було передати у функцію масив
  type TMAC = array [0..7] of Byte;
  //Як перше значення масив, друге значення, розмір даних у масиві
  function GetMAC(Value: TMAC; Length: DWORD): String;
  var
    i: Integer;
  begin
    if Length = 0 then Result := '  00-00-00'  else
    begin
      Result := ' ' ;
      for i:= 0 to Length-2 do
        Result := Result + IntToHex(Value[i],2)+' -' ;
      Result := Result + IntToHex(Value[ Length-1],2);
    end;
  end;

//Сама процедура
var
  FLibHandle : THandle;
  Table: TMibIfTable;
  i : integer;
  Size   : integer;
begin
  tmrTraffic.Enabled := false; //Припиняємо про всякий випадок таймер
  lvTraffic.Items.BeginUpdate;
  lvTraffic.Items.Clear;  //Очищаємо список
  FLibHandle := LoadLibrary(' IPHLPAPI.DLL' ); //Завантажуємо бібліотеку
  if FLibHandle = 0 then Exit;
  @GetIfTable := GetProcAddress(FLibHandle, ' GetIfTable' );
  if not Assigned(GetIfTable) then
  begin
    FreeLibrary(FLibHandle);
    Close;
  end;

  Size := SizeOf(Table);
  if GetIfTable(@Table, @Size, false ) = 0 then //Виконуємо функцію
    for i:= 0 to Table.dwNumEntries-1 do begin
      with lvTraffic.Items.Add do begin //Виводимо результати
        Caption := String(Table.Table[i].bDescr); //Найменування інтерфейсу
        SubItems.Add(GetMAC(TMAC(Table.Table[i].bPhysAddr),
          Table.Table[i].dwPhysAddrLen)); //MAC адреса
        SubItems.Add(IntToStr(Table.Table[i].dwInOctets)); //Усього прийнято
байт
        SubItems.Add(IntToStr(Table.Table[i].dwOutOctets)); //Усього відправлено
байт
      end;
    end;
  lvTraffic.Items.EndUpdate;
  FreeLibrary(FLibHandle);
  tmrTraffic.Enabled := true; //Не забуваємо активувати таймер
end;
```

```
 function OpenEnum(const NetContainerToOpen: PNetResource; ResScope, ResType,
ResUsage: DWORD): THandle;
var
 hNetEnum: THandle;
begin
 Result:=0;
 if (NO_ERROR<>WNetOpenEnum(ResScope, ResType, ResUsage,
                                  NetContainerToOpen, hNetEnum))
 then ShowMessage(‘ Помилка!’ )
 else Result:=hNetEnum;
end;

function EnumResources(const ParentNode: TTreeNode;
ResScope, ResType, ResUsage: DWORD; hNetEnum: THandle): UINT;
 function ShowResource(const ParentNode: TTreeNode; Res: TNetResource):
TTreeNode;
  begin
   Result:=MainForm.NetTree.Items.AddChild(ParentNode,
string(Res.lpRemoteName));
  end;

const
 RESOURCE_BUF_ENTRIES = 2000;

var
 ResourceBuffer: array[1..RESOURCE_BUF_ENTRIES] of TNetResource;
 i, ResourceBuf, EntriesToGet: dword;
 NewNode: TTreeNode;
begin
 Result:=0;
 while true do
  begin
   ResourceBuf:=sizeof(ResourceBuffer);
   EntriesToGet:=RESOURCE_BUF_ENTRIES;
   if (NO_ERROR<>WNetEnumResource(hNetEnum, EntriesToGet,
                               @ResourceBuffer, ResourceBuf))
   then
    begin
     case GetLastError() of
      NO_ERROR:  // проход буферу без перемикання
      Break;
      ERROR_NO_MORE_ITEMS:
          // Повертає о у тому випадку, коли останов
          // RESOURCE_BUF_ENTRIES данні на попередньому виклику, щоб
          // WNetEnumResource, та були точно
          // RESOURCE_BUF_ENTRIES данні в запису на момент
          // попереднього виклику
      Exit;
      else ShowMessage(Помилка!’ );
      Result:=1;
      Exit;
     end;
    end;
    for i:=1 to EntriesToGet do
    begin
     NewNode:=ShowResource(ParentNode, ResourceBuffer[i]);
     if (ResourceBuffer[i].dwUsage and RESOURCEUSAGE_CONTAINER)<>0
     then MainForm.Open_Do_Close_Enum(NewNode, ResScope, ResType, ResUsage,
@ResourceBuffer[i]);
     Application.ProcessMessages;
    end;
  end;
end;

procedure TMainForm.Open_Do_Close_Enum(const ParentNode: TTreeNode; ResScope,
ResType, ResUsage: DWORD; const NetContainerToOpen: PNetResource);
var
 hNetEnum: THandle;
begin
```

```
 hNetEnum:=OpenEnum(NetContainerToOpen, ResScope, ResType, ResUsage);
 if (hNetEnum=0)
 then Exit;
 EnumResources(ParentNode, ResScope, ResType, ResUsage, hNetEnum);
 if (NO_ERROR<>WNetCloseEnum(hNetEnum))
 then ShowMessage(‘ WNetCloseEnum Помилка’ );
end;

procedure TMainForm.Button1Click(Sender: TObject);
var
 ResScope, ResType, ResUsage: dword;
begin
 Button1.Caption:=‘ Пошук мережних ресурсів. Чекайте...’ ;
 Button1.Enabled:=false;
 //
 NetTree.Items.Clear;
 case rgScope.ItemIndex of
  1: ResScope:=RESOURCE_GLOBALNET;
  2: ResScope:=RESOURCE_REMEMBERED;
  else ResScope:=RESOURCE_CONNECTED;
 end;
 ResType:=0;
 if cbTypeAny.Checked
 then ResType:=ResType or RESOURCETYPE_ANY;
 if cbTypeDisk.Checked
 then ResType:=ResType or RESOURCETYPE_DISK;
 if cbTypePrint.Checked
 then ResType:=ResType or RESOURCETYPE_PRINT;
 ResUsage:=0;
 if cbUsageConnectable.Checked
 then ResUsage:=ResUsage or RESOURCEUSAGE_CONNECTABLE;
 if cbUsageContainer.Checked
 then ResUsage:=ResUsage or RESOURCEUSAGE_CONTAINER;
 Open_Do_Close_Enum(NetTree.Items.Add(nil, ‘ Network Resources’ ),
                              ResScope, ResType, ResUsage, nil);
 //
 Button1.Caption:=‘ Обновити список ресурсів’ ;
 Button1.Enabled:=true;

end;

procedure TMainForm.NetTreeCustomDrawItem(Sender: TCustomTreeView;
  Node: TTreeNode; State: TCustomDrawState; var DefaultDraw: Boolean);
begin
if cdsSelected in State
 then Sender.Canvas.Font.Style:=Sender.Canvas.Font.Style+[fsUnderline];
end;

procedure TMainForm.NetTreeDblClick(Sender: TObject);
begin
ShellExecute(0,’ open’ ,PChar(NetTree.Selected.Text),’ ‘ ,’ ‘ ,SW_SHOW);
end;

procedure TMainForm.NetTreeGetImageIndex(Sender: TObject; Node: TTreeNode);
begin
 if Node.HasChildren
 then Node.ImageIndex:=1
 else Node.ImageIndex:=0;
end;

procedure TMainForm.Button4Click(Sender: TObject);
begin
Form1.Show;
end;

procedure TMainForm.Button2Click(Sender: TObject);
begin
Form2.Show;
end;
```

```
procedure TMainForm.Button3Click(Sender: TObject);
begin
Form3.Show;
end;

end.
```

Файл IPHLPAPI.pas- обробка API функцій

```
unit IPHLPAPI;

interface
uses
  Windows, winsock;

const
  VERSION       = ' 1.5' ;

//------- Заголовок з Microsoft IPTYPES.H-H-----------------

const
  ANY_SIZE      = 1;
  MAX_ADAPTER_DESCRIPTION_LENGTH = 128; // arb.
  MAX_ADAPTER_NAME_LENGTH = 256; // змінна
  MAX_ADAPTER_ADDRESS_LENGTH = 8; // змінна
  DEFAULT_MINIMUM_ENTITIES = 32; // змінна
  MAX_HOSTNAME_LEN = 128; // змінна
  MAX_DOMAIN_NAME_LEN = 128; // змінна
  MAX_SCOPE_ID_LEN = 256; // змінна

 // Вузлові типи ( NETBIOS)
  BROADCAST_NODETYPE = 1;
  PEER_TO_PEER_NODETYPE = 2;
  MIXED_NODETYPE = 4;
  HYBRID_NODETYPE = 8;

  NETBIOSTypes  : array[0..8] of string[20] =
    ( ' Невизначений' , ' Передача' , ' Рівень до рівня' , ' ' , ' Змішаний' , '
' , ' ' , ' ' , ' Гібрід'
     );

 // Типи адаптеру
{ v1.4-> 1.5
  IF_OTHER_ADAPTERTYPE = 0;
  IF_ETHERNET_ADAPTERTYPE = 1;
  IF_TOKEN_RING_ADAPTERTYPE = 2;
  IF_FDDI_ADAPTERTYPE = 3;
  IF_PPP_ADAPTERTYPE = 4;
  IF_LOOPBACK_ADAPTERTYPE = 5;
  IF_SLIP_ADAPTERTYPE = 6;

 found in ipifcons.h :
#define MIB_IF_TYPE_OTHER               1
#define MIB_IF_TYPE_ETHERNET            6
#define MIB_IF_TYPE_TOKENRING           9
#define MIB_IF_TYPE_FDDI                15
#define MIB_IF_TYPE_PPP                 23
#define MIB_IF_TYPE_LOOPBACK            24
#define MIB_IF_TYPE_SLIP                28
}
  IF_OTHER_ADAPTERTYPE = 1;
  IF_ETHERNET_ADAPTERTYPE = 6;
  IF_TOKEN_RING_ADAPTERTYPE = 9;
  IF_FDDI_ADAPTERTYPE = 15;
  IF_PPP_ADAPTERTYPE = 23;
  IF_LOOPBACK_ADAPTERTYPE = 24;
  IF_SLIP_ADAPTERTYPE = 28;

//  AdaptTypes    : array[0..6] of string[10] =
//    ( ' інший' , ' ethernet' , ' tokenring' , ' FDDI' , ' PPP' , ' loopback' ,
' SLIP'  );
  AdaptTypes    : array[1..28] of string[10] =
```

```
    ( ' інший' , ' ' , ' ' , ' ' , ' ' , ' ethernet' , ' ' , ' ' , ' tokenring'
,
     ' ' , ' ' , ' ' , ' ' , ' ' , ' FDDI' , ' ' , ' ' , ' ' , ' ' , ' ' , ' ' ,
' ' , ' PPP' ,
     ' loopback' , ' ' , ' ' , ' ' , ' SLIP'  );
// Кінець змін в типі адаптерів

//-------для інших MS заготовочних файлів-------------------

  MAX_INTERFACE_NAME_LEN = 256; { mrapi.h }
  MAXLEN_PHYSADDR = 8; { iprtrmib.h }
  MAXLEN_IFDESCR = 256; {-"--     }

//-----------------------------------

type
  TMacAddress = array[1..MAX_ADAPTER_ADDRESS_LENGTH] of byte;

//---IP адресні структури------------------------

  PTIP_ADDRESS_STRING = ^TIP_ADDRESS_STRING;
  TIP_ADDRESS_STRING = array[0..15] of char; //  IP рядок
  //
  PTIP_ADDR_STRING = ^TIP_ADDR_STRING;
  TIP_ADDR_STRING = packed record // для використання у зв' язних списках
    Next: PTIP_ADDR_STRING;
    IpAddress: TIP_ADDRESS_STRING;
    IpMask: TIP_ADDRESS_STRING;
    Context: DWORD;
  end;

//-----Fixed Info структура---------------------

  PTFixedInfo = ^TFixedInfo;
  TFixedInfo = packed record
    HostName: array[1..MAX_HOSTNAME_LEN + 4] of char;    // данні
    DomainName: array[1..MAX_DOMAIN_NAME_LEN + 4] of char;   // данні
    CurrentDNSServer: PTIP_ADDR_STRING;
    DNSServerList: TIP_ADDR_STRING;
    NodeType: UINT;
    ScopeID: array[1..MAX_SCOPE_ID_LEN + 4] of char;   // данні
    EnableRouting: UINT;
    EnableProxy: UINT;
    EnableDNS: UINT;
  end;

//-----структура мережного інтерфейсу-----------------------

//////////////////////
//                                                                  //
// Наступне є діючими станами для WAN да LAN інтерфейсів. //
// Порядок станів створений для визначення. Для      //
// стану >= CONNECTED можливо передавати данні зразу. Стан >= DISCONNECTED //
// може передавати деякі данні. Стан < DISCONNECTED може   //
// не передавати дані.                                              //
// карта з поміткою UNREACHABLE якщо DIM викликає InterfaceUnreachable для
//
// причин. Крім невдачі з'єднання.                                  //
//                                                                  //
// NON_OPERATIONAL- Перевірка для LAN інтерфейсу. Позначає карту що не працює
//
//                      або не з'єднується з картою.        //
// UNREACHABLE- Перевірка WAN інтерфейсів . Позначає, що віддалений сайт      //
//                      не з'єднується за потрібний час.
//
// DISCONNECTED- Перевірка WAN інтерфейсів . Позначає, що віддалений сайт
//
//                      не з'єднується.
//
```

```
// CONNECTING- Перевірка WAN інтерфейсів . Означає спробу з'єднання     //
//                      з сайтом, якого немає.              //
// CONNECTED- Перевірка WAN інтерфейсів . Позначає, що віддалений сайт      //
//                      з'єднується.                                        //
// OPERATIONAL- Перевірка LAN Interfaces. Позначає карту підключену //
//                      в праці.                                          //
//                                                                        //
// Усі дії користувачів записуються до  MIB-II значення    //
// можуть бути використовані                                          //
//                                                                        //
//////////////////////////

const
// данні додані до ipifcons.h
  IF_OPER_STATUS_NON_OPERATIONAL = 0 ;
  IF_OPER_STATUS_UNREACHABLE = 1 ;
  IF_OPER_STATUS_DISCONNECTED = 2 ;
  IF_OPER_STATUS_CONNECTING = 3 ;
  IF_OPER_STATUS_CONNECTED = 4 ;
  IF_OPER_STATUS_OPERATIONAL = 5 ;

  MIB_IF_TYPE_OTHER = 1 ;
  MIB_IF_TYPE_ETHERNET = 6 ;
  MIB_IF_TYPE_TOKENRING = 9 ;
  MIB_IF_TYPE_FDDI = 15 ;
  MIB_IF_TYPE_PPP = 23 ;
  MIB_IF_TYPE_LOOPBACK = 24 ;
  MIB_IF_TYPE_SLIP = 28 ;

  MIB_IF_ADMIN_STATUS_UP = 1 ;
  MIB_IF_ADMIN_STATUS_DOWN = 2 ;
  MIB_IF_ADMIN_STATUS_TESTING = 3 ;

  MIB_IF_OPER_STATUS_NON_OPERATIONAL = 0 ;
  MIB_IF_OPER_STATUS_UNREACHABLE = 1 ;
  MIB_IF_OPER_STATUS_DISCONNECTED = 2 ;
  MIB_IF_OPER_STATUS_CONNECTING = 3 ;
  MIB_IF_OPER_STATUS_CONNECTED = 4 ;
  MIB_IF_OPER_STATUS_OPERATIONAL = 5 ;

type
  PTMibIfRow = ^TMibIfRow;
  TMibIfRow = packed record
    wszName: array[1..MAX_INTERFACE_NAME_LEN] of WCHAR;
    dwIndex: DWORD;
    dwType: DWORD;        // дивись MIB_IF_TYPE
    dwMTU: DWORD;
    dwSpeed: DWORD;
    dwPhysAddrLen: DWORD;
    bPhysAddr: array[1..MAXLEN_PHYSADDR] of byte;
    dwAdminStatus: DWORD;    // дивись MIB_IF_ADMIN_STATUS
    dwOperStatus: DWORD;     // дивись MIB_IF_OPER_STATUS
    dwLastChange: DWORD;
    dwInOctets: DWORD;
    dwInUcastPkts: DWORD;
    dwInNUCastPkts: DWORD;
    dwInDiscards: DWORD;
    dwInErrors: DWORD;
    dwInUnknownProtos: DWORD;
    dwOutOctets: DWORD;
    dwOutUCastPkts: DWORD;
    dwOutNUCastPkts: DWORD;
    dwOutDiscards: DWORD;
    dwOutErrors: DWORD;
    dwOutQLen: DWORD;
    dwDescrLen: DWORD;
    bDescr: array[1..MAXLEN_IFDESCR] of char; //byte;
  end;
```

```
 //
  PTMibIfTable = ^TMIBIfTable;
  TMibIfTable = packed record
    dwNumEntries: DWORD;
    Table: array[0..ANY_SIZE- 1] of TMibIfRow;
  end;

//---ADAPTER INFO структура-----------------------

  PTIP_ADAPTER_INFO = ^TIP_ADAPTER_INFO;
  TIP_ADAPTER_INFO = packed record
    Next: PTIP_ADAPTER_INFO;
    ComboIndex: DWORD;
    AdapterName: array[1..MAX_ADAPTER_NAME_LENGTH + 4] of char;       // данні
    Description: array[1..MAX_ADAPTER_DESCRIPTION_LENGTH + 4] of char;    //
данні
    AddressLength: UINT;
    Address: array[1..MAX_ADAPTER_ADDRESS_LENGTH] of byte;      // данні
    Index: DWORD;
    aType: UINT;
    DHCPEnabled: UINT;
    CurrentIPAddress: PTIP_ADDR_STRING;
    IPAddressList: TIP_ADDR_STRING;
    GatewayList: TIP_ADDR_STRING;
    DHCPServer: TIP_ADDR_STRING;
    HaveWINS: BOOL;
    PrimaryWINSServer: TIP_ADDR_STRING;
    SecondaryWINSServer: TIP_ADDR_STRING;
    LeaseObtained: LongInt ; // UNIX час, секунди з 1970
    LeaseExpires: LongInt;   // UNIX час, секунди з 1970
    SpareStuff: array [1..200] of char ;   // данні- простір для списку IP адрес
  end;

//--------TCP структура---------------------

  PTMibTCPRow = ^TMibTCPRow;
  TMibTCPRow = packed record
    dwState: DWORD;
    dwLocalAddr: DWORD;
    dwLocalPort: DWORD;
    dwRemoteAddr: DWORD;
    dwRemotePort: DWORD;
  end;
  //
  PTMibTCPTable = ^TMibTCPTable;
  TMibTCPTable = packed record
    dwNumEntries: DWORD;
    Table: array[0..0] of TMibTCPRow;
  end;
  //
  PTMibTCPStats = ^TMibTCPStats;
  TMibTCPStats = packed record
    dwRTOAlgorithm: DWORD;
    dwRTOMin: DWORD;
    dwRTOMax: DWORD;
    dwMaxConn: DWORD;
    dwActiveOpens: DWORD;
    dwPassiveOpens: DWORD;
    dwAttemptFails: DWORD;
    dwEstabResets: DWORD;
    dwCurrEstab: DWORD;
    dwInSegs: DWORD;
    dwOutSegs: DWORD;
    dwRetransSegs: DWORD;
    dwInErrs: DWORD;
    dwOutRsts: DWORD;
    dwNumConns: DWORD;
  end;
```

```
//-----UDP СТРУКТУРА -------------------------

  PTMibUDPRow = ^TMibUDPRow;
  TMibUDPRow = packed record
    dwLocalAddr: DWORD;
    dwLocalPort: DWORD;
  end;
 //
  PTMibUDPTable = ^TMIBUDPTable;
  TMIBUDPTable = packed record
    dwNumEntries: DWORD;
    UDPTable: array[0..ANY_SIZE- 1] of TMibUDPRow;
  end;
 //
  PTMibUdpStats = ^TMIBUdpStats;
  TMIBUdpStats = packed record
    dwInDatagrams: DWORD;
    dwNoPorts: DWORD;
    dwInErrors: DWORD;
    dwOutDatagrams: DWORD;
    dwNumAddrs: DWORD;
  end;

//------IP СТРУКТУРА ------------------------

 //
  PTMibIPNetRow = ^TMibIPNetRow;
  TMibIPNetRow = packed record
    dwIndex: DWord;
    dwPhysAddrLen: DWord;
    bPhysAddr: TMACAddress;
    dwAddr: DWord;
    dwType: DWord;
  end;
  //
  PTMibIPNetTable = ^TMibIPNetTable;
  TMibIPNetTable = packed record
    dwNumEntries: DWORD;
    Table: array[0..ANY_SIZE- 1] of TMibIPNetRow;
  end;
  //
  PTMibIPStats = ^TMibIPStats;
  TMibIPStats = packed record
    dwForwarding: DWORD;
    dwDefaultTTL: DWORD;
    dwInReceives: DWORD;
    dwInHdrErrors: DWORD;
    dwInAddrErrors: DWORD;
    dwForwDatagrams: DWORD;
    dwInUnknownProtos: DWORD;
    dwInDiscards: DWORD;
    dwInDelivers: DWORD;
    dwOutRequests: DWORD;
    dwRoutingDiscards: DWORD;
    dwOutDiscards: DWORD;
    dwOutNoRoutes: DWORD;
    dwReasmTimeOut: DWORD;
    dwReasmReqds: DWORD;
    dwReasmOKs: DWORD;
    dwReasmFails: DWORD;
    dwFragOKs: DWORD;
    dwFragFails: DWORD;
    dwFragCreates: DWORD;
    dwNumIf: DWORD;
    dwNumAddr: DWORD;
    dwNumRoutes: DWORD;
  end;
  //
  PTMibIPAddrRow = ^TMibIPAddrRow;
```

```
  TMibIPAddrRow = packed record
    dwAddr: DWORD;
    dwIndex: DWORD;
    dwMask: DWORD;
    dwBCastAddr: DWORD;
    dwReasmSize: DWORD;
    Unused1,
    Unused2: WORD;
  end;
  //
  PTMibIPAddrTable = ^TMibIPAddrTable;
  TMibIPAddrTable = packed record
    dwNumEntries: DWORD;
    Table: array[0..ANY_SIZE- 1] of TMibIPAddrRow;
  end;

  //
  PTMibIPForwardRow = ^TMibIPForwardRow;
  TMibIPForwardRow = packed record
    dwForwardDest: DWORD;
    dwForwardMask: DWORD;
    dwForwardPolicy: DWORD;
    dwForwardNextHop: DWORD;
    dwForwardIFIndex: DWORD;
    dwForwardType: DWORD;
    dwForwardProto: DWORD;
    dwForwardAge: DWORD;
    dwForwardNextHopAS: DWORD;
    dwForwardMetric1: DWORD;
    dwForwardMetric2: DWORD;
    dwForwardMetric3: DWORD;
    dwForwardMetric4: DWORD;
    dwForwardMetric5: DWORD;
  end;
  //
  PTMibIPForwardTable = ^TMibIPForwardTable;
  TMibIPForwardTable = packed record
    dwNumEntries: DWORD;
    Table: array[0..ANY_SIZE- 1] of TMibIPForwardRow;
  end;

//---- ICMP-СТРУКТУРА ----------------------

  PTMibICMPStats = ^TMibICMPStats;
  TMibICMPStats = packed record
    dwMsgs: DWORD;
    dwErrors: DWORD;
    dwDestUnreachs: DWORD;
    dwTimeExcds: DWORD;
    dwParmProbs: DWORD;
    dwSrcQuenchs: DWORD;
    dwRedirects: DWORD;
    dwEchos: DWORD;
    dwEchoReps: DWORD;
    dwTimeStamps: DWORD;
    dwTimeStampReps: DWORD;
    dwAddrMasks: DWORD;
    dwAddrReps: DWORD;
  end;

  PTMibICMPInfo = ^TMibICMPInfo;
  TMibICMPInfo = packed record
    InStats: TMibICMPStats;
    OutStats: TMibICMPStats;
  end;

//---------імпорт до IPHLPAPI.DLL------------------

var
```

```
GetAdaptersInfo: function ( pAdapterInfo: PTIP_ADAPTER_INFO;
  pOutBufLen: PULONG ): DWORD; stdcall;

GetNetworkParams: function ( FixedInfo: PTFixedInfo; pOutPutLen: PULONG ):
        DWORD; stdcall;

GetTcpTable: function ( pTCPTable: PTMibTCPTable; pDWSize: PDWORD;
  bOrder: BOOL ): DWORD; stdcall;

GetTcpStatistics: function ( pStats: PTMibTCPStats ): DWORD; stdcall;

GetUdpTable: function ( pUdpTable: PTMibUDPTable; pDWSize: PDWORD;
 bOrder: BOOL ): DWORD; stdcall;

GetUdpStatistics: function ( pStats: PTMibUdpStats ): DWORD; stdcall;

GetIpStatistics: function ( pStats: PTMibIPStats ): DWORD; stdcall;

GetIpNetTable: function ( pIpNetTable: PTMibIPNetTable;
  pdwSize: PULONG;  bOrder: BOOL ): DWORD; stdcall;

GetIpAddrTable: function ( pIpAddrTable: PTMibIPAddrTable;
  pdwSize: PULONG; bOrder: BOOL ): DWORD; stdcall;

GetIpForwardTable: function ( pIPForwardTable: PTMibIPForwardTable;
  pdwSize: PULONG; bOrder: BOOL ): DWORD; stdCall;

GetIcmpStatistics: function ( pStats: PTMibICMPInfo ): DWORD; stdCall;

GetRTTAndHopCount: function ( DestIPAddress: DWORD; HopCount: PULONG;
  MaxHops: ULONG; RTT: PULONG ): BOOL; stdCall;

GetIfTable: function ( pIfTable: PTMibIfTable; pdwSize: PULONG;
  bOrder: boolean ): DWORD; stdCall;

GetIfEntry: function ( pIfRow: PTMibIfRow ): DWORD; stdCall;

// попередження – недокументована функція, можливі баги при використанні
GetFriendlyIfIndex: function (var IfIndex: DWORD): DWORD; stdcall;


const
    IpHlpDLL = ' IPHLPAPI.DLL' ;
var
    IpHlpModule: THandle;

    function LoadIpHlp: Boolean;

implementation

function LoadIpHlp: Boolean;
begin
    Result := True;
    if IpHlpModule <> 0 then Exit;

// відкрити DLL
    IpHlpModule := LoadLibrary (IpHlpDLL);
    if IpHlpModule = 0 then
    begin
        Result := false;
        exit ;
    end ;
    GetAdaptersInfo := GetProcAddress (IpHlpModule, ' GetAdaptersInfo' ) ;
    GetNetworkParams := GetProcAddress (IpHlpModule, ' GetNetworkParams' ) ;
    GetTcpTable := GetProcAddress (IpHlpModule, ' GetTcpTable' ) ;
    GetTcpStatistics := GetProcAddress (IpHlpModule, ' GetTcpStatistics' ) ;
    GetUdpTable := GetProcAddress (IpHlpModule, ' GetUdpTable' ) ;
    GetUdpStatistics := GetProcAddress (IpHlpModule, ' GetUdpStatistics' ) ;
```

```
    GetIpStatistics := GetProcAddress (IpHlpModule, ' GetIpStatistics' ) ;
    GetIpNetTable := GetProcAddress (IpHlpModule, ' GetIpNetTable' ) ;
    GetIpAddrTable := GetProcAddress (IpHlpModule, ' GetIpAddrTable' ) ;
    GetIpForwardTable := GetProcAddress (IpHlpModule, ' GetIpForwardTable' ) ;
    GetIcmpStatistics := GetProcAddress (IpHlpModule, ' GetIcmpStatistics' ) ;
    GetRTTAndHopCount := GetProcAddress (IpHlpModule, ' GetRTTAndHopCount' ) ;
    GetIfTable := GetProcAddress (IpHlpModule, ' GetIfTable' ) ;
    GetIfEntry := GetProcAddress (IpHlpModule, ' GetIfEntry' ) ;
    GetFriendlyIfIndex := GetProcAddress (IpHlpModule, ' GetFriendlyIfIndex' ) ;
end;

initialization
    IpHlpModule := 0 ;
finalization
    if IpHlpModule <> 0 then
    begin
        FreeLibrary (IpHlpModule) ;
        IpHlpModule := 0 ;
    end ;

end.
```

Файл IPHelper.pas- функції роботи з IP-Протоколом

```
unit IPHelper;


interface

uses
  Windows, Messages, SysUtils, Classes, Dialogs, IpHlpApi;

const
  NULL_IP       = '   0.0.  0.0' ;

//---перетворення добре відомих номерів портів до імен сервісів--------

type
  TWellKnownPort = record
    Prt: DWORD;
    Srv: string[20];
  end;


const
    // тільки найбільш популярні сервіси...
  WellKnownPorts: array[1..32] of TWellKnownPort
  = (
//    ( Prt: 0; Srv:   ' RESRVED'  ),     {Зарезервовано}
    ( Prt: 7; Srv:   ' ECHO   '  ),     {Ping    }
    ( Prt: 9; Srv:   ' DISCARD'  ),
    ( Prt: 13; Srv:  ' DAYTIME'  ),
    ( Prt: 17; Srv:  ' QOTD   '  ),     {Показчик на день}
    ( Prt: 19; Srv:  ' CHARGEN'  ),     {Генератор символів}
    ( Prt: 20; Srv:  ' FTPDATA'  ),     { File Transfer Protocol- данні}
    ( Prt: 21; Srv:  ' FTPCTRL'  ),     { File Transfer Protocol- управління}
    ( Prt: 22; Srv:  ' SSH    '  ),
    ( Prt: 23; Srv:  ' TELNET '  ),
    ( Prt: 25; Srv:  ' SMTP   '  ),     { Simple Mail Transfer Protocol}
    ( Prt: 37; Srv:  ' TIME   '  ),     { Часовий протокол }
    ( Prt: 43; Srv:  ' WHOIS  '  ),     { Сервіс – Кто це  }
    ( Prt: 53; Srv:  ' DNS    '  ),     { Domain Name Service }
    ( Prt: 67; Srv:  ' BOOTPS '  ),     { BOOTP Сервер }
    ( Prt: 68; Srv:  ' BOOTPC '  ),     { BOOTP Кієнт }
    ( Prt: 69; Srv:  ' TFTP   '  ),     { стандартний  FTP  }
    ( Prt: 70; Srv:  ' GOPHER '  ),     { Протокол Gopher       }
    ( Prt: 79; Srv:  ' FINGER '  ),     { Протокол Finger       }
    ( Prt: 80; Srv:  ' HTTP   '  ),     { Протокол HTTP         }
    ( Prt: 88; Srv:  ' KERBROS'  ),     { Протокол Kerberos     }
    ( Prt: 109; Srv: ' POP2   '  ),     { Протокол Post Office Protocol Version
2 }
    ( Prt: 110; Srv: ' POP3   '  ),     { Протокол Post Office Protocol Version
3 }
    ( Prt: 111; Srv: ' SUN_RPC'  ),     { Протокол SUN Remote Procedure Call }
    ( Prt: 119; Srv: ' NNTP   '  ),     { Протокол Network News Transfer
Protocol }
    ( Prt: 123; Srv: ' NTP    '  ),     { Протокол Network Time protocol
}
    ( Prt: 135; Srv: ' DCOMRPC'  ),     { Протокол Location Service
}
    ( Prt: 137; Srv: ' NBNAME '  ),     { NETBIOS сервіс імен          }
    ( Prt: 138; Srv: ' NBDGRAM'  ),     { NETBIOS сервіс датаграм     }
    ( Prt: 139; Srv: ' NBSESS '  ),     { NETBIOS сервіс сесій         }
    ( Prt: 143; Srv: ' IMAP   '  ),     { Протокол Internet Message Access
Protocol }
    ( Prt: 161; Srv: ' SNMP   '  ),     { Протокол Simple Netw. Management
Protocol }
    ( Prt: 169; Srv: ' SEND   '  )
```

```
    );


//------перетворення ICMP кодів помилок до рядків-------------

const
  ICMP_ERROR_BASE = 11000;
  IcmpErr : array[1..22] of string =
  (
   ' IP_BUFFER_TOO_SMALL',' IP_DEST_NET_UNREACHABLE' , '
IP_DEST_HOST_UNREACHABLE' ,
   ' IP_PROTOCOL_UNREACHABLE' , ' IP_DEST_PORT_UNREACHABLE' , ' IP_NO_RESOURCES'
,
   ' IP_BAD_OPTION',' IP_HARDWARE_ПОМИЛКА' , ' IP_PACKET_TOO_BIG' , '
IP_REQUEST_TIMED_OUT' ,
   ' IP_BAD_REQUEST' ,' IP_BAD_ROUTE' , ' IP_TTL_EXPIRED_TRANSIT' ,
   ' IP_TTL_EXPIRED_REASSEM' ,' IP_PARAMETER_PROBLEM' , ' IP_SOURCE_QUENCH' ,
   ' IP_OPTION_TOO_BIG' , ' IP_BAD_DESTINATION' ,' IP_ADDRESS_DELETED' ,
   ' IP_SPEC_MTU_CHANGE' , ' IP_MTU_CHANGE' , ' IP_UNLOAD'
  );


//-----Перетворення різних перерахованих величин у рядки--------

  ARPEntryType  : array[1..4] of string = ( ' інший' , ' неправильний' ,
    ' динамічний' , ' статичний'
    );
  TCPConnState  :
    array[1..12] of string =
    ( ' closed' , ' listening' , ' syn_sent' ,
    ' syn_rcvd' , ' established' , ' fin_wait1' ,
    ' fin_wait2' , ' close_wait' , ' closing' ,
    ' last_ack' , ' time_wait' , ' delete_tcb'
    );

  TCPToAlgo     : array[1..4] of string =
    ( ' Const.Timeout' , '  MIL-STD-1778' ,
    ' Van Jacobson' , ' інший' );

  IPForwTypes   : array[1..4] of string =
    ( ' інший' , ' invalid' , ' local' , ' remote'  );

  IPForwProtos  : array[1..18] of string =
    ( ' інший' , ' LOCAL' , ' NETMGMT' , ' ICMP' , ' EGP' ,
    ' GGP' , ' HELO' , ' RIP' , ' IS_IS' , ' ES_IS' ,
    ' CISCO' , ' BBN' , ' OSPF' , ' BGP' , ' BOOTP' ,
    ' AUTO_STAT' , ' STATIC' , ' NOT_DOD'  );

type

// для IpHlpNetworkParams
  TNetworkParams = record
    HostName: string ;
    DomainName: string ;
    CurrentDnsServer: string ;
    DnsServerTot: integer ;
    DnsServerNames: array [0..9] of string ;
    NodeType: UINT;
    ScopeID: string ;
    EnableRouting: UINT;
    EnableProxy: UINT;
    EnableDNS: UINT;
  end;

  TIfRows = array of TMibIfRow ; // динамічний масив колонок

// для IpHlpAdaptersInfo
  TAdaptorInfo = record
    AdapterName: string ;
```

```
    Description: string ;
    MacAddress: string ;
    Index: DWORD;
    aType: UINT;
    DHCPEnabled: UINT;
    CurrIPAddress: string ;
    CurrIPMask: string ;
    IPAddressTot: integer ;
    IPAddressList: array of string ;
    IPMaskList: array of string ;
    GatewayTot: integer ;
    GatewayList: array of string ;
    DHCPTot: integer ;
    DHCPServer: array of string ;
    HaveWINS: BOOL;
    PrimWINSTot: integer ;
    PrimWINSServer: array of string ;
    SecWINSTot: integer ;
    SecWINSServer: array of string ;
    LeaseObtained: LongInt ; // UNIX час, секунди з 1970
    LeaseExpires: LongInt;   // UNIX час, секунди з 1970
  end ;

  TAdaptorRows = array of TAdaptorInfo ;


//--------експортуємі данні---------------------

function IpHlpAdaptersInfo(var AdpTot: integer;var AdpRows: TAdaptorRows):
integer ;
procedure Get_AdaptersInfo( List: TStrings );
function IpHlpNetworkParams (var NetworkParams: TNetworkParams): integer ;
procedure Get_NetworkParams( List: TStrings );
procedure Get_ARPTable( List: TStrings );
procedure Get_TCPTable( List: TStrings );
procedure Get_TCPStatistics( List: TStrings );
function IpHlpTCPStatistics (var TCPStats: TMibTCPStats): integer ;
procedure Get_UDPTable( List: TStrings );
procedure Get_UDPStatistics( List: TStrings );
function IpHlpUdpStatistics (UdpStats: TMibUDPStats): integer ;
procedure Get_IPAddrTable( List: TStrings );
procedure Get_IPForwardTable( List: TStrings );
procedure Get_IPStatistics( List: TStrings );
function IpHlpIPStatistics (var IPStats: TMibIPStats): integer ;
function Get_RTTAndHopCount( IPAddr: DWORD; MaxHops: Longint;
  var RTT: longint; var HopCount: longint ): integer;
procedure Get_ICMPStats( ICMPIn, ICMPOut: TStrings );
function IpHlpIfTable(var IfTot: integer; var IfRows: TIfRows): integer ;
procedure Get_IfTable( List: TStrings );
function IpHlpIfEntry(Index: integer; var IfRow: TMibIfRow): integer ;
procedure Get_RecentDestIPs( List: TStrings );

// функції перетворення
function MacAddr2Str( MacAddr: TMacAddress; size: integer ): string;
function IpAddr2Str( IPAddr: DWORD ): string;
function Str2IpAddr( IPStr: string ): DWORD;
function Port2Str( nwoPort: DWORD ): string;
function Port2Wrd( nwoPort: DWORD ): DWORD;
function Port2Svc( Port: DWORD ): string;
function ICMPErr2Str( ICMPErrCode: DWORD) : string;

implementation

var
  RecentIPs     : TStringList;

//-------Основні функції----------------------

{ отримання наступного "токена" з рядка }
```

```
function NextToken( var s: string; Separator: char ): string;
var
  Sep_Pos       : byte;
begin
  Result := ' ' ;
  if length( s ) > 0 then begin
    Sep_Pos := pos( Separator, s );
    if Sep_Pos > 0 then begin
      Result := copy( s, 1, Pred( Sep_Pos ) );
      Delete( s, 1, Sep_Pos );
    end
    else begin
      Result := s;
      s := ' ' ;
    end;
  end;
end;

//-----------------------------------
{ перетворення числового MAC-адреса до ww-xx-yy-zz рядка }
function MacAddr2Str( MacAddr: TMacAddress; size: integer ): string;
var
  i             : integer;
begin
  if Size = 0 then
  begin
    Result := '  00-00-00' ;
    EXIT;
  end
  else Result := ' ' ;
  //
  for i := 1 to Size do
    Result := Result + IntToHex( MacAddr[i], 2) + ' -' ;
  Delete( Result, Length( Result ), 1 );
end;

//-----------------------------------
{ перетворення IP-адреси в мережний байт типу DWORD }
function IpAddr2Str( IPAddr: DWORD ): string;
var
  i             : integer;
begin
  Result := ' ' ;
  for i := 1 to 4 do
  begin
    Result := Result + Format( ' %3d.' , [IPAddr and $FF] );
    IPAddr := IPAddr shr 8;
  end;
  Delete( Result, Length( Result ), 1 );
end;

//-----------------------------------
{ перетворення крапкової десяткової IP-адреси в мережний байт типу DWORD}
function Str2IpAddr( IPStr: string ): DWORD;
var
  i             : integer;
  Num           : DWORD;
begin
  Result := 0;
  for i := 1 to 4 do
  try
    Num := ( StrToInt( NextToken( IPStr, ' .'  ) ) ) shl 24;
    Result := ( Result shr 8 ) or Num;
  except
    Result := 0;
  end;

end;
```

```
//-----------------------------------
{ перетворення номеру порту в мережний байт типу DWORD }
function Port2Wrd( nwoPort: DWORD ): DWORD;
begin
  Result := Swap( WORD( nwoPort ) );
end;

//-----------------------------------
{ перетворення номеру порту в мережний байт типу string }
function Port2Str( nwoPort: DWORD ): string;
begin
  Result := IntToStr( Port2Wrd( nwoPort ) );
end;

//-----------------------------------
{ перетворення номеру порту в сервіс ID }
function Port2Svc( Port: DWORD ): string;
var
  i             : integer;
begin
  Result := Format( ' %4d' , [Port] ); // у випадку, якщо порт не знайдено
  for i := Low( WellKnownPorts ) to High( WellKnownPorts ) do
    if Port = WellKnownPorts[i].Prt then
    begin
      Result := WellKnownPorts[i].Srv;
      BREAK;
    end;
end;

//-----------------------------------
{ голона частина, фіксація мережний параметрів }

procedure Get_NetworkParams( List: TStrings );
var
    NetworkParams: TNetworkParams ;
    I, ErrorCode: integer ;
begin
    if not Assigned( List ) then EXIT;
    List.Clear;
    ErrorCode := IpHlpNetworkParams (NetworkParams) ;
    if ErrorCode <> 0 then
    begin
        List.Add (SysErrorMessage (ErrorCode));
        exit;
    end ;
    with NetworkParams do
    begin
        List.Add( ' Ім'я хосту          : '  + HostName );
        List.Add( ' Домен            : '  + DomainName );
        List.Add( ' NETBIOS тип : '  + NETBIOSTypes[NodeType] );
        List.Add( ' DHCP область        : '  + ScopeID );
        List.Add( ' ROUTING визначено   : '  + IntToStr( EnableRouting ) );
        List.Add( ' PROXY   визначено   : '  + IntToStr( EnableProxy ) );
        List.Add( ' DNS     визначено   : '  + IntToStr( EnableDNS ) );
        if DnsServerTot <> 0 then
        begin
            for I := 0 to Pred (DnsServerTot) do
                List.Add( ' DNS адреса серверу   : '  + DnsServerNames [I] ) ;
        end ;
    end ;
end ;

//-------------------------------------//
function IpHlpNetworkParams (var NetworkParams: TNetworkParams): integer ;
var
  FixedInfo     : PTFixedInfo;         // данні
  InfoSize      : Longint;
  PDnsServer    : PTIP_ADDR_STRING ;   // данні
begin
```

```
    InfoSize := 0 ;   // данні
    result := ERROR_NOT_SUPPORTED ;
    if NOT LoadIpHlp then exit ;
    result := GetNetworkParams( Nil, @InfoSize );  // данні
    if result <> ERROR_BUFFER_OVERFLOW then exit ; // данні
    GetMem (FixedInfo, InfoSize) ;                    // данні
    try
    result := GetNetworkParams( FixedInfo, @InfoSize );   // данні
    if result <> ERROR_SUCCESS then exit ;
    NetworkParams.DnsServerTot := 0 ;
    with FixedInfo^ do
    begin
        NetworkParams.HostName := trim (HostName) ;
        NetworkParams.DomainName := trim (DomainName) ;
        NetworkParams.ScopeId := trim (ScopeID) ;
        NetworkParams.NodeType := NodeType ;
        NetworkParams.EnableRouting := EnableRouting ;
        NetworkParams.EnableProxy := EnableProxy ;
        NetworkParams.EnableDNS := EnableDNS ;
        NetworkParams.DnsServerNames [0] := DNSServerList.IPAddress ;  // данні
        if NetworkParams.DnsServerNames [0] <> ' '  then
                                    NetworkParams.DnsServerTot := 1 ;
        PDnsServer := DnsServerList.Next;
        while PDnsServer <> Nil do
        begin
            NetworkParams.DnsServerNames [NetworkParams.DnsServerTot] :=
                                    PDnsServer^.IPAddress ;  // данні
            inc (NetworkParams.DnsServerTot) ;
            if NetworkParams.DnsServerTot >=
                          Length (NetworkParams.DnsServerNames) then exit ;
            PDnsServer := PDnsServer.Next ;
        end;
    end ;
    finally
       FreeMem (FixedInfo) ;                  // данні
    end ;
end;

//-----------------------------------

function ICMPErr2Str( ICMPErrCode: DWORD) : string;
begin
   Result := ' UnknownError : '  + IntToStr( ICMPErrCode );
   dec( ICMPErrCode, ICMP_ERROR_BASE );
   if ICMPErrCode in [Low(ICMpErr)..High(ICMPErr)] then
     Result := ICMPErr[ ICMPErrCode];
end;


//-----------------------------------

// включення байтів у/з для кожного адаптера

function IpHlpIfTable(var IfTot: integer; var IfRows: TIfRows): integer ;
var
  I,
  TableSize   : integer;
  pBuf, pNext : PChar;
begin
  result := ERROR_NOT_SUPPORTED ;
  if NOT LoadIpHlp then exit ;
  SetLength (IfRows, 0) ;
  IfTot := 0 ; // данні
  TableSize := 0;
   // перший виклик: необхідно отримати розмір пам' яті
  result := GetIfTable (Nil, @TableSize, false) ;  // данні
  if result <> ERROR_INSUFFICIENT_BUFFER then exit ;
  GetMem( pBuf, TableSize );
  try
```

```
      FillChar (pBuf^, TableSize, #0);  // очищаємо буфер з  W98 не беремо
крапку таблиці
      result := GetIfTable (PTMibIfTable (pBuf), @TableSize, false) ;
      if result <> NO_ERROR then exit ;
      IfTot := PTMibIfTable (pBuf)^.dwNumEntries ;
      if IfTot = 0 then exit ;
      SetLength (IfRows, IfTot) ;
      pNext := pBuf + SizeOf(IfTot) ;
      for i := 0 to Pred (IfTot) do
      begin
         IfRows [i] := PTMibIfRow (pNext )^ ;
         inc (pNext, SizeOf (TMibIfRow)) ;
      end;
  finally
      FreeMem (pBuf) ;
  end ;
end;

procedure Get_IfTable( List: TStrings );
var
  IfRows        : TIfRows ;
  Error, I      : integer;
  NumEntries    : integer;
  sDescr, sIfName: string ;
begin
  if not Assigned( List ) then EXIT;
  List.Clear;
  SetLength (IfRows, 0) ;
  Error := IpHlpIfTable (NumEntries, IfRows) ;
  if (Error <> 0) then
      List.Add( SysErrorMessage( GetLastError ) )
  else if NumEntries = 0 then
      List.Add( ' данних немає ' )
  else
  begin
      for I := 0 to Pred (NumEntries) do
      begin
          with IfRows [I] do
          begin
             if wszName [1] = #0 then
                 sIfName := ' '
             else
                 sIfName := WideCharToString (@wszName) ;  // конвертуємо Юнікод
до рядка
             sIfName := trim (sIfName) ;
             sDescr := bDescr ;
             sDescr := trim (sDescr);
             List.Add (Format (
               ' %0.8x |%3d | %16s |%8d |%12d |%2d |%2d |%10d |%10d | %-s| %-s'
,
               [dwIndex, dwType, MacAddr2Str( TMacAddress( bPhysAddr ),
               dwPhysAddrLen ), dwMTU, dwSpeed, dwAdminStatus,
               dwOPerStatus, Int64 (dwInOctets), Int64 (dwOutOctets),  //
конвертуємо до 32-біт
               sIfName, sDescr] )  // данні, додані в/з
               );
          end;
      end ;
  end ;
  SetLength (IfRows, 0) ;  // вільна пам' ять
end ;

function IpHlpIfEntry(Index: integer; var IfRow: TMibIfRow): integer ;
begin
  result := ERROR_NOT_SUPPORTED ;
  if NOT LoadIpHlp then exit ;
  FillChar (IfRow, SizeOf (TMibIfRow), #0);  // очищаємо буфер з  W98 не беремо
  IfRow.dwIndex := Index ;
  result := GetIfEntry (@IfRow) ;
```

```
end ;

//-----------------------------------
{ інформація про інстальовані адаптери }

function IpHlpAdaptersInfo(var AdpTot: integer; var AdpRows: TAdaptorRows):
integer ;
var
  BufLen        : DWORD;
  AdapterInfo   : PTIP_ADAPTER_INFO;
  PIpAddr       : PTIP_ADDR_STRING;
  PBuf          : PCHAR ;
  I             : integer ;
begin
  SetLength (AdpRows, 4) ;
  AdpTot := 0 ;
  BufLen := 0 ;
  result := GetAdaptersInfo( Nil, @BufLen );
  if (result <> ERROR_INSUFFICIENT_BUFFER) and (result = NO_ERROR) then exit ;
  GetMem( pBuf, BufLen );
  try
      FillChar (pBuf^, BufLen, #0);  // очищуємо буфер
      result := GetAdaptersInfo( PTIP_ADAPTER_INFO (PBuf), @BufLen );
      if result = NO_ERROR then
      begin
         AdapterInfo := PTIP_ADAPTER_INFO (PBuf) ;
         while ( AdapterInfo <> nil ) do
         begin
            AdpRows [AdpTot].IPAddressTot := 0 ;
            SetLength (AdpRows [AdpTot].IPAddressList, 2) ;
            SetLength (AdpRows [AdpTot].IPMaskList, 2) ;
            AdpRows [AdpTot].GatewayTot := 0 ;
            SetLength (AdpRows [AdpTot].GatewayList, 2) ;
            AdpRows [AdpTot].DHCPTot := 0 ;
            SetLength (AdpRows [AdpTot].DHCPServer, 2) ;
            AdpRows [AdpTot].PrimWINSTot := 0 ;
            SetLength (AdpRows [AdpTot].PrimWINSServer, 2) ;
            AdpRows [AdpTot].SecWINSTot := 0 ;
            SetLength (AdpRows [AdpTot].SecWINSServer, 2) ;
            AdpRows [AdpTot].CurrIPAddress := NULL_IP;
            AdpRows [AdpTot].CurrIPMask := NULL_IP;
            AdpRows [AdpTot].AdapterName := Trim( string(
AdapterInfo^.AdapterName ) );
            AdpRows [AdpTot].Description := Trim( string(
AdapterInfo^.Description ) );
            AdpRows [AdpTot].MacAddress := MacAddr2Str( TMacAddress(
                                AdapterInfo^.Address ),
AdapterInfo^.AddressLength ) ;
            AdpRows [AdpTot].Index := AdapterInfo^.Index ;
            AdpRows [AdpTot].aType := AdapterInfo^.aType ;
            AdpRows [AdpTot].DHCPEnabled := AdapterInfo^.DHCPEnabled ;
            if AdapterInfo^.CurrentIPAddress <> Nil then
            begin
                AdpRows [AdpTot].CurrIPAddress :=
AdapterInfo^.CurrentIPAddress.IpAddress ;
                AdpRows [AdpTot].CurrIPMask :=
AdapterInfo^.CurrentIPAddress.IpMask ;
            end ;

        // беремо список IP адрес та конвертуємо в IPAddressList
            I := 0 ;
            PIpAddr := @AdapterInfo^.IPAddressList ;
            while (PIpAddr <> Nil) do
            begin
                AdpRows [AdpTot].IPAddressList [I] := PIpAddr.IpAddress ;
                AdpRows [AdpTot].IPMaskList [I] := PIpAddr.IpMask ;
                PIpAddr := PIpAddr.Next ;
                inc (I) ;
                if Length (AdpRows [AdpTot].IPAddressList) <= I then
```

```
            begin
                 SetLength (AdpRows [AdpTot].IPAddressList, I -2) ;
                 SetLength (AdpRows [AdpTot].IPMaskList, I -2) ;
            end ;
        end ;
        AdpRows [AdpTot].IPAddressTot := I ;

    // беремо список IP адрес для GatewayList
        I := 0 ;
        PIpAddr := @AdapterInfo^.GatewayList ;
        while (PIpAddr <> Nil) do
        begin
            AdpRows [AdpTot].GatewayList [I] := PIpAddr.IpAddress ;
            PIpAddr := PIpAddr.Next ;
            inc (I) ;
            if Length (AdpRows [AdpTot].GatewayList) <= I then
                        SetLength (AdpRows [AdpTot].GatewayList, I -2) ;
        end ;
        AdpRows [AdpTot].GatewayTot := I ;

    // беремо список IP адрес для  GatewayList
        I := 0 ;
        PIpAddr := @AdapterInfo^.DHCPServer ;
        while (PIpAddr <> Nil) do
        begin
            AdpRows [AdpTot].DHCPServer [I] := PIpAddr.IpAddress ;
            PIpAddr := PIpAddr.Next ;
            inc (I) ;
            if Length (AdpRows [AdpTot].DHCPServer) <= I then
                        SetLength (AdpRows [AdpTot].DHCPServer, I -2) ;
        end ;
        AdpRows [AdpTot].DHCPTot := I ;

    // беремо список IP адрес для PrimaryWINSServer
        I := 0 ;
        PIpAddr := @AdapterInfo^.PrimaryWINSServer ;
        while (PIpAddr <> Nil) do
        begin
            AdpRows [AdpTot].PrimWINSServer [I] := PIpAddr.IpAddress ;
            PIpAddr := PIpAddr.Next ;
            inc (I) ;
            if Length (AdpRows [AdpTot].PrimWINSServer) <= I then
                        SetLength (AdpRows [AdpTot].PrimWINSServer, I -2) ;
        end ;
        AdpRows [AdpTot].PrimWINSTot := I ;

    // беремо список IP адрес для SecondaryWINSServer
        I := 0 ;
        PIpAddr := @AdapterInfo^.SecondaryWINSServer ;
        while (PIpAddr <> Nil) do
        begin
            AdpRows [AdpTot].SecWINSServer [I] := PIpAddr.IpAddress ;
            PIpAddr := PIpAddr.Next ;
            inc (I) ;
            if Length (AdpRows [AdpTot].SecWINSServer) <= I then
                        SetLength (AdpRows [AdpTot].SecWINSServer, I -2) ;
        end ;
        AdpRows [AdpTot].SecWINSTot := I ;

        AdpRows [AdpTot].LeaseObtained := AdapterInfo^.LeaseObtained ;
        AdpRows [AdpTot].LeaseExpires := AdapterInfo^.LeaseExpires ;

        inc (AdpTot) ;
        if Length (AdpRows) <= AdpTot then
                        SetLength (AdpRows, AdpTot -2) ;  // більше пам' яті
        AdapterInfo := AdapterInfo^.Next;
    end ;
    SetLength (AdpRows, AdpTot) ;
end ;
```

```
  finally
      FreeMem( pBuf );
  end ;
end ;

procedure Get_AdaptersInfo( List: TStrings );
var
  AdpTot: integer;
  AdpRows: TAdaptorRows ;
  Error: DWORD ;
  I: integer ;
  //J: integer ;
  //S: string ;        id.
begin
  if not Assigned( List ) then EXIT;
  List.Clear;
  SetLength (AdpRows, 0) ;
  AdpTot := 0 ;
  Error := IpHlpAdaptersInfo(AdpTot, AdpRows) ;
  if (Error <> 0) then
      List.Add( SysErrorMessage( GetLastError ) )
  else if AdpTot = 0 then
      List.Add( ' данних немає ’  )
  else
  begin
      for I := 0 to Pred (AdpTot) do
      begin
        with AdpRows [I] do
        begin
            //List.Add(AdapterName + ' |’  + Description ); // jpt : не
використовується
            List.Add( Format(' %8.8x | %6s | %16s | %2d | %16s | %16s | %16s’ ,
                [Index, AdaptTypes[aType], MacAddress, DHCPEnabled,
                GatewayList [0], DHCPServer [0], PrimWINSServer [0]] ) );
            {if IPAddressTot <> 0 then     // jpt : не використовується
            begin
                S := ' '  ;
                for J := 0 to Pred (IPAddressTot) do
                        S := S + IPAddressList [J] + ' /’  + IPMaskList [J] + '
| '  ;
                List.Add(IntToStr (IPAddressTot) + '  IP Addresse(s): '  + S);
            end ;
            List.Add( '  ' ); }
        end ;
      end ;
  end ;
  SetLength (AdpRows, 0) ;
end ;

//-----------------------------------
{ моні торимо час доступу до IP }
function Get_RTTAndHopCount( IPAddr: DWORD; MaxHops: Longint; var RTT: Longint;
  var HopCount: Longint ): integer;
begin
  if not GetRTTAndHopCount( IPAddr, @HopCount, MaxHops, @RTT ) then
  begin
    Result := GetLastError;
    RTT :=-1; // Расположення BAD_HOST_NAME,etc...
    HopCount :=-1;
  end
  else
    Result := NO_ERROR;
end;

//-----------------------------------
{ ARP-таблиця включає відношення між віддаленим IP та віддаленим MAC-адресом.
}
procedure Get_ARPTable( List: TStrings );
var
```

```
  IPNetRow      : TMibIPNetRow;
  TableSize     : DWORD;
  NumEntries    : DWORD;
  ErrorCode     : DWORD;
  i             : integer;
  pBuf          : PChar;
begin
  if not Assigned( List ) then EXIT;
  List.Clear;
  // перший виклик: беремо довжину таблиці
  TableSize := 0;
  ErrorCode := GetIPNetTable( Nil, @TableSize, false );   // данні
  //
  if ErrorCode = ERROR_NO_DATA then
  begin
    List.Add( '  ARP-кеш пустий.'  );
    EXIT;
  end;
  // беремо таблицю
  GetMem( pBuf, TableSize );
  NumEntries := 0 ;
  try
  ErrorCode := GetIpNetTable( PTMIBIPNetTable( pBuf ), @TableSize, false );
  if ErrorCode = NO_ERROR then
  begin
    NumEntries := PTMIBIPNetTable( pBuf )^.dwNumEntries;
    if NumEntries > 0 then
    begin
      inc( pBuf, SizeOf( DWORD ) ); // бермо розмір останьої таблиці
      for i := 1 to NumEntries do
      begin
        IPNetRow := PTMIBIPNetRow( PBuf )^;
        with IPNetRow do
          List.Add( Format( ' %8x | %12s | %16s | %10s' ,
                            [dwIndex, MacAddr2Str( bPhysAddr, dwPhysAddrLen ),
                            IPAddr2Str( dwAddr ), ARPEntryType[dwType]
                            ]));
        inc( pBuf, SizeOf( IPNetRow ) );
      end;
    end
    else
      List.Add( '  ARP-кеш пустий.'  );
  end
  else
    List.Add( SysErrorMessage( ErrorCode ) );

  // необхідно відновити показник!
  finally
      dec( pBuf, SizeOf( DWORD ) + NumEntries -SizeOf( IPNetRow ) );
      FreeMem( pBuf );
  end ;
end;


//---------------------------------
procedure Get_TCPTable( List: TStrings );
var
  TCPRow        : TMIBTCPRow;
  i,
    NumEntries  : integer;
  TableSize     : DWORD;
  ErrorCode     : DWORD;
  DestIP        : string;
  pBuf          : PChar;
begin
  if not Assigned( List ) then EXIT;
  List.Clear;
  RecentIPs.Clear;
  // перший виклик: беремо довжину таблиці
```

```
  TableSize := 0;
  NumEntries := 0 ;
  ErrorCode := GetTCPTable(Nil, @TableSize, false );  // данні
  if Errorcode <> ERROR_INSUFFICIENT_BUFFER then
    EXIT;

  // беремо розмір пам’ яті, викликаємо знову
  GetMem( pBuf, TableSize );
  // беремо таблицю
  ErrorCode := GetTCPTable( PTMIBTCPTable( pBuf ), @TableSize, false );
  if ErrorCode = NO_ERROR then
  begin

    NumEntries := PTMIBTCPTable( pBuf )^.dwNumEntries;
    if NumEntries > 0 then
    begin
      inc( pBuf, SizeOf( DWORD ) ); // беремо розмір останьої таблиці
      for i := 1 to NumEntries do
      begin
        TCPRow := PTMIBTCPRow( pBuf )^; // беремо останій запис
        with TCPRow do
        begin
          if dwRemoteAddr = 0 then
            dwRemotePort := 0;
          DestIP := IPAddr2Str( dwRemoteAddr );
          List.Add(
            Format( ‘ %15s : %-7s | %15s : %-7s | %-16s’ ,
            [IpAddr2Str( dwLocalAddr ),
            Port2Svc( Port2Wrd( dwLocalPort ) ),
              DestIP,
              Port2Svc( Port2Wrd( dwRemotePort ) ),
              TCPConnState[dwState]
              ] ) );
         //
            if (not ( dwRemoteAddr = 0 ))
            and ( RecentIps.IndexOf(DestIP) =-1 ) then
               RecentIPs.Add( DestIP );
        end;
        inc( pBuf, SizeOf( TMIBTCPRow ) );
      end;
    end;
  end
  else
    List.Add( SyserrorMessage( ErrorCode ) );
  dec( pBuf, SizeOf( DWORD ) + NumEntries -SizeOf( TMibTCPRow ) );
  FreeMem( pBuf );
end;

//-------------------------------
procedure Get_TCPStatistics( List: TStrings );
var
  TCPStats      : TMibTCPStats;
  ErrorCode     : DWORD;
begin
  if not Assigned( List ) then EXIT;
  List.Clear;
  if NOT LoadIpHlp then exit ;
  ErrorCode := GetTCPStatistics( @TCPStats );
  if ErrorCode = NO_ERROR then
    with TCPStats do
    begin
      List.Add( ‘ Алгоритм повторної передачі : ‘  + TCPToAlgo[dwRTOAlgorithm]
);
      List.Add( ‘ Мінімальний час виходу         : ‘  + IntToStr( dwRTOMin ) + ‘
ms’  );
      List.Add( ‘ Максимальний час виходу         : ‘  + IntToStr( dwRTOMax ) +
‘  ms’  );
      List.Add( ‘ Максимальне число підключень : ‘  + IntToStr( dwRTOAlgorithm )
);
```

```
      List.Add( ' Активні підключення             : '  + IntToStr( dwActiveOpens
) );
      List.Add( ' пасивні підключення            : '  + IntToStr( dwPassiveOpens
) );
      List.Add( ' Невдала спроба відкриття     : '  + IntToStr( dwAttemptFails )
);
      List.Add( ' Скидання встановленого підключення  : '  + IntToStr(
dwEstabResets ) );
      List.Add( ' Поточне встановлене підключення.: '  + IntToStr( dwCurrEstab )
);
      List.Add( ' Отримані сегменти        : '  + IntToStr( dwInSegs ) );
      List.Add( ' Передані сегменти            : '  + IntToStr( dwOutSegs ) );
      List.Add( ' Перепідключені сегменти   : '  + IntToStr( dwReTransSegs ) );
      List.Add( ' помилка входу          : '  + IntToStr( dwInErrs ) );
      List.Add( ' Перезавантаження вихідних          : '  + IntToStr( dwOutRsts
) );
      List.Add( ' Совокупні зв' язки    : '  + IntToStr( dwNumConns ) );
    end
  else
    List.Add( SyserrorMessage( ErrorCode ) );
end;

function IpHlpTCPStatistics (var TCPStats: TMibTCPStats): integer ;
begin
    result := ERROR_NOT_SUPPORTED ;
    if NOT LoadIpHlp then exit ;
    result := GetTCPStatistics( @TCPStats );
end;

//-----------------------------------
procedure Get_UDPTable( List: TStrings );
var
  UDPRow        : TMIBUDPRow;
  i,
    NumEntries  : integer;
  TableSize     : DWORD;
  ErrorCode     : DWORD;
  pBuf          : PChar;
begin
  if not Assigned( List ) then EXIT;
  List.Clear;

  // перший виклик: беремо довжину таблиці
  TableSize := 0;
  NumEntries := 0 ;
  ErrorCode := GetUDPTable(Nil, @TableSize, false );
  if Errorcode <> ERROR_INSUFFICIENT_BUFFER then
    EXIT;

  // виділяємо пам' ять, викликаємо знову
  GetMem( pBuf, TableSize );

  // беремо таблицю
  ErrorCode := GetUDPTable( PTMIBUDPTable( pBuf ), @TableSize, false );
  if ErrorCode = NO_ERROR then
  begin
    NumEntries := PTMIBUDPTable( pBuf )^.dwNumEntries;
    if NumEntries > 0 then
    begin
      inc( pBuf, SizeOf( DWORD ) ); // беремо розмір останьої таблиці
      for i := 1 to NumEntries do
      begin
        UDPRow := PTMIBUDPRow( pBuf )^; // беремо останій запис
        with UDPRow do
          List.Add( Format( ' %15s : %-6s' ,
            [IpAddr2Str( dwLocalAddr ),
            Port2Svc( Port2Wrd( dwLocalPort ) )
              ] ) );
        inc( pBuf, SizeOf( TMIBUDPRow ) );
```

```
      end;
    end
    else
      List.Add( ' немає даних.'  );
  end
  else
    List.Add( SyserrorMessage( ErrorCode ) );
  dec( pBuf, SizeOf( DWORD ) + NumEntries -SizeOf( TMibUDPRow ) );
  FreeMem( pBuf );
end;

//-----------------------------------
procedure Get_IPAddrTable( List: TStrings );
var
  IPAddrRow     : TMibIPAddrRow;
  TableSize     : DWORD;
  ErrorCode     : DWORD;
  i             : integer;
  pBuf          : PChar;
  NumEntries    : DWORD;
begin
  if not Assigned( List ) then EXIT;
  List.Clear;
  TableSize := 0; ;
  NumEntries := 0 ;
  // перший виклик: беремо довжину таблиці
  ErrorCode := GetIpAddrTable(Nil, @TableSize, true );  // данні
  if Errorcode <> ERROR_INSUFFICIENT_BUFFER then
    EXIT;

  GetMem( pBuf, TableSize );
  // беремо таблицю
  ErrorCode := GetIpAddrTable( PTMibIPAddrTable( pBuf ), @TableSize, true );
  if ErrorCode = NO_ERROR then
  begin
    NumEntries := PTMibIPAddrTable( pBuf )^.dwNumEntries;
    if NumEntries > 0 then
    begin
      inc( pBuf, SizeOf( DWORD ) );
      for i := 1 to NumEntries do
      begin
        IPAddrRow := PTMIBIPAddrRow( pBuf )^;
        with IPAddrRow do
          List.Add( Format( ' %8.8x | %15s | %15s | %15s | %8.8d' ,
            [dwIndex,
            IPAddr2Str( dwAddr ),
             IPAddr2Str( dwMask ),
             IPAddr2Str( dwBCastAddr ),
             dwReasmSize
             ] ) );
        inc( pBuf, SizeOf( TMIBIPAddrRow ) );
      end;
    end
    else
      List.Add( ' немає даних.'  );
  end
  else
    List.Add( SysErrorMessage( ErrorCode ) );

  // відновлюємо показчик!
  dec( pBuf, SizeOf( DWORD ) + NumEntries -SizeOf( IPAddrRow ) );
  FreeMem( pBuf );
end;

//-----------------------------------
{ отримуємо дані з таблиці маршрутизації; }
procedure Get_IPForwardTable( List: TStrings );
var
  IPForwRow     : TMibIPForwardRow;
```

```
  TableSize     : DWORD;
  ErrorCode     : DWORD;
  i             : integer;
  pBuf          : PChar;
  NumEntries    : DWORD;
begin

  if not Assigned( List ) then EXIT;
  List.Clear;
  TableSize := 0;

  // перший виклик: беремо довжину таблиці
  NumEntries := 0 ;
  ErrorCode := GetIpForwardTable(Nil, @TableSize, true);
  if Errorcode <> ERROR_INSUFFICIENT_BUFFER then
    EXIT;

  // беремо таблицю
  GetMem( pBuf, TableSize );
  ErrorCode := GetIpForwardTable( PTMibIPForwardTable( pBuf ), @TableSize,
true);
  if ErrorCode = NO_ERROR then
  begin
    NumEntries := PTMibIPForwardTable( pBuf )^.dwNumEntries;
    if NumEntries > 0 then
    begin
      inc( pBuf, SizeOf( DWORD ) );
      for i := 1 to NumEntries do
      begin
        IPForwRow := PTMibIPForwardRow( pBuf )^;
        with IPForwRow do
        begin
          if (dwForwardType < 1)
          or (dwForwardType > 4) then
                  dwForwardType := 1 ;   // данні
          List.Add( Format(
            ' %15s | %15s | %15s | %8.8x | %7s |   %5.5d |  %7s |   %2.2d' ,
            [IPAddr2Str( dwForwardDest ),
            IPAddr2Str( dwForwardMask ),
              IPAddr2Str( dwForwardNextHop ),
              dwForwardIFIndex,
              IPForwTypes[dwForwardType],
              dwForwardNextHopAS,
              IPForwProtos[dwForwardProto],
              dwForwardMetric1
              ] ) );
        end ;
        inc( pBuf, SizeOf( TMibIPForwardRow ) );
      end;
    end
    else
      List.Add( ' немає даних.'  );
  end
  else
    List.Add( SysErrorMessage( ErrorCode ) );
  dec( pBuf, SizeOf( DWORD ) + NumEntries -SizeOf( TMibIPForwardRow ) );
  FreeMem( pBuf );
end;

//----------------------------------
procedure Get_IPStatistics( List: TStrings );
var
  IPStats       : TMibIPStats;
  ErrorCode     : integer;
begin
  if not Assigned( List ) then EXIT;
  if NOT LoadIpHlp then exit ;
  ErrorCode := GetIPStatistics( @IPStats );
  if ErrorCode = NO_ERROR then
```

```
  begin
    List.Clear;
    with IPStats do
    begin
      if dwForwarding = 1 then
        List.add( ' Розблокована пересилка      : '  + ' так'  )
      else
        List.add(' Розблокована пересилка      : '  + ' ні'  );
      List.add( ' Любий TTL             : '  + inttostr( dwDefaultTTL ) );
      List.add( ' Датаграмма прийнята     : '  + inttostr( dwInReceives ) );
      List.add( ' Помилка заголовку      (In)  : '  + inttostr( dwInHdrErrors )
);
      List.add( ' Помилка адреси    (In)  : '  + inttostr( dwInAddrErrors ) );
      List.add( ' Датаграма переслана     : '  + inttostr( dwForwDatagrams ) );
// данні
      List.add( ' Невизначений протокол (In)  : '  + inttostr( dwInUnknownProtos
) );
      List.add( ' Датаграмма відмовлена     : '  + inttostr( dwInDiscards ) );
      List.add( ' Датаграма встановлена     : '  + inttostr( dwInDelivers ) );
      List.add( ' Зовнішній запрос            : '  + inttostr( dwOutRequests )
);
      List.add( ' Маршрутизація не виконана      : '  + inttostr(
dwRoutingDiscards ) );
      List.add( ' Немає маршрутів        (Out)  : '  + inttostr( dwOutNoRoutes )
);
      List.add( ' Перебраний час     : '  + inttostr( dwReasmTimeOut ) );
      List.add( ' Запит перебору     : '  + inttostr( dwReasmReqds ) );
      List.add( ' Повний перебор : '  + inttostr( dwReasmOKs ) );
      List.add( ' Помилка перебору     : '  + inttostr( dwReasmFails ) );
      List.add( ' Повна фрагментація: '  + inttostr( dwFragOKs ) );
      List.add( ' Помилка фрагментації   : '  + inttostr( dwFragFails ) );
      List.add( ' Датаграмма фрагментована   : '  + inttostr( dwFRagCreates )
);
      List.add( ' Кількість інтерфейсів    : '  + inttostr( dwNumIf ) );
      List.add( ' Кількість IP-адрес  : '  + inttostr( dwNumAddr ) );
      List.add( ' Маршрут в таблиці маршрутизатора  : '  + inttostr( dwNumRoutes
) );
    end;
  end
  else
    List.Add( SysErrorMessage( ErrorCode ) );
end;

function IpHlpIPStatistics (var IPStats: TMibIPStats): integer ;      // данні
begin
    result := ERROR_NOT_SUPPORTED ;
    if NOT LoadIpHlp then exit ;
    result := GetIPStatistics( @IPStats );
end ;

//---------------------------------
procedure Get_UdpStatistics( List: TStrings );
var
  UdpStats      : TMibUDPStats;
  ErrorCode     : integer;
begin
  if not Assigned( List ) then EXIT;
  ErrorCode := GetUDPStatistics( @UdpStats );
  if ErrorCode = NO_ERROR then
  begin
    List.Clear;
    with UDPStats do
    begin
      List.add( ' Датаграми (In)    : '  + inttostr( dwInDatagrams ) );
      List.add( '  Датаграми (Out)   : '  + inttostr( dwOutDatagrams ) );
      List.add( ' Немає портів         : '  + inttostr( dwNoPorts ) );
      List.add( ' Помилка    (In)    : '  + inttostr( dwInErrors ) );
      List.add( ' UDP список портів  : '  + inttostr( dwNumAddrs ) );
    end;
```

```
  end
  else
    List.Add( SysErrorMessage( ErrorCode ) );
end;

//-------------------------------------*//
function IpHlpUdpStatistics (UdpStats: TMibUDPStats): integer ;     // данні
begin
    result := ERROR_NOT_SUPPORTED ;
    if NOT LoadIpHlp then exit ;
    result := GetUDPStatistics (@UdpStats) ;
end ;

//-----------------------------------
procedure Get_ICMPStats( ICMPIn, ICMPOut: TStrings );
var
  ErrorCode     : DWORD;
  ICMPStats     : PTMibICMPInfo;
begin
  if ( ICMPIn = nil ) or ( ICMPOut = nil ) then EXIT;
  ICMPIn.Clear;
  ICMPOut.Clear;
  New( ICMPStats );
  ErrorCode := GetICMPStatistics( ICMPStats );
  if ErrorCode = NO_ERROR then
  begin
    with ICMPStats.InStats do
    begin
      ICMPIn.Add( ' Прийнято повідомлень    : ' + IntToStr( dwMsgs ) );
      ICMPIn.Add( ' Помилка                 : ' + IntToStr( dwErrors ) );
      ICMPIn.Add( ' Розташування недосягенно    : ' + IntToStr( dwDestUnreachs
) );
      ICMPIn.Add( ' Час перевищений         : ' + IntToStr( dwTimeEcxcds ) );
      ICMPIn.Add( ' Проблеми з параметрами      : ' + IntToStr( dwParmProbs )
);
      ICMPIn.Add( ' Джерело відключено        : ' + IntToStr( dwSrcQuenchs ) );
      ICMPIn.Add( ' Переназначено           : ' + IntToStr( dwRedirects ) );
      ICMPIn.Add( ' Ехо запит          : ' + IntToStr( dwEchos ) );
      ICMPIn.Add( ' Ехо відповідь           : ' + IntToStr( dwEchoReps ) );
      ICMPIn.Add( ' Запит мітки часу    : ' + IntToStr( dwTimeStamps ) );
      ICMPIn.Add( ' Відповідь мітки часу    : ' + IntToStr( dwTimeStampReps )
);
      ICMPIn.Add( ' Запит маски адрес : ' + IntToStr( dwAddrMasks ) );
      ICMPIn.Add( ' Відповідь маски адрес   : ' + IntToStr( dwAddrReps ) );
    end;
     //
    with ICMPStats.OutStats do
    begin
      ICMPOut.Add( ' Повідомлення вдправлено        : ' + IntToStr( dwMsgs ) );
      ICMPOut.Add( 'Помилка                 : ' + IntToStr( dwErrors ) );
      ICMPOut.Add( 'Розташування недосягенно    : ' + IntToStr( dwDestUnreachs
) );
      ICMPOut.Add( 'Час перевищений         : ' + IntToStr( dwTimeEcxcds ) );
      ICMPOut.Add( 'Проблеми з параметрами      : ' + IntToStr( dwParmProbs )
);
      ICMPOut.Add( 'Джерело відключено        : ' + IntToStr( dwSrcQuenchs ) );
      ICMPOut.Add( 'Переназначено           : ' + IntToStr( dwRedirects ) );
      ICMPOut.Add( 'Ехо запит          : ' + IntToStr( dwEchos ) );
      ICMPOut.Add( 'Ехо відповідь           : ' + IntToStr( dwEchoReps ) );
      ICMPOut.Add( 'Запит мітки часу    : ' + IntToStr( dwTimeStamps ) );
      ICMPOut.Add( 'Відповідь мітки часу    : ' + IntToStr( dwTimeStampReps )
);
      ICMPOut.Add( 'Запит маски адрес: ' + IntToStr( dwAddrMasks ) );
      ICMPOut.Add( 'Відповідь маски адрес   : ' + IntToStr( dwAddrReps ) );
    end;
  end
  else
    IcmpIn.Add( SysErrorMessage( ErrorCode ) );
  Dispose( ICMPStats );
```

```
end;

//-----------------------------------
procedure Get_RecentDestIPs( List: TStrings );
begin
  if Assigned( List ) then
    List.Assign( RecentIPs )
end;

initialization

  RecentIPs := TStringList.Create;

finalization

  RecentIPs.Free;

end.
```

Файл About.pas- довідка

```
unit About;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls;

type
  TForm1 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Label9: TLabel;
    Button1: TButton;
    Image2: TImage;
    Image1: TImage;
    Image3: TImage;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
begin
Form1.Close;
end;

end.
```