УДК 004

Д.Колесник, магістр гр. КН-24М,
Центральноукраїнський національний технічний університет

ДОСЛІДЖЕННЯ ТА ПРИНЦИПИ ПОБУДОВИ СИСТЕМИ ПРОТИДІЇ ДЕТЕКТУВАННЮ АНТИВІРУСНОЮ СИСТЕМОЮ ЗА ДОПОМОГОЮ ШТУЧНОГО ІНТЕЛЕКТУ

У статті розроблено програмне забезпечення, яке призначено для системи протидії детектуванню антивірусною системою за допомогою штучного інтелекту. Метою розробки є дослідження та принципи побудови системи протидії детектуванню антивірусною системою за допомогою штучного інтелекту. Об'єктом дослідження є процес протидії детектуванню антивірусною системою за допомогою штучного інтелекту. Предметом дослідження є методи протидії детектуванню антивірусною системою за допомогою штучного інтелекту. Методи дослідження базуються на методах штучного інтелекту, методах захисту інформації, методах математичної статистики, методах розробки програмного забезпечення. Результат роботи – програмна реалізація системи протидії детектуванню антивірусною системою за допомогою штучного інтелекту. В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

протидія, детектування, антивірусна система, штучний інтелект

Постановка проблеми. У даній роботі будуть запропоновані ефективні методи обходу статичного, динамічного й евристичного аналізу, використовуваного в новітніх антивірусних продуктах. Деякі техніки вже відомі навзагал, але є й додаткові трюки, які є ключовими при генерації недетектуємого вірусу. Обсяг файлу не менш важливий при використанні даних методів, і я постарався оптимізувати розмір настільки, наскільки можливо. У цьому документі також буде порушена тема внутрішнього пристрою антивірусів і операційної системи Windows.

Методи реалізації технік антидетекта залежать від типу вірусу. Всі методи, описувані в даній роботі, будуть працювати з усіма типами вірусів, однак основна увага приділена складеним корисним навантаженням для meterpreter, оскільки цей командний інтерпретатор уміє практично все, що й інші вірусні програми. Одержання сесії за допомогою meterpreter на віддаленій машині відкриває масу можливостей: розширення привілеїв, крадіжку облікових записів, міграцію між процесами, маніпуляцію реєстром і інші трюки пост-експлуатації.

Крім того, навколо meterpreter зібралося потужне й активне співтовариство, і цей інструмент популярний серед фахівців з безпеки.

Аналіз останніх досліджень і публікацій. При аналізі останніх досліджень і публікацій [1-30] було виявлено певні прогалини у забезпеченні системи протидії детектуванню антивірусною системою за допомогою штучного інтелекту.

Мета й завдання дослідження. Метою роботи є дослідження та принципи побудови системи протидії детектуванню антивірусною системою за допомогою штучного інтелекту.

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

– Огляд існуючих систем протидії детектуванню антивірусною системою за допомогою штучного інтелекту.

– Дослідження системи протидії детектуванню антивірусною системою за допомогою штучного інтелекту.

– Програмна реалізація системи протидії детектуванню антивірусною системою за допомогою штучного інтелекту.

Об'єктом дослідження є процес протидії детектуванню антивірусною системою за допомогою штучного інтелекту.

Предметом дослідження є методи протидії детектуванню антивірусною системою за допомогою штучного інтелекту.

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

Виклад основного матеріалу. Перед початком вивчення ефективним методів, варто згадати пару слів про те, які є проблеми із широко відомими техніками й утилітами. На сьогоднішній день антивірусні компанії повністю оцінили всі ризики й найчастіше крім пошуку сигнатур і вірусного впливу шукають ознаки крипторів і пакувальників. У порівнянні з вірусами детектування крипторів і пакувальників простіше, оскільки останні мають однакові підозрілі алгоритми, на зразок розшифровки PE-файлу й запуску в пам'яті.

PE ін'єкція

Для того щоб зрозуміти схему запуску PE-образа в пам'яті, необхідно згадати про тім, як ОС Windows завантажує PE-файли. Звичайно при компіляції PE-файлу на адресу головного модуля встановлюється значення 0x00400000. Потім обробляються покажчики повних адрес і розраховуються інструкцій довгих переходів на базі адреси головного модуля. По закінченні компіляції в PE-файлі створюється секція таблиці релокації, що містить адреси інструкцій, що залежать від базової адреси образа, наприклад, покажчики повних адрес і інструкцій довгого переходу.

Під час виконання PE-образа операційна система перевіряє доступність кращого адресного простору. Якщо даний простір не доступно, перед стартом системний завантажник процесів повинен підбудувати абсолютні адреси в пам'яті. За допомогою секції релокації завантажник виправляє всі інструкції, що залежать від адреси, і запускає припинений процес. Весь цей механізм називається «Рандомізація розміщення адресного простору» (Address Space Layout Randomization; ASLR).

Для того щоб виконати PE-образ у пам'яті, криптору потрібно распарсити PE-заголовки й перемістити абсолютні адреси. Найчастіше там є підроблений системний завантажник, що відразу викликає підозру. Коли ми аналізуємо криптори, написані на З або мовах високого рівня, практично в кожному випадку спостерігаються API функції «NtUnmapViewOfSection» і «ZwUnmapViewOfSection», які попросту відключають подання секції від віртуального адресного простору процесу. Ці функції грають дуже важливу роль при використанні методу RunPE, що зустрічається практично в 90% крипторів.

Природно, антивірусні продукти не можуть позначати кожну програму як вірусну, котра використовує ці API-функції. Однак сам факт наявності даних функцій значить багато чого. Існує невеликий відсоток крипторів (в основному написаних на асемблері), які не використовують ці функції й виконують релокацію вручну. Якийсь час ці криптори працюють ефективно, але рано або пізно наступлять наслідку через спробу підробити системний завантажник. Інший серйозний мінус – при шифруванні PE-файлу серйозно збільшується ентропія. Якщо антивірусний сканер детектує нетиповий рівень ентропії, цілком можливо PE-файл буде позначений як підозрілий.

Ідеальний метод

Концепція шифрування вірусного коду має право на життя, але функцію дешифрування варто обробити гарною обфускацією. Коли справа доходить до виконання дешифрованого коду в пам'яті, можна обійтися без переміщення абсолютних адрес. Крім того, усередині вірусу потрібно передбачити алгоритм перевірки знаходження усередині пісочниці. Якщо перевірка виявила, що вірусний код аналізується антивірусом, функцію дешифрування запускати не слід. Замість шифрування всього PE-файлу, більш грамотно шифрувати шелл-код або тільки секція .text. У цьому випадку ентропія й розмір файлу залишаються на прийнятному рівні, і не міняються заголовки образа й секції.

Функція буде детектувати, чи аналізується вірус динамічно усередині чи пісочниці ні. Якщо функція виявляє ознаки антивірусного сканера, то або знову викликається функція main, або вірус аварійно завершується. Якщо ж ознаки сканування відсутні, буде викликана функція для розшифровки шелл-коду.

Для збереження ентропії й розміру на прийнятному рівні я будуть обробляти шелл-код шифрувальником на базі алгоритму XOR з використанням мультибайтового ключа. Алгоритм XOR не є стандартом в індустрії, як наприклад RC4 або blowfish, але в нашім випадку сильне шифрування не потрібно. Антивірусні продукти не будуть займатися дешифруванням. Нам досить захисту від статичного аналізу рядків. Крім того, XOR-Шифрування набагато швидше й не використовує бібліотек, що впливає на підсумковий розмір вірусу.

Оскільки ми пишемо новий шматок вірусу, геш-сигнатура не буде відома антивірусним продуктам, і нам не потрібно турбуватися про виявлення за допомогою детектування, побудованого на базі сигнатур. Ми шифруємо шелл-код і обфускуємо функції анти-детектування/налагодження й алгоритм дешифрування, що цілком достатньо для обходу статичного/евристичного аналізу. Нам залишилося знайти спосіб обходу динамічного аналізу, що є найбільш важливою частиною функції «AV detect». Перед написанням даної частини необхідно розібратися, як працюють евристичні алгоритми антивірусних продуктів.

Евристичні движки

Евристичні движки засновані на статичному аналізі й механізмі правил. Головна мета таких движків – детектування вірусів нового покоління, які ще не відомі, за допомогою угруповання й оцінці погроз/ризиків в окремих фрагментів коду відповідно до визначених критеріїв. Навіть при скануванні найпростішої програми, що виводить на печатку «Привіт, світ», евристичний движок визначає рівень погрози й ризику. Якщо оцінка перевищує граничний рівень, файл позначається як вірусний. Евристичні движки – найбільш просунутий компонент антивірусних продуктів і використовують значну кількість правил і критеріїв. Оскільки антивірусні компанії не випускають документації, що описує евристичні движки, відомі лише деякі критерії оцінки погроз/ризиків. Природно, тут може бути багато помилкових допущень і помилок.

От деякі відомі правила, використовувані при оцінці погроз:

Присутність циклу дешифрування.

- Читання поточного ім'я комп'ютера.
- Читання криптографічного машинного GUID.
- З'єднання з випадковими доменами.
- Читання дати установки Windows.
- Видалення файлів, що виконуються.
- Присутність потенційного IP-адреси в пам'яті бінарного файлу.
- Модифікація налаштувань проксі-сервера.
- Установка хуків/патчів до запущеного процесу.
- Інжектування в explorer.
- Інжектування у віддалений процес.
- Запит інформації про процес.
- Установка процесу в режим помилки для приховання повідомлення об помилки.
- Нетипова ентропія.
- Можлива перевірка на присутність антивірусного движка.
- Присутність функціонала для розширення привілеїв.
- Модифікація налаштувань політик додатків.
- Читання версії BIOS системи/відеокарти.
- Кінець PE-заголовка перебуває усередині нестандартної секції.
- Створення захищених областей пам'яті.
- Створення безлічі процесів.
- Спроба призупинити роботу на довгий час.

– Нестандартні секції.
– Читання Windows Product Id.
– Присутність функціонала для запуску/взаємодії із драйверами пристроїв.
– Присутність функціонала для блокування користувальницького уведення.

При написанні функції по детектуванню антивірусних продуктів і дешифрування шелл-коду, варто брати до уваги ці правила.

Дешифрування шелл-коду

Обфускація механізму дешифрування надзвичайно важлива, оскільки більшість евристичних движків у стані виявити цикли дешифрування усередині PE-файлів. Після значного збільшення випадків, коли програма вимагала викуп, деякі евристичні движки були майже повністю заточені під детектування процедур дешифрування. Після виявлення процедури дешифрування деякі сканери чекали, поки в регістрі ECX не виявиться значення 0, що в більшості випадків означає закінчення циклу. По досягненню закінчення циклу дешифрування виробляється повторний аналіз умісту файлу, що буде являти собою функцію дешифрування шелл-коду.

Показано цикл for, усередині якого виконуються логічні операції XOR між байтом шелл-коду й байтом ключа. Асемблерні блоки нагорі й унизу, по суті, не виконують ніяких операцій. Ці блоки «накривають» XOR-операцію випадковим набором байтів і переходів. Оскільки ми не користуємося просунуті механізми дешифрування, подібної обфускації буде досить для функції «Decrypt Shellcode».

Детектування динамічного аналізу

У процесі написання механізму детектування пісочниці нам необхідно зробити обфускацію методів. Якщо евристичний движок виявить ознаки методів анти реверс-інжинірингу, різко зросте оцінка погрози з боку нашого вірусу.

Детектування відладника

Наш перший механізм детектування антивірусного продукту буде перевіряти присутність відладника усередині процесу. Для рішення даного завдання існує API-функція, що визначає, чи налагоджує процес відладником рівня користувача (user-mode debugger). Але ми не будемо використовувати дану функцію, оскільки більшість антивірусних продуктів моніторять вираження API-Викликів. Швидше за все, після детектування виклик буде розцінений як протидіючому реверсу-інжинірингу. Замість використання API-функції ми будемо шукати байт «BeingDebuged» в PEB-блоці.

Ділянка коду на рисунку вище витягає байт BeingDebuged з PEB-блоку. Якщо відладник є присутнім, перевірка повторюється заново, і так доти, поки в стеці не буде переповнення. Після переповнення спрацює виключення, і процес буде закритий. Цей шлях – найпростіший для виходу із програми. Ручна перевірка байта BeingDebuged дозволить обійти пристойну кількість антивірусних продуктів, але в деяких додатках передбачений подібний випадок, і ми повинні зробити обфускацію коду, щоб захиститися від статичного аналізу рядків.

Додавання переходів (інструкцій JZ) після кожної операції не вплине на логіку роботи функції, але додавання сміттєвих байтів між переходами дозволить обійти фільтри статичного аналізу рядків.

Завантаження підробленої бібліотеки

У цьому методі ми спробуємо завантажити неіснуючу бібліотеку під час виконання додатка. Звичайно коли ми намагаємося завантажити неіснуючу бібліотеку, HISTENCE повертає NULL, але деякі механізми динамічного аналізу, використовувані в антивірусних продуктах, допускають такі випадки й продовжують подальше дослідження потоку виконання додатка.

Використання функції GetTickCout

У цьому методі ми будемо опиратися на той факт, що час, що приділяється на сканування, обмежено. У більшості випадків антивірусні сканери створені для кінцевого користувача. Відповідно, додаток повинне бути як можна більш дружелюбно й придатне для

щоденного використання. Цей факт означає, що сканер не може витрачати занадто багато часу на сканування файлів, а повинен завершувати весь процес якнайшвидше. Спочатку розроблювачі вірусів використовували функцію «sleep» з метою дочекатися закінчення сканування, але зараз цей трюк практично не працює, оскільки кожний антивірусний продукт пропускає дану функцію. Ми будемо використовувати інший метод, показаний існує, що використовує API-функцію «GetThickCount». Дана функція витягає кількість мілісекунд, що пройшли з моменту запуску системи (до 49.7 днів). Ми будемо використовувати цю функцію для одержання часу, що пройшло з моменту запуску операційної системи, потім зупинимося на 1 секунду. Після функції sleep ми перевіримо, чи була ця ділянка пропущена, за допомогою порівняння двох значень, отриманих за допомогою GetTickCout.

Перевірка кількості ядер

Цей метод перевіряє кількість ядер процесора, що присутні у системі. Оскільки антивірусні продукти не можуть займати занадто багато ресурсів, ми можемо перевірити кількість ядер для того, щоб визначити, чи перебуваємо ми в пісочниці. Багато антивірусів не підтримують мульти-ядерну обробку й не можуть зарезервувати більше одного ядра для пісочниці.

Виділення великого обсягу пам'яті

Цей метод також експлуатує обмежений час, використовуваний при скануванні. Ми виділяємо близько 100 Мб пам'яті й заповнюємо даний простір порожніми байтами. Потім пам'ять звільняється.

Коли пам'ять, використовувана програмою, під час виконання починає рости, через якийсь час антивірусні сканери зупиняються, щоб не витрачати багато часу на аналіз файлу. Цей метод можна використовувати кілька разів. Дана техніка дуже стара й примітивна, але усе ще допомагає обійти пристойна кількість сканерів.

Маніпуляція прапором трасування

Як треба з назви, цей прапор використовується під час трасування програм. Якщо прапор трасування встановлений, кожна інструкція буде ініціювати виключення «SINGLE_STEP». Прапором трасування можна маніпулювати з метою протидії трасувальниками за допомогою коду, показаного існує:

Перевірка мьютексів

Цей метод через свою простоту виглядає дуже перспективним. Суть техніки полягає в перевірці присутності певного мьютекса в системі.

Якщо функція «CreateMutex» не поверне помилку ERROR_ALREADY_EXISTS, вірус запускається ще раз, оскільки більшість антивірусних продуктів не дозволяють програмам, які піддаються динамічному аналізу, запускати нові процеси або працювати з файлами поза пісочницею. Якщо за результатами перевірки мьютекса вертається помилка ERROR_ALREADY_EXISTS, можна запускати функцію дешифрування. Існують набагато більше креативні способи використання мьютексов з метою анти-детектування.

Правильні методи виконання шелл-коду

Починаючи з Windows Vista, компанія Microsoft впровадила технологію DEP (Data Execution Prevention; Запобігання виконання даних). Ця міра, спрямована на зміцнення безпеки, допомагає запобігти псуванню комп'ютера шляхом періодичного моніторингу. Моніторинг дозволяє підтримувати коректно використання пам'яті. Якщо механізм DEP виявляє випадок некоректного використання пам'яті вашого комп'ютера, програма закривається, і користувач сповіщається. Відповідно, ви не можете просто помістити трохи байт у символьний масив і виконати свою творчість. Вам потрібно виділити область пам'яті із прапорами на читання, запис і виконання за допомогою API-функцій.

Компанія Microsoft передбачила трохи API-функцій для резервування сторінок пам'яті. Більшість вірусів для резервування використовують функцію «VirtualAlloc», що, як ви вже догадалися, полегшує завдання по детектуванню. Використання інших функцій для маніпуляції пам'яттю допоможе виконати той же самий трюк більш непомітно.

Далі я покаджу методи виконання шелл-коду за допомогою різних API-функцій.

HeapCreate/HeapAlloc:

ОС Windows також дозволяє створити купи з атрибутами на читання, запис і виконання.

LoadLibrary/GetProcAddress

Комбінація WINAPI-функцій LoadLibrary і GetProcAddress дозволяє використовувати інші API-функції. У цьому випадку не буде прямого виклику функцій, пов'язаних з маніпуляцією пам'яті, а сам вірус, імовірно, буде більше непомітний.

GetModuleHandle/GetProcAddress

Цей метод зовсім не використовує функцію LoadLibrary, а одержує оброблювач уже завантаженого модуля kernel32.dll за допомогою функції GetModuleHandle. Дана техніка для запуску шелл-коду, можливо, одна із самих непомітних.

Мультипоточність

PE-файли, що використовують кілька потоків, завжди складніше аналізувати, у тому числі й для антивірусних продуктів. За допомогою мультипоточного підходу ми можемо запускати шелл-код і одночасно продовжувати виконання функції «AV Detect».

У коді, показаному вище, виконання шелл-коду відбувається в окремому потоці. Паралельно виконанню шелл-коду в нескінченному циклі виконується функція для обходу антивірусу. Такий підхід дозволяє перевіряти присутність пісочниці й динамічного аналізу, що життєво важливо для обходу просунутих евристичних движків, які чекають запуску шелл-коду.

При компіляції вихідних текстів вірусів необхідно включити захист стека й видалити символи з метою утруднення реверс-інжинірингу й зменшення розміру кінцевого файлу, що виконується. Рекомендується компілювати в Embarcadero RAD Studio через присутність асемблерних вставок.

Якщо скористатися одночасно всіма методами, продемонстрованими в даній роботі, згенерований вірус зможе обійти 35 найбільш просунутих антивірусних продукту.

Впровадження бекдору в PE-файл

Далі, у даній роботі розповімо про декілька методів, використовуваних для розміщення вірусу в PE-файлі. Щоб зрозуміти суть матеріалу, що викладається, читачеві необхідно бути знайомим з асемблером для архітектури x86 і відладниками хоча б на середньому рівні й розуміти формат PE файлів. Цей документ був опублікований 8 грудня 2016 року на сайті pentest.blog, а також підготовлений в PDF форматі для офлайнового читання.

У цей час практично всі фахівці з безпеки, пентестери й аналітики вірусів мають справу з бекдорами на щоденній основі. Приміщення трояна в систему або в конкретну програму – найбільш популярний спосіб для підтримки постійного доступу до цільової машини. У більшості статей розповідається про методи для імплантування вірусів в 32 бітні PE файли, але оскільки PE-формат являє собою модифіковану версію формату Unix COFF (Common Object File Format; Загальний формат об'єктних файлів), логіка, закладена в основу цих технік, застосовна й до всіх інших типів файлів, що виконуються. Крім того, непомітність убудованого вірусу дуже важлива й прямо впливає на тривалість знаходження в системі. Методи, які будуть описані в даній роботі, спрямовані на зниження відсотка детектування настільки наскільки можливо. Перед подальшим читанням рекомендую ознайомитися з першою частиною, де розповідалося про обхід технік детектування, внутрішньому пристрої антивірусів і фундаментальних постулатів антидетектування.

Уведемо додаткову термінологію.

Навчальне проникнення

Існують групи, що складаються зі світлих хакерів (або білих капелюхів), які атакують цифрову інфраструктуру організації, як якби це робив реальний зловмисник, для того, щоб протестувати стійкість системи до зовнішніх погроз (по-іншому це називається пентест). Наприклад, компанія Microsoft регулярно проводить подібні кібер-навчання. Плюси від

подібних заходів лежать на поверхні – у процесі можуть найтися проломи й проблеми, які можна заздалегідь усунути. Крім того, тут можуть виявитися шляхи влучення конфіденційної інформації назовні, схеми експлуатації й інші «недокументування» можливості системи.

Рандомізація розміщення адресного простору

ASLR являє собою техніку безпеки, спрямовану на захист від атак, пов'язаних з переповнення буфера. Щоб не дати зловмисникові коректно перейти на певну функцію усередині пам'яті, ASLR у випадковому порядку вишиковує позиції ключові областей інформації в адресному просторі процесу. Сюди ж включається базова адреса файлу, що виконується, і позиції стека, купи й бібліотек.

Code Cave (Печера в коді)

Code Cave являє собою шматок коду, що записується іншою програмою на згадку стороннього процесу. Цей код може бути виконаний за допомогою створення віддаленого потоку усередині цільового процесу. Code cave найчастіше є посиланням на секцію скриптових функцій коду, куди можна інжектувати будь-які інструкції. Наприклад, якщо в пам'яті скрипту 5 байт, і 3 байти використовуються, в 2 байти, що залишилися скрипту можна додати зовнішній код.

Контрольна сума

Контрольна сума являє собою невелику порцію інформації із блоку цифрових даних для виявлення помилок, які можуть з'явитися під час передачі або зберігання файлу. Звичайно за допомогою контрольної суми перевіряється настановний файл після одержання із сервера. Між нами говорячи, контрольні суми звичайно використовуються для верифікації цілісності даних, але не враховують дійсність інформації.

Основні методи

Всі приклади з даної роботи будуть продемонстровані на базі файлу, що виконується, SSH-клієнта з ім'ям putty. Є кілька причин для вибору саме цього додатка як піддослідний зразок. Putty використовує безліч бібліотек і API-функцій. По-друге, впровадження вірусу в ssh-клієнт залучає менше уваги, оскільки програма вже виконує tcp-з'єднання, і, таким чином, буде простіше уникнути моніторингу з боку системи безпеки.

Код бекдору буде взятий із шелл-коду, використовуваного при зворотному TCP-з'єднанні й написаного під meterpreter. Головна мета – інжектувати шелл-код у цільовий PE-файл без псування функціональності додатка. Інжектований шелл-код буде запускатися в окремому потоці й буде постійно намагатися приєднатися до оброблювача. Друге завдання – під час виконання всіх цих маніпуляцій необхідно залишатися непомітним настільки, наскільки можливо.

Загальний підхід впровадження вірусу в PE-файл складається з 4 кроків:

– Знаходження доступного простору для коду бекдору.
– Перехоплення потоку виконання.
– Впровадження бекдору.
– Відновлення потоку виконання.

У кожному кроці є свої нюанси, які прямо впливають на стійкість, тривалість і непомітність убудованого вірусу.

Проблема з доступним простором

Знаходження доступного простору – перший крок до реалізації нашого завдання. Надзвичайно важливо вибрати правильне місце усередині PE-файлу для впровадження бекдору. Оцінка погрози з боку зараженого файлу сильно залежить від того, як ви вирішите це завдання. Тут існує два підходи.

Додавання нової секції

У порівнянні із другим підходом тут більше ймовірність виявлення. Хоча, з іншого боку, при додаванні нової секції в нас немає обмежень по просторі, і ми можемо впровадити бекдор будь-якого розміру.

За допомогою дизасемблера або редактори LordPE PE-файл можна розширити за допомогою додавання заголовка нової секції. Існує таблиця секцій файлу, що виконується, putty. За допомогою PE-редактора додана нова секція «NewSec» розміром 1000 байт.

Під час створення нової секції важливо встановити прапори на читання/запис/виконання, щоб запустити шелл-код, коли PE-образ проектується (map) на згадку.

Після додавання заголовка секції необхідно адаптувати розмір файлу, що робиться в шістнадцятковому редактору за допомогою додавання порожніх байтів розміром з нову секцію в кінець файлу.

Після додавання нової порожньої секції необхідно запустити виконується файл, що, і перевірити, є чи помилки. Якщо все пройшло гладко, нова секція готова до модифікації в відладнику.

Рішення проблеми доступного простору за допомогою додавання нової секції має деякі недоліки. Практично всі антивіруси пізнають нестандартні секції, і якщо, до того ж, там є повний набір прапорів на читання/запис/виконання, те ця ситуація виглядає ще більш підозріло.

Навіть якщо ми просто додамо нову секцію з повними правами без бекдору, деякі антивіруси вже позначають виконується файл, що, як вірусний.

Використання Code Cave

У другому методі, спрямованому на рішення проблеми доступного простору, використовуються code cave'и із цільового файлу, що виконується. Практично всі скомпільовані бінарні файли мають code cave'и, які можуть бути використані для впровадження вірусу. Code cave у порівнянні з новою секцією залучає набагато менше уваги, оскільки в цьому випадку застосовуються вже існуючі звичайні секції. Додатковий і не менш важливий плюс полягає в тім, що після впровадження вірусу розмір PE-файлу не міняється. Однак ця техніка має свої недоліки.

Кількість і розмір code cave’ов варіюється від файлу до файлу, але в цілому загальний розмір менше, ніж при додаванні нової секції. При використанні code cave’ов код бекдору варто зменшувати настільки, наскільки можливо. Другий недолік – набір прапорів. Оскільки виконання буде перенаправлятися в code cave, у секції повинні бути права на виконання. У випадку з деякими шелл-кодами (які самі себе кодують або піддають обфускації) потрібно права на запис для того, щоб виконувати зміни усередині секції.

Використання декількох code cave'ов дозволяє перебороти обмеження із простором. Тут же додатковий плюс у тім, що вірус збирається з окремих шматків. Однак зміна привілеїв секції буде виглядати підозрілим. Існують просунуті методи для модифікації привілеїв в областей пам'яті під час виконання додатка з метою запобігання прямої зміни прапорів секції, але оскільки ці техніки вимагають спеціалізованого шелл-коду, кодування й парсингу таблиці IAT, ця тема буде порушена в наступних статтях.

За допомогою утиліти Cminer дуже легко підрахувати всі code cave'и бінарного файлу. Використовуємо команду ./Cminer putty.exe 300 для пошуку code cave'ов більше 300 байт.

У нашім випадку знайдено 5 гарних екземплярів для наступного використання. Стартова адреса задає адреса віртуальний пам'яті (virtual memory address, VMA) в code cave при завантаженні PE-файлу на згадку. Зсув файлу, вимірюваного в байтах, – адреса місцезнаходження потрібної області усередині PE-файлу.

За результатами пошуку з'ясувалося, що більшість областей перебувають усередині секції даних. Оскільки в цих секціях не встановлений прапор на виконання, будуть потрібні зміни. Розмір бекдору біля 400-500 байт, і області Cave 5 вистачить позаочі. Стартова адреса даної області потрібно зберегти, а після зміни привілеїв секції перший етап впровадження вірусу можна вважати завершеним. Тепер потрібно перенаправляти виконання на нашу область.

Перехоплення потоку виконання

На цьому етапі потрібно перенаправляти потік виконання на код бекдору за допомогою модифікації потрібної інструкції у файлі, що виконується. Тут варто згадати важливу деталь щодо вибору інструкції для зміни. Всі бінарні інструкції мають розмір у байтах. Щоб перейти до адреси розташування бекдору буде потрібно довгий перехід (long jump) розміром, що використовує 5 або 6 байт. При зміні бінарного файлу інструкція, що буде патчиться, повинна бути того ж розміру, що й довгий перехід, інакше попередня й наступна інструкція будуть зіпсовані.

Дуже важливий вибір правильного місця для перенапрямку виконання, оскільки якщо перенапрямок буде здійснюватися прямо, неминуче детектування на етапі динамічного аналізу антивірусними продуктами.

Приховання усередині користувальницьких функцій

Перше, що спадає на думку, для обходу пісочниці й динамічного аналізу – відкладене виконання шелл-коду або використання детектора пісочниці, за результатами роботи якого виконується та або інша вітка алгоритму. З іншого боку, у більшості випадків через обмеження по розмірах ми не можемо додати зайві ділянки коду в PE-файл. Крім того, реалізація технік антидетектування на низькому рівні вимагає багато часу й сил.

Цей метод використовує функції, що вимагають дій користувача. Перенапрямок виконання усередині подібних функцій буде спрацьовувати тільки в тому випадку, якщо користувач працює в програмі. Якщо дана техніка буде реалізована коректно, успіх практично гарантований, і, до того ж, не буде збільшений розмір бекдору.

По натисканню кнопки «Open» із графічної оболонки запускається функція перевірки встановленого IP-адреси.

Якщо поле IP-адреси не порожнє, і значення коректне, запускається функція для з'єднання із зазначеним IP-адресою.

Якщо клієнт успішно створив ssh-сесію, з'явиться нове вікно для уведення ім'я користувача й пароля.

Саме в цьому місці відбудеться перенапрямок. Оскільки антивірусні продукти не настільки просунуті для аналізу такого роду механізмів, впроваджений бекдор, швидше за все, не буде виявлений за допомогою динамічного аналізу.

За допомогою нескладних методів реверс-інжинірингу, призначених для роботи з рядками й посиланнями на рядки, буде не складно знайти адресу функції з'єднання після того, як клієнт установив з'єднання із призначеним IP-адресою. Рядок «login as:», що з'являється в спливаючому вікні, допоможе нам знайти адресу функції з'єднання. У пошуках посилань на рядки нам допоможе IDA Pro.

Для знаходження рядка «login as:» в IDA Pro відкриваємо Views->Open Subviews->Strings on IDA

Після знаходження потрібного рядка двічі клікаємо і переходимо до місцезнаходження. Усередині секцій даних IDA знаходить всі перехресні посилання на рядки. Для виводу всіх перехресних посилань натискаємо комбінацію клавіш «Ctrl+X».

Існує посилання усередині функції, що виводить рядок «login as:»

Існує інструкція, що ми будемо змінювати. Після виконання коду бекдору, ця інструкція буде використовуватися знову.

Після зміни інструкції PUSH 467C7C на JMP 0x47A478 процес перенапрямку потоку виконання можна вважати завершеним. Важливо не забувати, що адреса наступної інструкції буде використовуватися як адреса повернення після виконання коду вірусу. Наступний крок – інжектування коду бекдору.

Інжектування коду бекдору

Перша думка, що спадає на думку при впровадженні бекдору, – збереження регістрів перед виконанням вірусного коду. Кожне значення усередині регістрів – надзвичайно важливо для виконання програми. Помістивши інструкції PUSHAD і PUSHFD на початку code cave, ми зможемо зберегти всі регістри й прапори регістрів усередині стека. Ці значення

будуть повернуті після виконання коду вірусу, і програма продовжить виконання без яких-небудь проблем.

Як було згадано раніше, наш бекдор являє собою зворотний tcp-шелл-код під meterpreter, узятий із проекту metasploit. Однак буде потрібно деякі зміни усередині шелл-коду. Звичайно зворотний tcp-шелл-код намагається приєднатися до оброблювача деяка кількість разів, і якщо з'єднатися не вдалося, процес закривається за допомогою виклику API-функції ExitProcess.

Проблема полягає в тім, що якщо з'єднатися з оброблювачем не вийде, виконання клієнта putty буде зупинено. Після внесення невеликих змін після невдачі шелл-код буде знову намагатися з'єднатися з оброблювачем. До того ж, небагато зменшиться розмір шелл-коду.

Після внесення змін усередині асемблерного коду виконуємо компіляцію за допомогою команди nasm -f bin stager_reverse_tcp_nx.asm. Тепер зворотний tcp-шелл-код готовий до вживання, але поки ще не поміщений у правильне місце. Наша мета – здійснити виконання в окремому потоці, для створення якого буде потрібно окремий шелл-код, що виконує виклик API-функції CreateThread. Функція буде вказувати на первісний зворотний tcp-шелл-код.

Після приміщення байтів шелл-коду усередину файлу createthread.asm у шістнадцятковому форматі, як показано на рисунку вище, виконуємо компіляцію за допомогою команди nasm -f bin createthread.asm. Тепер шелл-код готовий для впровадження в code cave, але перед вставкою варто виконати кодування для того, щоб обійти статичний/сигнатурний аналіз антивірусу. Оскільки всі кодувальники із проекту metasploit відомі більшості антивірусів, рекомендується використовувати нестандартне кодування. У даній роботі не буде розказано про створення нестандартних кодувальників, але можна обійтися сполученням декількох кодувальників із проекту metasploit. Після кожного акту кодування завантажуємо шелл-код у проект Virus Total в «сирому» форматі й перевіряємо результати перевірки. Комбінуємо кодувальники доти, поки шелл-код стане повністю непомітним.

Після правильного кодування шелл-код готовий до впровадження усередину code cave. Вибираємо інструкцію, яка треба за інструкцією PUSHFD, і натискаємо Ctrl+E в відладнику Immunity Debugger. Шелл-код буде вставлений у шістнадцятковому форматі.

Одержати закодований шелл-код у шістнадцятковому форматі можна двома способами: вивести на печатку за допомогою команди xxd -ps createthread або відкрити й скопіювати в шістнадцятковому редакторі. При копіюванні шістнадцяткових значень в відладник Immunity Debugger не забувайте про обмеження на копіюємі байти, які присутні при вставці коду. Необхідно запам'ятати останні два вставлених байти, і після натискання на кнопку OK потрібно зробити повторне копіювання наступних ділянок. Коли шелл-код повністю вставлений в code cave, операцію по впровадженню можна вважати завершеної.

Відновлення потоку виконання

Після створення потоку виконання бекдору необхідно відновити виконання програми. Тобто регістр EIP повинен указувати на функцію, що перенаправляла виконання на code cave. Однак перед переходом на цю функцію варто відновити раніше збережені регістри.

Помістивши інструкції POPFD і POPAD у кінець шелл-коду, ми відновимо усе раніше збережені регістри зі стека в тім же порядку. Після відновлення регістрів необхідно згадати ще про один нюанс. При перехопленні потоку виконання інструкція PUSH 467C7C була замінена на інструкцію JMP 0x47A478 для того, щоб перенаправляти виконання на code cave. При приміщенні інструкції PUSH 467C7C у кінець коду, перехоплена інструкція також відновиться. Тепер прийшов час повернутися до функції, що перенаправляла виконання до code cave за допомогою вставки інструкції JMP 0x41CB73. Кінець результуючого коду повинен виглядати, як показано існує.

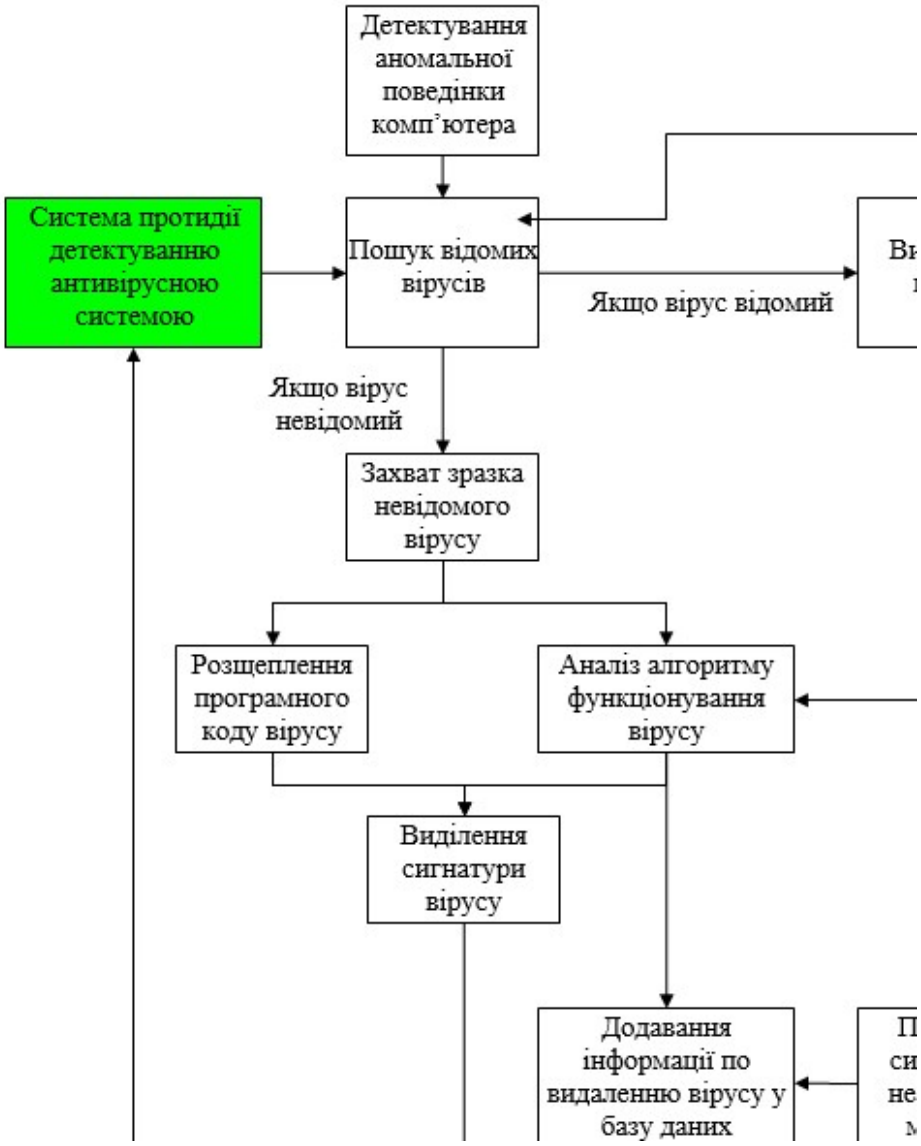

Рисунок 1 – Структурна схема системи

Потім виділяємо всі змінені й вставлені інструкції, натискаємо праву кнопку миші й копіюємо у виконується файл, що. Ця операція повинна бути пророблена з кожною модифікованою інструкцією. Коду всі інструкції скопійовані й збережені у файл, закриваємо відладник і тестуємо нашу творчість. Якщо виконання проходить без помилок, бекдор готовий до вживання.

У завершенні рекомендується змінити контрольну суму файлу, що вийшов, щоб зберегти автентичність і не викликати зайві підозри.

Якщо всі описані методи використані правильно, готовий бекдор буде повністю непомітним. На закінчення будуть наведені контрзаходи для захисту від описаних раніше технік. Це методи будуть корисні адміністраторам, аналітикам вірусів і розроблювачам антивірусів.

Контроль над привілеями секцій

Коли мова заходить про заражені файли, у першу чергу варто думати про детектуванні аномалій, пов'язаних із привілеями секцій. За замовчуванням компілятори ніколи не встановлюють повні права на секції, якщо тільки програміст не виставив спеціальні налаштування. Особливо не повинні мати привілеїв на виконання секції даних: .data i.rdata. Крім того, секції коду (наприклад, .text) не повинні мати прав на запис. Подібні аномалії, пов'язані зі зміною привілеїв, варто розглядати як підозріле поводження.

Присутність нестандартних секцій

Якщо програміст не вносить зміни в конфігурацію, компілятори звичайно створюють 5-6 стандартних типів секцій. В усі продукти, пов'язані з безпекою, повинен бути впроваджений механізм виявлення нестандартних і підозрілих секцій. Даний механізм повинен стежити за ентропією й вирівнюємо даних усередині секції. Якщо секція містить високу ентропію й нестандартно впорядковану інформацію, це ще одна підозріла ознака.

Перевірка сигнатур

Незважаючи на те, що ця техніка стара як світ, метод є дуже ефективним при перевірці файлів, що завантажуються з інтернету. Перевірка сигнатури sha1 – один з найбільш надійних способів уникнути зараження системи.

Перевірка контрольної суми файлу

Якщо є розходження між контрольною суму усередині заголовка образу й поточною контрольною сумою файлу, це означає, що файл був змінений. У системи безпеки варто впровадити механізм перевірки автентичності файлу за допомогою порівняння поточної контрольної суми й контрольної суми заголовка образу.

Висновки. У статті наведені теоретичне узагальнення й рішення наукового завдання дослідження методів протидії детектуванню антивірусною системою за допомогою штучного інтелекту. Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем протидії детектуванню антивірусною системою за допомогою штучного інтелекту.

– Досліджена система протидії детектуванню антивірусною системою за допомогою штучного інтелекту.

– На основі отриманих результатів досліджень створена програмна реалізація системи протидії детектуванню антивірусною системою за допомогою штучного інтелекту.

Розроблені алгоритми дозволяють успішно вирішувати завдання протидії детектуванню антивірусною системою за допомогою штучного інтелекту. 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. Вінтенко Б.Ю., Миронець І.В., Смірнов О.А. «Модель комп’ютерно-орієнтованої процедури системи підтримки оперативного персоналу АЕС». IV Міжнародна науково-практична Інтернет-конференція «Інновації та перспективні шляхи розвитку інформаційних технологій (ІПШРІТ-2025)» м.Черкаси 25 листопада 2025 року – Черкаси: ЧДТУ.– 2025. – С.101-103.
2. Kuznetsov O., Frontoni E., Kuznetsova Y., Chevardin V., Smirnov O. «Architectural foundations for adaptive security in edge computing systems». Cybersecurity Defensive Walls in Edge Computing, 2025. pp. 21-61.
3. Вінтенко, Б.Ю., Миронець, І.В., Смірнов, О.А., Коваленко, О.В., Усік, П.С., Буравченко, К.О., Лисенко, І.А. «Логіко-структурна модель комп'ютерно-орієнтованої процедури системи підтримки оперативного персоналу АЕС». Кібербезпека: освіта, наука, техніка. 2025. Том 2 № 30. С. 413-427, 2025.
4. Смірнова, Т.В. «Дослідження методів, моделей та сучасних ІТ-рішень для підтримки технологічних процесів у критичній інфраструктурі держави». Кібербезпека: освіта, наука, техніка. 2025. Том 2 № 30. С.195-208, 2025.
5. Усік, П.С., Смірнова, Т.В., Буравченко, К.О., Смірнов, О.А., Улічев, О.С., Смірнов, С.А. «Дослідження технологій забезпечення кібербезпеки банківських систем з використанням штучного інтелекту». Кібербезпека: освіта, наука, техніка. 2025. Том 1 № 29. С.704–716, 2025
6. Kuznetsov, O., Frontoni, E., Kuznetsova, K., Arnesano, M., Smirnov, O. «A secure biometric authentication architecture for blockchain-driven cyber-physical systems». Security and Privacy of Cyber Physical Systems Emerging Trends Technologies and Applications, 2025, pp. 193–224.
7. Kuznetsov, O., Smirnov, O., Akhmetov, B., Alimseitova, Z., Imoize, A.L. «Deep Learning Frontiers in Copy-Move Forgery Detection: Advances, Challenges, and Future Directions». Advancements in Cybersecurity Next Generation Systems and Applications, 2025. 202-229.
8. Вінтенко Б., Смірнов О., Миронець І., Смірнова Т., Смірнов С. «Імітаційна модель шляхів вхідних даних комп’ютерної інтелектуальної системи підтримки оператора енергоблоку АЕС». Комбінаторні конфігурації та їхні застосування: Матеріали XXVII Міжнародного науково-практичного семінару, присвяченого 125-річчю Національного університету «Запорізька політехніка» (Запоріжжя-Кропивницький-Київ, 4-6 червня 2025 р.). Запоріжжя: НУ «Запорізька політехніка», 2025. С.82-91.
9. Al-Azzeh, J., Ayyoub, B., Mesleh, A., Smirnova, T., Gnatyuk, S., Drieiev, O., Smirnov, O., Dorenskyi, O. «Cloud-

Based Information System for Evaluating Caverns in the Process of Blasting Metal Surfaces of Details». International Review on Modelling and Simulations 18 (1), 2025. pp. 32-42.
10. Вінтенко Б.Ю., Смірнов О.А., Миронець І.В., Смірнова Т.В., Коваленко О.В., Мацуй А.М. «Модель шляхів отримання вхідних даних комп'ютерної інтелектуальної системи підтримки оперативного персоналу АЕС». Центральноукраїнський науковий вісник. Технічні науки. 2025. Вип. 11(42), ч. II. С.52-62.
11. Вінтенко Б.Ю., Смірнов О.А., Миронець І.В., Смірнова Т.В. «Методи забезпечення відмовостійкості інтелектуальних систем підтримки оператора». VIII міжнародна науково-практична конференція "Інформаційна безпека та комп'ютерні технології", м. Кропивницький. 24-25 квітня 2025 р. – Кропивницький: ЦНТУ. – 2025. – С. 44-46.
12. Смірнов, О.А., Константинова, Л.В., Конопліцька-Слободенюк, О.К., Козірова, Н.В, Якименко, Н.М., Доренський, О.П., Буравченко, К.О. «Дослідження інструментів штучного інтелекту для роботи з базами даних та аналізу даних». Кібербезпека: освіта, наука, техніка. 2025. №3(27), С. 429–448.
13. Lakhno, V., Malyukov, V., Smirnov, O., Bebeshko, B., Chubaievskiy, V., Zhumadilova, M., Malyukova, I., Smirnov, S. «Multifactorial Model for Targeted Attacks Counteracting Within the Framework of a Multi-Step Quality Game with Fuzzy Information». 8th International Symposium on Intelligent Informatics, ISI 2023, 2025. vol 389. pp 377-389. Springer, Singapore.
14. Smirnov O., Fedorov E., Neskorodieva A., Neskorodieva T. «Intellectual Classification method of Gymnastic Elements Based on Combinations of Descriptive and Generative Approache». CEUR Workshop Proceedings Volume 3664, 2024, Pages 11-23.
15. Kuznetsov, O., Kryvinska, N., Ilchenko, O., Smirnova, T., Ulianovska, Y. «Comparative Analysis of Cryptocurrency Trading Platforms Using the Analytic Hierarchy Process». CEUR Workshop Proceedings, 2023, 3628, pp. 106-115.
16. Malyukov V., Bebeshko B., Lakhno V., Smirnov O., Malyukova I., Mohylnyi H. «Managing the Purchase-Sale Process of Digital Currencies Under Fuzzy Conditions». Lecture Notes in Networks and Systems, 2023, 729 LNNS, pp. 104–112.
17. Al-Mudhafar Aqeel, A.M., Smirnova, T., Buravchenko, K., Smirnov, O. «The method of assessing and improving the user experience of subscribers in software-configured networks based on the use of machine learning». Advanced Information Systems, 2023, 7(2), pp. 49-56.
18. Smirnov, O., Sydorenko, V., Aleksander, M., Zhyharevych, O., Yenchev, S. «Simulation of the cloud IoT-based monitoring system for critical infrastructures». CEUR Workshop Proceedings, Volume 3530, 2023, pp. 256-265.
19. Аль-Мудхафар Акіл Абдулхуссейн М., Смірнова Т.В., Буравченко К.О., Смірнов О.А. «Метод оцінки та підвищення користувальницького досвіду абонентів в програмно-конфігурованих мережах на основі використання машинного навчання». Сучасні інформаційні системи, 2023, том 7, № 2, С. 49-56.
20. Smirnov, O., Karapetyan, A., Fedorov, E., «Creating Neural Network and Single Solution Human-Based Metaheuristic Methods of Solving the Traveling Salesman Problem». CEUR Workshop Proceedings, Volume 3312, 2022, pp. 47-58.
21. Smirnov, O., Neskorodieva, T., Fedorov, E., Rudakov, K., Neskorodieva, A. «Method Detection Audit Data Anomalies on Basis Restricted Cauchy Machine» CEUR Workshop Proceedings, Volume 3187, 2022, pp. 1-12.
22. Smirnov O., Smirnova T., Anas M. Al-Oraiqat, Drieiev O., Polishchuk L., Sheroz Khan, Yassin M. Y. Hasan, Aladdein M. Amro, Hazim S. AlRawashdeh «Method for Determining Treated Metal Surface Quality Using Computer Vision Technology». Sensors (Basel, Switzerland) Volume 22, Issue 16, 6223, 2022.
23. Smirnov, O., Lakhno, V., Akhmetov, B., Chubaievskyi, V., Khorolska, K., Bebeshko, B. «Selection of a Rational Composition of İnformation Protection Means Using a Genetic Algorithm». In: Rajakumar, G., Du, KL., Vuppalapati, C., Beligiannis, G.N. (eds) Intelligent Communication Technologies and Virtual Mobile Networks. Lecture Notes on Data Engineering and Communications Technologies, vol 131. 2023. Springer, Singapore. pp. 21-34.
24. Kuznetsov, A., Oleshko, I., Chernov, K., Bagmut, M., Smirnova, T. «Biometric authentication using convolutional neural networks». Lecture Notes in Networks and Systems. Volume 152, 2021, Pages 85-98.
25. Smirnov O., Kuznetsov A., Kryvinska N., Kiian A., Kuznetsova K. «Full Non-Binary Constant-Weight Codes». SN Computer Science, Vol 2, 337, 2021. https://doi.org/10.1007/s42979-021-00739-w.
26. Smirnov O., Neskorodieva T., Fedorov E., Rymar P. «Neural Network Modeling Method of Transformations Data of Audit Production with Returnable Waste». CEUR Workshop Proceedings Volume 3101, 2021, Pages 192-207.
27. Smirnov, O., Kuznetsov, A., Potii, O., Poluyanenko, N., Stelnyk, I., Mialkovsky, D. «Combining and filtering functions in the framework of nonlinear-feedback shift register». International Journal of Computing; 2020, Volume 19, Issue 2 – Research Institute for Intelligent Computer Systems – 2020. – P. 247-256.
28. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova T. «Non-binary constant weight coding technique». CEUR Workshop Proceedings. Volume 2740, 2020, Pages 102-114.
29. Smirnov O., Kuznetsov A., Kiian A., Cherep A., Kanabekova M., Chepurko I. «Testing of code-based pseudorandom number generators for post-quantum application». 2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT), Ukraine, Kyiv, May 14-18. 2020. P. 172-177.
30. 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.
31. 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.