УДК 004

Б.Федоров, магістр гр. КІ-21М-1,4,
Центральноукраїнський національний технічний університет

ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ДЛЯ ПРОТИДІЇ ДЕКОМПІЛЯЦІЇ КОДУ

У статті розроблено програмне забезпечення, яке призначено для системи для протидії декомпіляції коду. Метою розробки є дослідження та програмна реалізація системи для протидії декомпіляції коду. Об'єктом дослідження є процес для протидії декомпіляції коду. Предметом дослідження є методи для протидії декомпіляції коду. Методи дослідження базуються на методах захисту інформації, методах математичної статистики, методах розробки програмного забезпечення. Результат роботи – програмна реалізація системи для протидії декомпіляції коду. В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.
комп'ютерна інженерія, протидія декомпіляції коду

Постановка проблеми.

На сучасному етапі для протидії декомпіляції коду використовується найпоширеніша методика автоматичного захисту програм від аналізу на основі технології віртуалізації машинного коду. Дана методика ставить у відповідність кожній машинній інструкції одну або кілька інструкцій автоматично згенерованого віртуального процесора, називаних байт-кодом або псевдокодом. У тіло програми, що захищається, вбудовується захищений від аналізу інтерпретатор, завданням якого є виконання згенерованого на етапі захисту байта-коду. Основним недоліком такого підходу є низька швидкість роботи захищеного в такий спосіб коду. Більше того, у більшості випадків все-таки можливе створення автоматичного декомпілятора псевдокоду у вихідний машинний код.

У зв'язку із цим для підвищення стійкості до автоматичних засобів деактивації захисту й забезпечення більш високої швидкодії захищеної програми в порівнянні з існуючими методиками необхідне створення нових підходів. В основі їх лежить принцип програмного «чорного ящика», реалізація якого припускає використання технологій заплутування коду й даних програми або обфускації

Дана обставина визначає актуальність розробки методик обфускації коду й даних програми, що не вимагають для своєї роботи вихідного коду програми й забезпечуючих стійкість стосовно автоматичних утиліт деактивації захисту.

Аналіз останніх досліджень і публікацій. При аналізі останніх досліджень і публікацій [1-10] було виявлено певні прогалини у забезпеченні системи для протидії декомпіляції коду.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи для протидії декомпіляції коду.

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

- Огляд існуючих систем для протидії декомпіляції коду.
- Дослідження системи для протидії декомпіляції коду.
- Програмна реалізація системи для протидії декомпіляції коду.

Об'єктом дослідження є процес для протидії декомпіляції коду.

Предметом дослідження є методи для протидії декомпіляції коду.

Методи дослідження базуються на методах захисту інформації, методах математичної статистики, методах розробки програмного забезпечення.

Виклад основного матеріалу

Обфускація коду – це техніка, яку використовують автори зловмисного програмного забезпечення та інші зловмисники, щоб приховати справжні наміри свого коду та уникнути виявлення програмним забезпеченням безпеки.

Обфускація коду – це процес ускладнення розуміння, аналізу та зворотного проектування програмного коду. Це техніка, яка використовується авторами зловмисних програм та іншими зловмисниками, щоб приховати справжні наміри свого коду та уникнути виявлення програмним забезпеченням безпеки.

У цій статті ми розглянемо різні прийоми та методи, які використовуються для обфускації коду, а також те, як це зробити.

Як працює обфускація коду?

Обфускація коду працює шляхом перетворення вихідного коду у форму, яку важко зрозуміти й проаналізувати. Цього можна досягти за допомогою різних методів, включаючи шифрування, упаковку коду та обфускацію потоку керування.

Мета обфускації коду полягає в тому, щоб аналітикам і програмному забезпеченню безпеки було важко зрозуміти функціональність коду, що ускладнює його виявлення та видалення, а також робить код більш стійким до зворотного проектування.

Хоча обфускація може стримувати випадкові спроби зрозуміти код, важливо зазначити, що рішучі та досвідчені зловмисники все одно можуть реконструювати обфускований код, витративши достатньо зусиль і часу.

Обфускація – це лише один із рівнів захисту, і на неї не слід покладатися виключно для захисту критичних аспектів системи.

Методи обфускації коду

Методи обфускації коду передбачають перетворення коду, щоб зробити його більш складним для розуміння, зворотне проектування або підробку. Ось кілька поширених прийомів:

1. Шифрування рядків

Ця техніка передбачає шифрування рядків у коді, щоб аналітикам було важко зрозуміти передбачувані дії коду.

Наприклад, у зразку зловмисного програмного забезпечення, який шифрує рядок «Видалити всі файли», аналітикам буде важко зрозуміти намічені дії зловмисного програмного забезпечення без попереднього розшифрування рядка.

2. Обфускація потоку керування

Ця техніка передбачає використання складних розгалужених і циклічних структур у коді, щоб аналітикам було важко зрозуміти потік керування кодом.

Наприклад, зразок зловмисного програмного забезпечення, який використовує кілька вкладених операторів if-else і кілька операторів goto, ускладнить аналітикам розуміння потоку керування шкідливим програмним забезпеченням.

3. Антиналагодження

Цей метод передбачає використання різних методів для виявлення та запобігання налагодженню коду, що ускладнює аналітикам розуміння поведінки коду.

Наприклад, аналітикам буде важко проаналізувати та зрозуміти зразок зловмисного програмного забезпечення, який перевіряє наявність налагоджувача та завершує роботу, якщо його виявлено.

4. Упаковка коду

Ця техніка передбачає стиснення та шифрування коду, щоб аналітикам було важко видобувати та аналізувати функціональні можливості коду.

Наприклад, зразок зловмисного програмного забезпечення, який використовує пакувальник UPX для стиснення та шифрування коду, аналітикам буде важко витягнути та проаналізувати функціональність коду.

5. Введення коду

Ця техніка передбачає введення коду в законні процеси або системні бібліотеки, щоб аналітикам було важко виявити та ізолювати код.

Наприклад, аналітикам буде важко виявити та виділити код зловмисного програмного забезпечення, яке впроваджується в процес explorer.exe.

6. Поліморфізм

Ця техніка передбачає постійну модифікацію коду, щоб уникнути систем виявлення на основі сигнатур.

Наприклад, системам виявлення на основі сигнатур буде важко виявити зразок шкідливого програмного забезпечення, який генерує новий код під час кожного запуску.

7. Метаморфізм

Ця техніка є більш просунутою формою поліморфізму, яка передбачає використання методів генерації коду для створення кількох версій зловмисного програмного забезпечення, які мають різні коди, але виконують однакові шкідливі дії.

Наприклад, системам виявлення на основі сигнатур буде важко виявити зразок шкідливого програмного забезпечення, який створює новий код для кожного нового зараження.

8. Безфайлове шкідливе програмне забезпечення

Цей метод передбачає використання методів на основі пам’яті для запуску зловмисного програмного забезпечення без запису у файлову систему, що ускладнює аналітикам виявлення та аналіз зловмисного програмного забезпечення.

Наприклад, аналітикам буде важко виявити та проаналізувати зразок шкідливого програмного забезпечення, який повністю працює в пам’яті та не залишає жодних слідів у файловій системі.

9. Перейменування функцій API

Перейменування функцій API – це стратегія, спрямована на те, щоб зробити код зловмисного ПЗ більш стійким до аналізу шляхом зміни імен функцій, які він використовує для взаємодії з операційною системою. Це частина ширшого набору методів, які використовуються для приховування справжньої природи та призначення шкідливого коду.

10. Стиснення коду

Стиснення коду – це техніка, яка використовується в обфускації коду для зменшення розміру виконуваного файлу шляхом стиснення коду. Це може ускладнити аналіз і зворотне проектування коду.

Які приклади обфускації?

Щоб ви зрозуміли, як працює обфускація коду та як автори зловмисного програмного забезпечення використовують такі методи або техніки. Нижче наведено кілька прикладів обфускації коду, які допоможуть вам краще його зрозуміти.

Одним із прикладів обфускації коду є використання шифрування рядків у шкідливих програмах. Цей метод передбачає шифрування рядків у коді зловмисного програмного забезпечення, що ускладнює аналітикам розуміння намічених дій зловмисного програмного забезпечення.

Наприклад, у зразку зловмисного програмного забезпечення, який шифрує рядок «Видалити всі файли», аналітикам буде важко зрозуміти намічені дії зловмисного програмного забезпечення без попереднього розшифрування рядка.

Іншим прикладом є використання методів захисту від помилок у зловмисному програмному забезпеченні. Цей метод передбачає використання різних методів для виявлення та запобігання налагодженню коду зловмисного програмного забезпечення, що ускладнює аналітикам розуміння поведінки зловмисного програмного забезпечення.

Наприклад, аналітикам буде важко проаналізувати та зрозуміти зразок зловмисного програмного забезпечення, який перевіряє наявність налагоджувача та завершує роботу, якщо його виявлено.

Які є деякі інструменти обфускації коду?

Існує декілька комерційних інструментів із відкритим вихідним кодом для обфускації коду, таких як ConfuserEx, Skater.NET Obfuscator і Crypto Obfuscator.

Ці інструменти можна використовувати для шифрування рядків, упаковки коду та

виконання обфускації потоку керування. Важливо зазначити, що хоча ці інструменти можна використовувати для обфускації коду, вони також можуть використовуватися зловмисниками, щоб приховати справжні наміри свого коду.

Обфускація коду – це техніка, яка використовується для ускладнення розуміння, аналізу та зворотного проектування програмного коду. Він використовується авторами зловмисного програмного забезпечення та іншими зловмисниками, щоб приховати справжні наміри свого коду та уникнути виявлення програмним забезпеченням безпеки.

Обфускацію коду можна досягти за допомогою різноманітних технік і методів, включаючи шифрування, упаковку коду та обфускацію потоку керування.

Інструменти обфускації доступні, щоб допомогти розробникам захистити свій код, але важливо розуміти, що ці інструменти також можуть використовуватися зловмисниками, щоб приховати справжні наміри свого коду.

Обфускація може ускладнити розуміння коду, вона не забезпечує справжнього шифрування чи надійної безпеки. Обфускація більше схожа на маскування – вона просто перетворює код, щоб він став менш читабельним, але основна логіка все ще присутня у формі, яку можна переробити.

Якщо ваша головна турбота про безпеку, покладатися лише на обфускацію недостатньо. Фактично, багато експертів з безпеки вважають обфускацію відносно слабкою формою захисту. Рішучі зловмисники, які володіють потрібними інструментами та навичками, все одно можуть перепроектувати обфускований код.

Для більшої безпеки краще використовувати шифрування. Шифрування передбачає перетворення даних таким чином, що лише авторизовані сторони можуть скасувати перетворення. У контексті коду це може включати шифрування конфіденційних частин коду та їх дешифрування під час виконання.

Обфускація коду в React Native

Є кілька причин, чому ви можете захотіти обфускати свій код React Native:

– **Захист інтелектуальної власності:** обфускація вашого коду ускладнює іншим людям викрасти ваші ідеї або скопіювати вашу роботу.

– **Запобігайте зворотному проектуванню:** обфускація може ускладнити зловмисникам зворотне проектування вашої програми та виявлення вразливостей.

– **Покращення безпеки:** обфускований код складніше використовувати, оскільки зловмисникам буде важче зрозуміти, як він працює.

Як обфускати рідний код React

Існує кілька різних способів обфускати рідний код React. Деякі з найпопулярніших методів включають:

– **Метод плагіна:** ви можете використовувати плагін для обфускації свого коду React Native. Доступно кілька різних плагінів, наприклад:

– obfuscator-io-metro-plugin

– @smartface/obfuscator-io-metro-plugin

– react-native-obfuscating-transformer

– **ProGuard:** ProGuard – це безкоштовний інструмент із відкритим кодом, який можна використовувати для обфускації коду Java. Він включений до Android SDK і може використовуватися для обфускації коду React Native, написаного на Java.

– **R8:** R8 – це новіший інструмент, який також включено в Android SDK. Він призначений для заміни ProGuard і пропонує ряд переваг, включаючи покращену продуктивність і підтримку новіших версій Java.

– **Hermes:** Hermes – це механізм JavaScript, розроблений Facebook. Він розроблений, щоб бути більш ефективним і безпечним, ніж механізм JavaScript за замовчуванням у React Native. Hermes містить вбудований обфускатор, який можна використовувати для обфускації коду React Native, написаного на JavaScript.

– **DexGuard:** DexGuard – це комерційний обфускатор, спеціально розроблений для програм Android, який пропонує низку розширених функцій.

– **iXGuard:** iXGuard – це комерційний обфускатор, спеціально розроблений для додатків iOS, який пропонує ряд розширених функцій.

Сьогодні я зосереджуся на методі плагінів для обфускації нашого коду в React Native. Я поясню кожен крок, як ми можемо обфускати наш код за допомогою obfuscator-io-metro-plugin.

Обфускація рідного коду React за допомогою плагіна Metro

Щоб використовувати плагін Metro Obfuscator.io, вам потрібно буде встановити його з npm. Ви можете зробити це, виконавши таку команду:

npm i -D плагін obfuscator-io-metro

Після встановлення плагіна вам потрібно буде додати його до файлу конфігурації Metro. Ви можете зробити це, додавши такий код до свого metro.config.jsфайлу:

```
const jsoMetroPlugin = require ("obfuscator-io-metro-plugin")(
  {
    // для цих опцій шукайте параметри бібліотеки javascript-obfuscator зверху url
    compact: false,
    sourceMap: false, // вихідна карта, згенерована після обфускації, не є корисною правильно тепер використовуйте значення за замовчуванням, тобто false
    controlFlowFlattening: true,
    controlFlowFlatteningThreshold: 1,
    numbersToExpressions: true,
    simplify: true,
    stringArrayShuffle: true,
    splitStrings: true,
    stringArrayThreshold: 1,
  },
  {
    runInDev: false  /* необов’язково */,
    logObfusc atedFiles: правда  /* необов’язкові згенеровані файли будуть розташовані в./.jso */,
  }
);
модуль. exports = {
  transformer: {
    getTransformOptions: async () => ({
      transform: {
        experimentalImportSupport: false,
        inlineRequires: false,
      },
    }),
  },
  ...jsoMetroPlugin,     /*додайте цей рядок у свій попередній модуль після визначеного вище */
 };
```

Як перевірити, чи працює обфускація

Після того, як ви успішно налаштували плагін Obfuscator.io Metro, ви можете перевірити, чи код обфускований чи ні, створивши збірку за допомогою assembleReleaseабо bundleReleaseабо IPA з Xcode і перевіривши код у такому місці:

/. jso

Якщо код у цьому місці обфускований, плагін працює правильно.

Обфускація коду є цінним інструментом, який може допомогти вам захистити вашу інтелектуальну власність, запобігти зворотній інженерії та покращити безпеку вашої програми React Native. Однак важливо ретельно зважити всі «за» і «проти», перш ніж

вирішити, чи варто обфускати код.

Розробка структурної схеми

На рисунку 1 зображена структурна схема програмного забезпечення, яке реалізує процес обфускації коду.

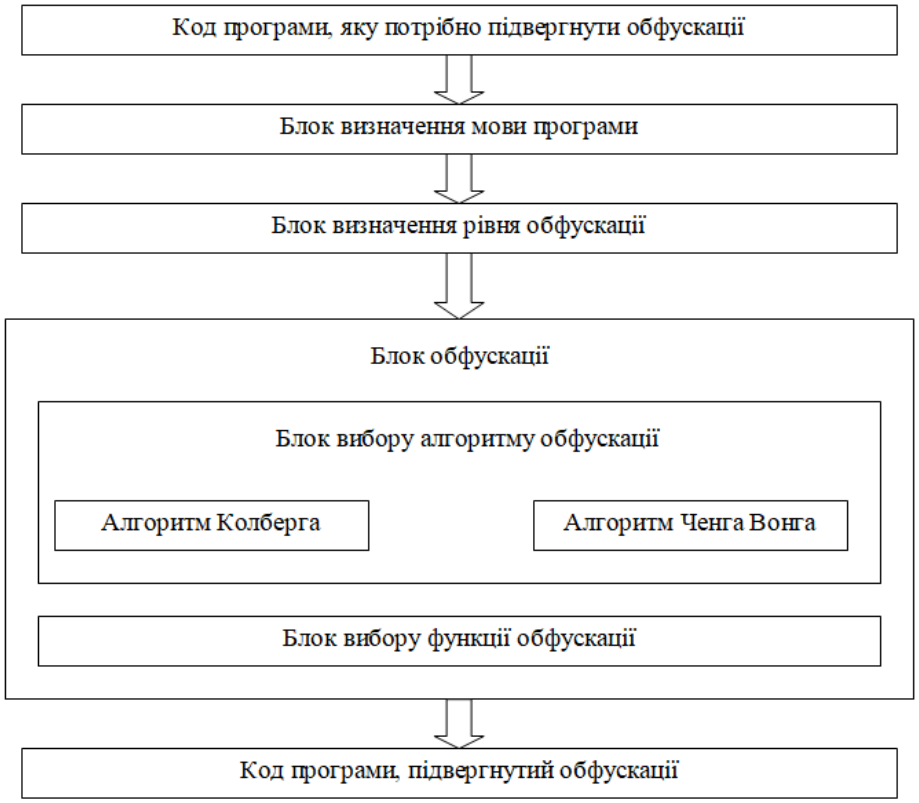

Рисунок 1 – Структурна схема системи

Обфускація коду Java/Kotlin

Обфускацію коду Java/Kotlin надзвичайно легко деобфускати та провести зворотне проектування. По-перше, конфіденційну частину коду ніколи не слід писати цими мовами програмування. Якщо з якоїсь причини це єдиний спосіб дій, тоді найкраще захистити цю чутливу частину коду за допомогою спеціального рішення, а не покладатися лише на обфускацію.

Як найкраща практика для розробки високовартісних програм, слід використовувати мови, які компілюються безпосередньо для складання, наприклад C/C++/Rust тощо.

Конфіденційний код: рівень бізнес-логіки програми, який обробляє/надсилає/отримує конфіденційну інформацію про користувачів або процеси, які мають велике значення для постачальника послуг.

Обфускація – це чудово, але вона приносить побічні ефекти, коли застосовується до всієї програми

Багато разів перетворення, які застосовуються для отримання заплутаного коду, означають, що:

a) Рекламний код буде введено в програму.

b) Буде створено додаткові шари тупикового циклу.

c) Буде введено додаткові символи та значення для перенаправлення коду та багато іншого.

У цих випадках дуже поширеним результатом є те, що програма має тенденцію до збільшення розміру. Це також може означати, що продуктивність програми також знижується, що є червоним прапорцем з точки зору програмування та досвіду користувача. Крім того, існує ймовірність «зламання» або збою програми.

Повноцінні програми, до яких застосовується обфускація, ускладнюють розробнику

виявлення помилок і їх виправлення. Для продовження розробки їм знадобиться певне технологічне рішення відображення, яке допоможе поєднати проблемну область у заплутаному коді з вихідним простим кодом.

Код додатка (повний додаток; написаний на Java/Kotlin), який обфускований за допомогою сучасних інструментів, можна легко деобфускатувати та декомпілювати за допомогою таких інструментів, як інструмент JEB. Для нативного коду такі інструменти, як Ghidra або IDA Pro, можна використовувати для досягнення тих самих результатів. Подивіться на декомпільований код (схожий на вихідний код) і зрештою виконайте маніпуляції з кодом. Є відео під назвою «JEB в дії», у яких новачок може дослідити класи та визначити логіку, а також статті, у яких обфусковані програми розглядаються за допомогою таких інструментів, як інструмент JEB. Важливо розуміти, що мета полягає в тому, щоб запобігти маніпулюванню програмою. Це вимагає додаткових кроків і більш прямого підходу до обфускації.

На додаток до попереднього пункту, обфускований двійковий файл не повністю захищений від зворотного проектування. Справжній захист від зворотного проектування реалізується або централізовано, наприклад, ретрансляція ресурсів програми з центральним сервером для перевірки шаблонів послідовності інструкцій, системних викликів тощо, або децентралізовано, наприклад, безпечне вбудовування важливої інформації в захищену частину програми та впровадження захисних кодів і контрольних сум над важливою частиною програми, що ускладнює проникнення зловмисника.

Де обфускація справді має значення?

Обфускація має значення в чутливій частині коду, де обробляється важлива інформація. Двійковий блок, який відповідає за обробку конфіденційних даних, має бути частиною, де потрібно реалізувати обфускацію та інші функції безпеки, щоб повністю захистити конфіденційні дані та всі процеси навколо них. Навіть у цьому випадку безпека через невідомість (обфускацію) відіграє лише невелику роль у збереженні інформації. Це може викликати запитання з точки зору зловмисника, якщо він бачить лише певну частину коду обфускованою – чи не приклав би він всю свою енергію та зосередився на розумінні та деобфускації цієї частини коду? Відповідь: так, він буде. І саме тому важливо впроваджувати набагато більше з точки зору безпеки (зверніться до розділу нижче, щоб отримати докладнішу інформацію), щоб досягти рівня, який буде займати зловмисника протягом тривалого часу в надії, що він зрештою дасть вгору.

Елементи, які мають набагато більший пріоритет, ніж обфускація, у безпеці програми

Щоб отримати механізми безпеки, які максимально відбивають зловмисників, мають бути інші механізми. Наприклад, захист від використання програми в середовищі підвищених привілеїв, як-от рутований або зламаний телефон, емулятори, фреймворки підключення, такі як FRIDA або Xposed, налагоджений телефон/додаток тощо.

Крім того, зв’язок між програмою та серверами має бути захищений від перехоплення та атак типу Man-in-the-Middle (MitM). Це означає, що конфіденційні програми не можуть покладатися лише на безпеку, яку пропонують TLS або інші публічні протоколи, і повинні мати більше рівнів безпеки, наприклад шифрування корисного навантаження та закріплення сертифіката. Захист ключів API – таким чином захист серверної частини від доступу невідомих сторін є не менш важливим.

Зберігання інформації в зашифрованому вигляді в пісочниці мобільного додатка та можливість «самознищення» цієї бази даних із пам’яті після виявлення атаки готує програму до найгіршого та захищає у разі атаки. Подібним чином захист і шифрування активів, які входять до складу програми, таких як сертифікати сервера для TLS, важливі для збереження цілісності процесів і уникнення масштабованих атак (коли один екземпляр програми на пристрої зламано, інші екземпляри не впливають на це).).

Сильна зовнішня прив’язка довіри також важлива для безпечної атестації примірника програми, який запитує ініціалізацію – це підвищує довіру постачальника послуг до програми, яка намагається підключитися до серверної частини служби для виконання бізнес-

логіки. Забезпечення додаткового рівня довіри на додаток до рівнів, вбудованих у мобільний додаток.

Висновки. У статті наведені теоретичне узагальнення й рішення наукового завдання дослідження методів для протидії декомпіляції коду. Рішення даного завдання полягало у вирішенні наступних задач: Був проведений огляд існуючих систем для протидії декомпіляції коду. Досліджена система для протидії декомпіляції коду. На основі отриманих результатів досліджень створена програмна реалізація системи для протидії декомпіляції коду. Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання для протидії декомпіляції коду. Проведено аналіз предметної галузі в ході якого були виявлені об'єкти, взаємодія яких носить істотний характер для функціональної діяльності предметної галузі, і їхні основні характеристики; побудована алгоритм і вибраний середовище розробки.

Список літератури

1. 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
2. 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
3. Smirnov O., Kuznetsov A., Kovalchuk D., Kuznetsova T. «New technique for data hiding in cover images using adaptively generated pseudorandom sequences». CEUR Workshop Proceedings Volume 2654, 2020, Pages 1-14. (Scopus).
4. Smirnov O., Lutsenko M., Kuznetsov A., Kiian A., Kuznetsova T., «Biometric cryptosystems: overview, state-of-the-art and perspective directions». Lecture Notes in Networks and Systems, vol 152. Springer, Cham. 2021, pp 66-84. (Scopus).
5. Smirnov O., Kuznetsov A., Pushkar'ov A., Serhiienko R., Babenko V., Kuznetsova T., «Representation of Cascade Codes in the Frequency Domain». In: Radivilova T., Ageyev D., Kryvinska N. (eds) Data-Centric Business and Applications. Lecture Notes on Data Engineering and Communications Technologies, vol 48. Springer, Cham. 2021. pp 557-587. (Scopus).
6. Smirnov, O., Markovets, O. Vovk, N., Turchyn, Y., «Model of informational support for social network administrators' content creation». CEUR Workshop Proceedings Volume 2616, 2020, Pages 125-136. (Scopus).
7. Smirnov, O., Drieieva, H., Drieiev, O., Polishchuk, Y., Brzhanov, R., Aleksander, M. «Method of fractal traffic generation by a model of generator on the graph». CEUR Workshop Proceedings Volume 2616, 2020, Pages 366-379. (Scopus).
8. Smirnov, O., Shekhanin, K., Kuznetsov, A., Krasnobayev, V. «Detecting Hidden Information in FAT». International Journal of Computer Network and Information Security (IJCNIS). Vol. 12, No. 3, 2020. PP.33-43. (Scopus).
9. Smirnov, O., Drieieva, H., Drieiev, O., Simakhin, V., Bondar, S., Odarchenko, R. «Managing multifractal properties of the binary sequence generated with the Markov chains», CEUR Workshop Proceedings Volume 2608, 2020, Pages 633-645. (Scopus).
10. Smirnov, O., Kuznetsov, A., Gorbacheva, L., Babenko, V., «Hiding data in images using a pseudo-random sequence», CEUR Workshop Proceedings Volume 2608, 2020, Pages 646-660., (Scopus).
11. Smirnov, O., Kuznetsov, A., Kolovanova, I., Kuznetsova, T., «Noise immunity of the algebraic geometric codes». International Journal of Computing; 2019, Volume 18, Issue 4 – Research Institute for Intelligent Computer Systems – 2019. – P. 393-407. (Scopus).
12. Smirnov, O., Ulichev, O., Meleshko, Y., Khokh, V., Goncharenko, I. «Method of Choosing Objects for Informational Influence in Social Networks during Information Campaign Based on the Analytic Hierarchy Process». CEUR Workshop Proceedings, Vol 2588, P. 215-227, 2019. (Scopus).
13. Smirnov, O., Krasnobayev, V., Yanko, A., Kuznetsova, T. «Methods of nulling numbers in the system of residual classes». CEUR Workshop Proceedings, Vol 2588, P. 90-106, 2019. (Scopus).
14. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Pastukhov, M., Kuznetsova, K., Prokopovych-Tkachenko, D., «Discrete Signals with Special Correlation Properties», CEUR Workshop Proceedings Volume 2353, CEUR Workshop Proceedings 2019, Pages 618-629. (Scopus).
15. Smirnov, O., Kuznetsov, A., Kiian, A., Kuznetsova, K., Ivko, T., Prokopovych-Tkachenko, D., «Soft Decoding Based on Ordered Subsets of Verification Equations of Turbo-Productive Codes», CEUR Workshop Proceedings Volume 2353, CEUR Workshop Proceedings 2019, Pages 873-884. (Scopus).

16. тСмирнов А.А., Коваленко А.В. Комплекс математических моделей технологии тестирования web-приложений. Інформаційні технології: сучасний стан та перспективи: монографія / За загальною редакцією В.С. Пономаренка. – Х.: ТОВ «ДІСА ПЛЮС», 2018. – 461 с.
17. Смирнов А.А., Коваленко А.В. Разработка метода управления рисками разработки программного обеспечения. Інформаційні технології: проблеми та перспективи: монографія / За загальною редакцією В.С. Пономаренка. – Х.: Видавець Рожко С.Г., 2017. – 447 с.
18. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Смірнов С.А., Поліщук Л.І., «Дослідження стійкості до диференціального криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» Системи управління, навігації та зв'язку, 2022, № 3(69). С. 93-98. 2022.
19. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Поліщук Л.І., Смірнов С.А. «Дослідження статистичної стійкості та швидкісних характеристик запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» Вісник Хмельницького національного університету. Серія: «Технічні науки», № 2 (307). С. 46-52. 2022.
20. Смірнов О.А., Смірнова Т.В., Константинова Л.В., Смірнов С.А., Якименко Н.М., «Дослідження стійкості до лінійного криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» Системи управління, навігації та зв'язку, 2022, № 1(67). С. 84-89.

УДК 004

В.Шевченко, магістр гр. КН-21М-1,4,
Центральноукраїнський національний технічний університет

ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ПРОТОКОЛІВ СТЕКУ TCP/IP У ХМАРНИХ СЕРВІСАХ

У статті розроблено програмне забезпечення, яке призначено для системи доступу до хмарних сервісів з використанням технології PKI. Метою розробки є дослідження та програмна реалізація системи доступу до хмарних сервісів з використанням технології PKI. Об’єктом дослідження є процес доступу до хмарних сервісів з використанням технології PKI. Предметом дослідження є методи доступу до хмарних сервісів з використанням технології PKI. Методи дослідження базуються на методах захисту інформації та хмарних обчислень, методах математичної статистики, методах розробки програмного забезпечення. Результат роботи – програмна реалізація системи доступу до хмарних сервісів з використанням технології PKI. В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.
комп’ютерні науки, хмарні сервіси, PKI

Постановка проблеми. Сьогодні більшість компаній так чи інакше використовують Інтернет, і із цим зв'язані проблеми захисту віддалених і мобільних користувачів інформаційних систем компанії, захисту корпоративних хмарних сервісів (інтранет-сайту й будь-якого додатка компанії, що працює по http протоколу). Інтернет – це зона підвищеного ризику, відповідно, потрібні спеціальні засоби захисту при роботі віддалених користувачів з WEB-додатками по SSL-протоколу. Таким чином, підсистема захисту WEB-ресурсів вирішує наступні задачі:

– забезпечення єдиного інтерфейсу до додатків;
– інтегрований контроль доступу до корпоративних хмарних сервісів;
– захист клієнтських браузерів;
– захист хмарних сервісів.

Існує багато технологій захисту хмарних сервісів. У магістерському проекті пропонується система захисту основана на використанні протоколів SSL/TLS, які побудовані з використання інфраструктури відкритих ключів (PKI).

У протоколі SSL/TLS використовується ряд симметричних алгоритмів, асиметричних