МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
МАРІУПОЛЬСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ЕКОНОМІКО-ПРАВОВИЙ ФАКУЛЬТЕТ
КАФЕДРА СИСТЕМНОГО АНАЛІЗУ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Допустити до захисту

В.о. завідувача кафедри

_____ Мнацаканян М.С.

(підпис) *(ПІБ завідувача кафедри)*

«___»_____20__ р.

РОЗРОБКА ПРОГРАМНОГО МОДУЛЯ ВИЯВЛЕННЯ ШКІДЛИВИХ ПРОГРАМ ПІД ЧАС АНАЛІЗУ PE ФАЙЛІВ З ВИКОРИСТАННЯМ МЕТОДІВ МАШИННОГО НАВЧАННЯ

Кваліфікаційна робота
здобувача вищої освіти
першого (бакалаврського) рівня вищої освіти
освітньо-професійної програми
«____Комп'ютерні науки_____»
(назва освітньо-професійної програми)

_Романець Роман Романович _
(прізвище, імя, по батькові здобувача вищої освіти)

Науковий керівник:
_Міщенко А.В., д.т.н., професор__
(прізвище, ініціали, науковий ступінь, вчене звання)

Рецензент:
___Лукашенко В.В., к.т.н., доцент_
(прізвище, ініціали, науковий ступінь, вчене звання, місце роботи)

Кваліфікаційна робота захищена з
оцінкою_____
Секретар ЕК_____
«____» _____20__ р.

Київ -2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
МАРІУПОЛЬСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ЕКОНОМІКО-ПРАВОВИЙ ФАКУЛЬТЕТ
КАФЕДРА СИСТЕМНОГО АНАЛІЗУ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Рівень вищої освіти *бакалавр*
Шифр та назва спеціальності *122 «Комп'ютерні науки»*
Освітньо-професійна програма *«Комп'ютерні науки»*

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри к.т.н.
(науковий ступінь, вчене звання)
_____ Мнацаканян М.С.
(підпис) *(ПІБ завідувача кафедри)*
«__» _____ 20__ р.

ПЛАН ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Романець Роман Романович
(прізвище, ім'я, по батькові)

1. Тема роботи: «Розробка програмного модуля виявлення шкідливих програм під час аналізу PE файлів з використанням методів машинного навчання»
керівник роботи_ Міщенко А.В., д.т.н., професор_____,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом Маріупольського державного університету від «___» _____20__ р. №__
2. Строк подання здобувачем роботи 30.05.2023
3. Вихідні дані до роботи (мета, об'єкт, предмет) Мета: створення програмного модуля для виявлення шкідливих програм під час аналізу PE файлів з використанням методів машинного навчання; Об'єкт: дослідження алгоритмів машинного навчання, методів детекту шкідливих файлів, PE файлів;Предмет: двигун для детектування шкідливих програм, який аналізує PE файли з використанням машинного навчання.

4. Зміст роботи

РОЗДІЛ 1. АНАЛІЗ СТРУКТУРИ PE ФАЙЛІВ

РОЗДІЛ 2. АЛГОРИТМИ МАШИННОГО НАВЧАННЯ

РОЗДІЛ 3. ТЕСТУВАННЯ ДЕТЕКТУ ШКІДЛИВИХ ПРОГРАМ

5.Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консьутанта	Підпис, дата	
		Завдання видав	Завдання прийняв

6.Дата видачі завдання 01.03.2023

Календарний план

№	Назва етапів кваліфікаційної роботи	Срок виконання етапів роботи	Примітка
1	Розробка концепції додатку	25.03.2023	
2	Знаходження інформації	02.04.2023	
3	Побудова додатку	29.04.2023	
4	Оформлення роботи	12.05.2023	
5	Розробка презентації	21.05.2023	

Здобувач _____ Романець Р.Р.

(підпис) (прізвища та ініціали)

Науковий керівник роботи _____ Міщенко А.В.

(підпис) (прізвища та ініціали)

ЗМІСТ

ВСТУП

Сьогодні комп'ютерні програми та додатки розробляються з великою швидкістю. З розвитком комп'ютерної техніки та інтернету, все більше людей починають турбуватися про їхню безпеку, безпеку їхніх даних. Розвиток шкідливого програмного забезпечення призводить до розвитку більш сучасних механізмів захисту. За дослідженням Symantec більше 44.5 мільйонів шкідливих програм було створено за травень 2015[1]. За 2017 рік світова громадськість познайомилася з WannaCry, Petya, NotPetya. Аналіз такої кількості даних вимагає від антивірусних компаній дедалі більших зусиль. Віруси стає дедалі складніше виявити. Сучасне шкідливе ПЗ здатне як впроваджуватись і заражати чисті файли системи та інших програм, так і самостійно поширюватись та змінювати своє тіло виконання у процесі життєдіяльності. Застосовуються різні способи приховати шкідливий код. Використовуються інструменти шифрування секцій, коду, даних. Шкідливе програмне забезпечення може не тільки вкрасти або пошкодити дані користувача, вимагати гроші, уповільнити роботу комп'ютера, але і перетворити комп'ютер користувача на шпигуна, заволодіти його керуванням.

Традиційний підхід до виявлення шкідливих програм базується на зіставленні сигнатур досліджуваних файлів[2].Процедура полягає в наступному:

1. Новий вірус/шкідливе ПЗ починає поширюватися
2. Експерти антивірусних компаній отримують зразки для дослідження поведення вірусу
3. Експерти надають вірусу унікальну сигнатуру, що є послідовністю інструкцій.
4. Сигнатура додається до бази даних сигнатур шкідливих програм
5. Клієнти повідомляються про оновлення бази сигнатур

6. Клієнти оновлюють їх бази сигнатур, таким чином стаючи захищеними від цього виду шкідливого ПЗ

Слід зазначити, що сигнатури дозволяють гарантовано визначити тип вірусу. Це дозволяє додатково вносити до бази сигнатур способи лікування заражених файлів, вірусів.

Даний підхід при всій своїй простоті має ключові недоліками:

1. Можливість захищати клієнта лише від відомих вірусів. Не можна по-отримати сигнатуру абсолютно нового вірусу, не дослідивши його. Тут варто обмовитися, що сигнатури, як правило, створюються так, щоб покривати не один, а якнайбільше, сімейство вірусів. Однак завжди існує така зміна тіла вірусу, при якому сигнатура перестає виявлятися

2. Постійне зростання бази даних сигнатур. Зі збільшенням кількості вірусів, їх сімейств, а також здатності вірусів змінюватись збільшується і швидкість наповнення бази

3. При появі вірусу та до оновлення бази сигнатур клієнт вразливий для нової шкідливої програми. Тільки визначивши досліджуваний файл як вірус можна отримати його сигнатуру та додати до бази

Більше того, розробники вірусів навчилися успішно обходити пошук сигнатур, обфусціруя тіло вірусу. Поліморфні та метаморфічні шкідливі програми змінюють свій зовнішній вигляд у процесі життєдіяльності. Все це спонукало антивірусні компанії розробляти альтернативні способи захисту. А саме, можна виділити 2 великі напрямки досліджень[3]:

1. Статичний аналіз (аналізування структури бінарного файлу, його атрибутів, логічних структур, потоку виконання та даних)

2. Динамічний аналіз (відстеження дій програми під час виконання, побудова її профілю)

Кожен із методів має свої переваги та недоліки. Так, як правило, для кращого детектування шкідливих програм вони використовуються одночасно. У кожному з цих методів існує можливість помилково визначити наявність у файлі вірусу, коли насправді файл чистий. У таких випадках передбачуване лікування може зіпсувати файл, що спричинить втрату інформації.

Динамічний аналіз дозволяє оминути обфускацію бінарного файлу. Так, наприклад, вірусописувачі широко використовують системи упаковки, шифрування коду та даних, обфускацію функцій та потоку управління. Але ті ж самі методи використовуються і для створення додатків розробниками, щоб захистити інтелектуальну власність, ускладнити реверс інжиніринг. Метод виділяє кілька основних дій, таких як видалення файлу, запис у файл, спілкування з мережею, відкриття порту на прослуховування, розсилання листів та інші. Цей профіль файлу, його діяльності вивчається експертом або методами машинного навчання для винесення вердикту про шкідливість зразка. Тим не менш, динамічний аналіз можливий лише при виконанні досліджуваного коду, що робить операційну систему більш вразливою, а також деякі віруси можуть визначати оточення, що запускається. маскуючи себе, і, тим самим, поводяться по-різному в тестовому та робочому оточенні. Таким чином, потрібно створювати максимально схожі оточення, що є ще однією проблемою, яка потребує вирішення.

Статичний аналіз здатний доповнювати динамічний, надаючи інформацію про атрибути бінарного файлу. Статичний метод аналізує програму до виконання, витягує атрибути з бінарного файлу, підраховує статистики та на основі цієї інформації виносить вердикт про загрозу досліджуваного файлу. Такий підхід є безпечним - вердикт виноситься ще до виконання файлу, але погано працює на файлах з обфускаціями, запакованими секціями. Також зі збільшенням розміру файлу час, необхідний для аналізу, збільшується. Крім цього, для розробки якісного статичного аналізатора потрібно розуміти роботу завантажувача бінарного файлу. Різницю між документацією та дійсною поведінкою завантажувача. Віруси можуть використовувати частину полів бінарного файлу для своїх потреб. Наприклад, у вигляді зберігання даних або адреси виконання шкідливого коду.

Далі ми зупинимося на статичному аналізі PE (Portable Executable) файли для операційної системи Windows. Вибір пов'язаний із величезною популярністю цієї системи. PE формат є стандартом виконуваних файлів і бібліотек. Потрібно розуміти, що шаблон, алгоритм проектування движка статичного аналізу шкідливого ПЗ застосовний і до інших платформ.

Завдання полягає у створенні движка машинного навчання для статичного аналізу PE файлів для операційної системи Windows з подальшою інтеграцією до програмного продукту. Також важливо відзначити, що двигун машинного навчання повинен мати високу швидкість роботи. Для слабких персональних комп'ютерів час сканування системного диска повинен утримуватися в кілька хвилин. Для вирішення цього завдання потрібно:

1. досліджувати існуючі підходи
2. виявити найбільш важливі ознаки для визначення шкідливості файлу в умовах обмежених ресурсів та часу
3. порівняти сучасні алгоритми машинного навчання стосовно даної задачі
4. продати двигун у рамках програмного продукту

Дана робота відповідає на питання про можливість використання сучасних-методів машинного навчання для ефективного детектування шкідливого ПЗ у реальних умовах. Реалізується двигун статичного аналізу, що має практичне застосування та є модулем системи захисту машин кінцевого користувача.

РОЗДІЛ 1. АНАЛІЗ СТРУКТУРИ PE ФАЙЛІВ

1.1 Постановка задачі

Завдання полягає у створенні статичного движка машинного навчання під час аналізу PE (Portable Executable) файлів з використанням методів машинного навчання, що дозволяє з високою точністю виносити вердикт про шкідливість файлу.

Перед тим як вирішувати завдання, сформулюємоїї формально. А саме, обговоримо вимоги та обмеження, що пред'являються до рішення, вхідні та вихідні дані та метрики якості, за якими визначатиметься найліпше рішення.

У роботі обмежуємося розглядом файлів для операційної системи Windows, а саме файлів PE формату. Алгоритм дій для пошуку кращої моделі для інших файлів, файлів інших систем буде тим самим. Вибір зроблений з урахуванням популярності операційної системи Windows серед інших систем. А всі файли Windows, що виконуються, мають PE формат. Статичний аналіз файлу заснований лише на вивченні структури його файлу, у нашому випадку, на вивченні PE структури та витягу корисних для детекції ознак. Виконання коду з метою неприпустимо. Під час запуску досліджуваної програми надійність системи знижується. Частка хибно-позитивних спрацьовувань має бути якнайменшою. Двигун якнайрідше повинен помилятися на визначенні чистих файлів як шкідливих. Це може призвести до втрати або пошкодження даних користувача. Будемо рахувати кількість помилок у 3% прийнятною. Найбільш складною характеристикою вимог визначення є точність (accuracy). Точність алгоритму має бути якомога вищою, з іншого боку нам також важлива швидкість алгоритму. Користувач не повинен відчувати незручності при використанні антивірусного захисту. Отже нам потрібно знайти прийнятні параметри для якості та швидкості. Також, досить складно зіставити хоча б приблизно результати більшості досліджень. Немає загальнодоступної великої бази файлів, яку можна було б зразком. Кожне дослідження проводиться зі своїми даними. Також треба розуміти, що немає сенсу порівнювати алгоритми машинного навчання та сигнатурний метод на тих самих даних. З великою ймовірністю файли застаріли та їх сигнатури вже є в базах. Виберемо поріг якості в 95% як мінімально допустимий, при котором алгоритм навіть не буде розглядатися.

Швидкість двигуна підберемо із розрахунку повного сканування системного жорсткого диска HDD. На нову систему налічується близько 20-30 тисяч PE-файлів. Тому виберемо прийнятну середню швидкість роботи алгоритму, яка дорівнює 30 файлам на секунду. Тепер ми можемо сформулювати загальні вимоги:

1. Джерелом даних для моделі є PE файл
2. Можлива лише робота з вихідним бінарним файлом без його виконання
3. Точність (accuracy) передбачення вище 95%
4. Частка хибно позитивних спрацьовувань менше 3%
5. Середня швидкість роботи алгоритму - 30 файлів за секунду

Двигун на вхід приймає PE-файл і повертає вердикт про шкідливість файлу:

Вхід:PE-файл

Вихід: вердикт (0 – файл чистий, 1 – шкідливий)

Двигун являє собою pipeline. Вхідні дані проходять кілька етапів обробки і на виході отримуємо вердикт про файл.

Виділимо компоненти з яких складається двигун:

1. PE-парсер, що дозволяє отримати значущі для аналізу атрибути
2. Класифікатор, який приймає на вхід атрибути досліджуваного PE-файлу і відносить його до одного з класів (0 - для чистих файлів і 1 - для шкідливих)

Хоча в даній роботі не розглядається питання створення PE-парсера, він нерозривно пов'язаний із класифікатором і також є компонентом двигуна. Класифікатор диктує набір значимих аналізу атрибут. Схема движка представлена на Рис.1.Взагалі кажучи, не всі файли, що надходять на вхід PE-парсера, можуть бути PE-файлами. Файли можуть бути битими або файлами формату Portable Executable. В реальних умовах PE-парсер повідомляє про помилку та ознаки таких файлів не передаються до класифікатора. Тому для простоти вважатимемо, що всі передані на вхід

PE-парсера файли мають формат Portable Executable і їх атрибути передаються в Attribute Selector.

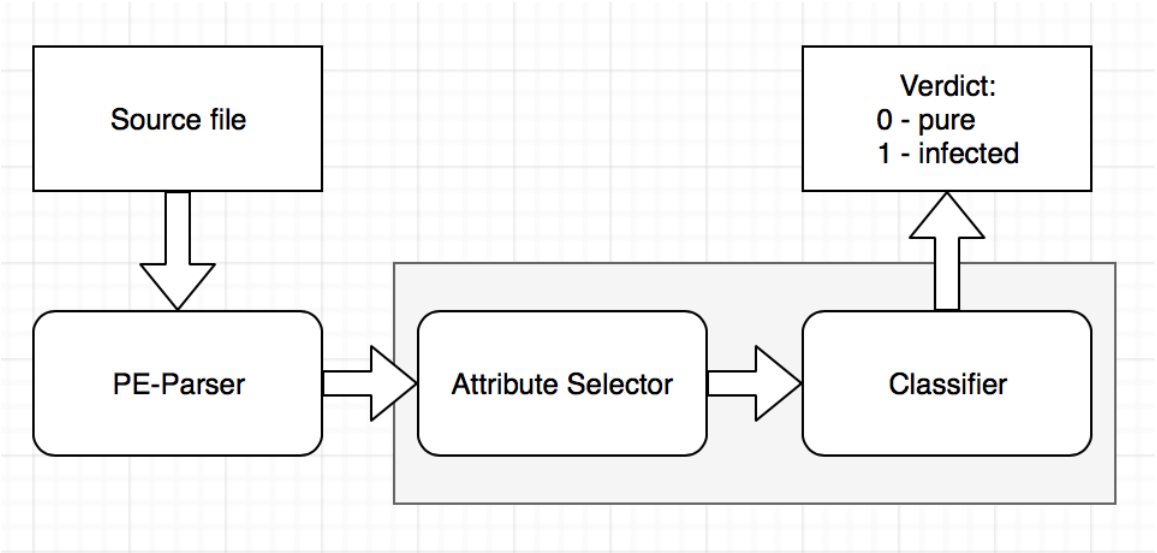


Рисунок 1 -Загальна архітектура двигуна з аналізу PE файлів. Сіра область - фокус досліджень у цій роботі

Як було згадано. якість моделі оцінюємо метрикою accuracy - відношення кількості правильно класифікованих файлів до всіх файлів. Це буде нашою основною метрикою. Також, при оцінці та порівнянні якості моделей варто звертати увагу і на false positive rate. Метрика accuracy при нерівномірній вибірці може не відображати дійсну здатність класифікації алгоритму, тоді як разом із false positive rate дає більш повну картину. Метрика accuracy основна в тому сенсі, що вона є визначальною при виборі серед алгоритмів, у яких false positive rate менша від порогового значення.

Вимога на швидкість двигуна сформульована таким чином, що можно говорити лише про середню його швидкість роботи та середній час обробки файлу. Це з залежністю часу роботи модуля від розміру файла. Вимога високої швидкості роботи двигуна і, одночасно з цим, хорошої точності передбачення змушує розумно підходити до вибору атрибутів та їх кількості для вилучення з PE файлу, а також до вибору алгоритму машинного навчання, тому принципово не будуть розглядатися нейронні мережі та методи глибинного навчання. Це вимагає, з одного боку, зчитування більшої кількості

байт файлу, а з іншого, використання складніших моделей, перемноження матриць, обчислення нелінійностей, що буде на порядок повільніше підходів, представлених у даній роботі.

1.2 Огляд існуючих рішень

Більшість статичних аналізаторів ґрунтуються на вилученні структури PE (Portable Executable) файлу, як у деревоподібному вигляді, так і у вигляді сукупності полів. Використовуються як ознаки, одержувані з машинного коду, викликів функцій, і ознаки, одержувані з розгляду файлу як послідовності байт, визначення статистик, ентропій, підрахунок n-грам.

Мета роботи[4]полягала у створенні відмовостійкого PE парсера - PortEX, здатного також виявляти аномалії у структурах бінарних файлів. Було проведено дослідження стратегій зараження файлів, виділено найімовірніші атрибути чистих файлів. Також проводилася оцінка залежності ентропії секцій PE файлу та ймовірності того, що файл шкідливий. Для отримання ймовірності того, що файл шкідливий застосовувалися евристики. Були визначені:

1. Бустери (шаблони, які вказують на поведінку або зовнішній вигляд шкідливого ПЗ). Збільшують ймовірність того, що файл буде ідентифікований як шкідливий

2. Стопери (шаблони, які вказують на поведінку або зовнішній вигляд, що є нетиповим для шкідливого ПЗ). Зменшують імовірність того, що файл буде визначений як шкідливий

Обчислювалися сумарний бустер і сумарний стопер на основі евристик і статистики за зібраними файлами і на основі значення, що вийшло, видавалася ймовірність шкідливості файлу. Ентропія секцій PE файлу допомоглавизначати шкідливість файлу. Кордони та визначення бустерів і стоперів є евристики, засновані на статистиці за багатьма файлами, які можна поліпшити методами машинного навчання.

У[2] аналіз будується на вилученні 197 атрибутів із бінарного файлу. Атрибути є або характер присутності (DLLs referred, APIs referred), або значення полів структур файлу. Виділення значних атрибут відбувається на основі статистичних тестів (t-тест, $\chi 2$-тест). Використовуючи відібрані 19-20 ознак, навчають моделі бустингу, випадкового лісу. Відсутній глибший аналіз PE-формату, розгляд ділянок файлу в вигляді послідовності байт.

Також немає інформації про швидкість роботи алгоритму.

Достатньо повна картина вибору важливих атрибут PE файлу наведена в[5].Наведено таблицю (Таблиця 1) ознак, що використовуються в попередніх роботах на цю тематику. Ознаки – ознаки, витягнуті з PE файлу. Або із заголовка, або з тіла. У колонці «Структура» показано додаткові ознаки, які збираються з файлу. Додатково введено скорочення наступних скорочень: API: Application Programming Interface, BYT: Byte code, FC: Function Call, STC: Structural features, OP: Operation code

Таблиця 1 Методи статичного аналізу з типами видобутих ознак

Рік	Стаття	Тип	Ознаки заголовка	Ознаки тіла	Структура
2008	Ye et al.[6]	детекція	API	-	Itemset
2009	PE-Miner[7]	детекція	STC	STC	-
2009	Tabish та ін.[8]	детекція	BYT	BYT	N-gram
2009	Griffin та ін.[9]	детекція	BYT	BYT	Sequence
2009	Hu та ін.[10]	детекція	-	FC	Graph
2010	Sami та ін.[11]	детекція	API	-	Itemset
2011	Natara j et al. [12]	класифікація	BYT	BYT	-
2012	Jacob та ін.[13]	класифікація	STC	BYT	N-gram
2013	Santos та ін.[14]	детекція	-	OP	Sequence
2014	Nissim та ін.[15]	детекція	BYT	BYT	N-gram
2015	DLLMiner[16]	детекція	DLL	-	Tree

Завдання стояло у класифікації шкідливих програм на типи (malwarefamily). Стаття підготовлена на основі конкурсу kaggle, де учасникам Microsoft надала 21741 файлів без заголовка PE файлу (PE header) для класифікації за 9 типами. У дослідженні основний акцент був зроблений на виборі кращого набору атрибут. Вибір проводився на основі полів структур PE файлу, і розглядаючи файл як послідовності байт. Більш конкретно, наведемо класифікацію ознак, що розглядаються у статтях:

1. Hex dump-based features (Ознаки, витягнуті з послідовності байт файлу)
   (a) N-грам. Послідовність N байт. 1-gram - частота байтів і була обрана у статті
   (b) Метадані. Наприклад, розмір файлу
   (c) Ентропія. Підрахунок ентропії всього файлу або методом ковзного вікна
   (d) Подання файлу як зображення[12]
   (e) Довжина рядків. Вилучення довжин можливих рядків ASCII символів з файлу
2. Features extracted from disassembledfiles (Вилучення ознак із структур PE файлу)
   (a) Метадані. Наприклад, кількість рядків у файлі
   (b) Частота спеціальних символів. Частота символів: −, +, ∗ ,], [, ?, @
   (c) Коди операцій. Підрахунок статистики з асемблерних інструкцій
   (d) Реєстри. Підрахунок статистики за реєстрами, що використовуються
   (e) Application Programming Interface. Перевірка присутності/відсутності певного набору API дзвінків
   (f) секції. Статистика з секцій
   (g) Встановлення байт. Статистика з db, dw та dd інструкцій

Така класифікація дозволяє повною мірою побачити дослідження в даній галузі з вилучення ознак та поточні результати. Зауважимо, що частина цих ознак не доступна нашому дослідженні. Підрахунок статистикиза довгою послідовністю байт є дорогою операцією. А саме, підрахунок статистики по всьому файлу або подання файлу як зображення уникатимемо.

Також були оригінальні ідеї з інших статей, а саме представлення файлу у вигляді зображення[12].У статті наведено графік важливості при-

знаків визначення класу шкідливого ПЗ. В якості алгоритму використаний бустинг на вирішальних деревах (XGBoost)[17],а також для отримання кращого результату в конкурсі бегінг (bagging)[18].У реальних умовах у нас доступна інформація з PE header. Також, цікаво дослідити значущість атрибут у задачі визначення шкідливого ПЗ.

Для повноти картини наведемо короткий опис оригінального дослідження[13],в якому PE файл представлявся у вигляді чорно-білого зображення. Завдання стояло у визначенні типу (malware family) шкідливого ПЗ. Послідовність байт представляли як матриці (0: чорний, 1: білий). Ширина зображення фіксувалася в залежності від розміру файлу (32 <10kB, 64, 128, ..., 1024 -> 1000kB). На основі зображень було визначено загальні патерни, текстури для 25 типів шкідливих програм. Для аналізу текстури використовувався фільтр Габора. Ознаки, отримані з зображень, використовували в класифікаторі — метод k-найближчих сусідів із метрика евкліду. Хоч дослідження й цікаве, але з наших вимог, а саме, суворого обмеження часу роботи двигуна, ми змушені відмовитися від цього виду ознак.

Згадаймо про існування бібліотек, націлених на боротьбу з деофу-кацією коду. Один із таких інструментів FLOSS. FLOSS комбінує кілька підходів статичного аналізу для пошуку обфусцированных ASCII рядків в аналізованому файлі. Зокрема, аналізує потік управління для аналізу файлу на функції, базові блоки. Використовує евристики для пошукудекодуючих функцій. Емулює поведінку функцій декодування для вилучення читаних ASCII рядків. Такий інструментарій покликаний допомогти статичному аналізу із зашифрованими файлами та отримати більше інформації про них. Використовуючи FLOSS, наприклад, можна отримати список імпортованих api та бібліотек. У цій роботі ми не будемо використовувати цей інструментарій. Завдання полягає у створенні швидкого та якісного алгоритму машинного навчання. Для більш глибокого аналізу варто дослідити стійкість до відмов подібних бібліотек і порівняти існуючі інструменти деобфускації. У цій роботі це питання не висвітлюється.

Таким чином, було проведено загальний огляд методик та підходів до завдання статичного аналізу шкідливого ПЗ. Більш докладні роботи, представля ють повний огляд методів можна знайти в[19],[20].

На відміну від попередніх досліджень, мета даної роботи полягає у:

1. узагальнення отриманих результатів досліджень
2. отримання найбільш важливих атрибутів для виявлення шкідливого ПЗ на типовому для користувачів наборі файлів в умовах обмежених ресурсів і часу
3. реалізації двигуна в програмному продукті з використанням сучасних алгоритмів машинного навчання

1.3 PE формат

Для аналізу файлу, що досліджується, класифікатор отримує інформацію від PE-парсера. Знання про PE (Portable Executable) форматі необхідне для оцінки корисності інформації, що витягується, і розуміння завантаження файлу в пам'ять для виділення найбільш значущих атрибут.

Далі наводиться короткий опис PE формату, його структури як для 32 бітних, так і 64 бітних систем.

Повна специфікація доступна на сайті Microsoft[21],Тим не менш, у документації є двозначності. Потрібно розуміти, що специфікація, наведена на сайті Microsoft, має швидше оглядовий характер. Як насправді роблять завантажувачі можна спостерігати лише експериментально. У особливо хитрих випадках той самий файл може бути запущений одним завантажувачем, але, запустивши іншим, може призвести всю систему до перезавантаження. Варто пам'ятати, що завантажувач під час запуску бінарного файлу може його змінювати.

«Portable Executable (PE, переносящий виконуваний) — формат файлів, об'єктного коду та динамічних бібліотек, що виконуються, використовуваний у 32- та 64-розрядних версіях операційної системи Microsoft Windows. Формат PE є структурою даних, що містить всю інформацію, необхідну PE-завантажувачу для відображення файлу в пам'ять[23]

Всі файли PE можна розділити на файли з розширенням EXE і DLL. Файли DLL призначені для експортування функцій або даних для використання іншими програмами. Тобто вони зазвичай можуть бути запущені в контексті інших програм. Такі файли мають розширення .dll, .sys, .ocx,

.cpl, .fon, .drv[24].EXE файли запускаються у власних процесах. Вони зазвичай мають розширення .exe та не експортують символи. Незважаючи на такий умовний поділ, формат не забороняє створювати гібриди. Наприклад, файли EXE можуть експортувати символи.

Структура PE-файлу представлена на Рис. 2 .Атрибути цієї структури, і навіть статистика по послідовностям байт цієї структури використовується отримання ознак.

Будь-який PE файл починається з MS-DOS Stub'a. Залишений для сумісності і є заглушкою. Перші два байти якої мають бути рівними " MZ " . Атрибут e_lfanew, є зміщення PE File заголовка щодо початку файлу і вказує на сигнатуру "PEx0x0". Не обов'язково, щоб e_lfanew вказувало область відразу після MS-DOS Stub, що також відображено на Рис. 2- Невикористана область. Секції розташовуються відразу після PE File заголовка.

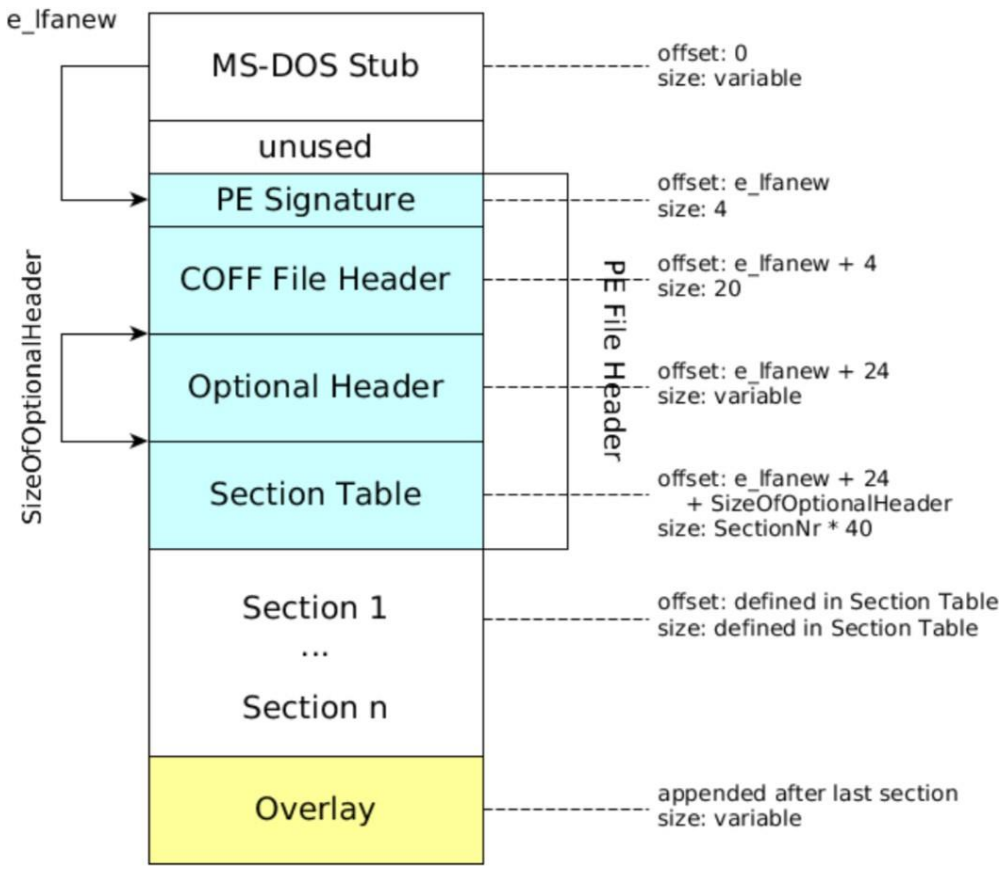


Рисунок 2 –Структура PE файлу

Після PE сигнатури "PEx0x0" у PE File Header розташовується COFF File Header. Має фіксований розмір. Містить загальну інформацію про файл, таку як: тип цільової машини (x64, ARM, MIPS і так далі), кількість секцій — NumberOfSections, дата створення файлу, розмір опціонального заряду. вправність - SizeOfOptionalHeader. Атрибути всього файлу - Characteristics (саме тут містяться прапори – чи є файл EXE чи DLL). Практично всі атрибути цієї структури слід розглядати під час детектування вірусів, зараження файлу. Виключимо з розгляду дату створення файлу - він не повинен впливати на ймовірність зараження, але вибірка може виявитися сміченої за цим параметром.

Далі йде опціональний заголовок (Optional Header). Хоча ця структура має таку назву, її присутність обов'язково для завантаження файлу. Структура містить інформацію про файл. Таку, наприклад, як 32-х або 64-х бітний адресний простір. Сумарний розмір секцій коду, ініціалізованих та неініціалізованих даних (згідно з[22]ці поля ніким не перевіряються так само як і відносні базові адреси кодової секції та секції даних). Деякі антивіруси мають евристику на значення адреси точки входу — AddressOfEntryPoint. Передбачається, що точка входу розташовується у першій секції файлу (зазвичай .text секція). Базова адреса завантаження програми (ImageBase) повинна бути кратна 64кб. Також, для аналізу корисні вирівнювання - FileAlignment, SectionAlignment, а точніше поля PE файлу, від яких потрібно вирівнювання. Також в опціональному заголовку міститься каталог даних (DataDirectory), кількість елементів якої - NumberOfRvaAndSizes.

У каталозі даних (DataDirectory) кожен елемент — структура, що складається з покажчика та розміру. Кожен із записів має певну роль. Так, IMAGE_DIRECTORY_ENTRY_EXPORT — покажчик на таблицю функцій і даних, що експортуються (зустрічається в основному в DLL). Вказівник IMAGE_DIRECTORY_ENTRY_SECURITY має особливий інтерес. Він вказує на Certificate Table. Таблиця, що знаходиться на диску. При IMAGE_DIRECTORY_ENTRY_SECURITY != 0 практично виключена ймовірність того, що файл є шкідливим. Також, якщо IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR != 0, то файл є .NET додатком, що складається з байт-коду, що також знижує ймовірність шкідливості файлу.

Після опціонального заголовка йде таблиця секцій (Section Table), що містить NumberOfSections секцій. Як правило, секції мають такий порядок. Спочатку описана секція з кодом, після – кілька секцій ініціалізованих даних, а після – секція неініціалізованих даних. Кожна секція має ім'я (не відіграє ніякої ролі при завантаженні файлу), адресу початку секції у пам'яті та у файлі (вирівняні), віртуальну та фізичну

довжину секції (VirtualSize та SizeOfRawData). Віртуальні адреси секцій повинні йти поспіль, не накладаючись і не утворюючи перепусток. Поле Characteristics є правом доступу до секції та особливостями її завантаження. Варто розглянути як секції, і їх атрибути. Для детекції вірусів та аналізу також корисні поля таблиці експорту, імпорту.

Таблиця експорту потрібна для зв'язку експортованих функцій з їхніми адресами. Містить покажчики на 3 таблиці: таблиця імен, ординалів, адрес.

Існує 3 режими імпорту: стандартний, що зв'язує (bound import), відкладений (delay import). Для аналізу PE файлу варто аналізувати назви бібліотек та функцій (api). Тут постає питання про спосіб подання цієї інформації. Вектор ознак має бути фіксований, що накладає певні обмеження. З одного боку ми повинні ви-брати найпоширеніші api, з іншого - ті, які статистично значущі при відділенні шкідливого ПЗ від чистих файлів.

Виклавши загальну картину структур PE файлів, ми можемо розпочати формування атрибутів та його відбору. Стають видно такі властивості одержуваних ознак. Ознаки строго діляться за роль і значення структури з якої були вилучені:

1. чисельна ознака (кількість секцій, символів, розмір опціонального заголовка, адреса точки входу)
2. прапор присутності (секції, api таблиці імпорту)
3. значення функції від послідовності байт (ентропія секцій)

Тут ми не розглядаємо функції від порівняно довгих послідовностей байт, наприклад, ентропію всього файлу, розподіл певних символів у файлі або подання файлу у вигляді зображення. Двигун повинен швидко працювати навіть на слабких машинах, тому завдання полягає у відборі найбільш цінних з погляду визначення шкідливого ПЗ ознак.

Тут стають помітні й головні проблеми статичного аналізу. Одна з основних - зашифровані, упаковані секції. Віруси шифрують свій

код виконання так, що він більше не видно з погляду статичного аналізатора. Для вирішення цієї проблеми або додають дешифратор, або ґрунтуються на статистиці цієї секції. У першому випадку завдання постає окремим дослідженням, яке буде розглянуто тут. Більшість дешифраторів використовують евристики та перебирають відомі методи шифрування, а є ті, які дешифрують під час виконання бінарного файлу.

РОЗДІЛ 2. АЛГОРИТМИ МАШИННОГО НАВЧАННЯ

Далі будуть розглянуті алгоритми машинного навчання для задач класифікації, застосування яких найбільш виправдане в досліджуваному завданні. А саме, ми повинні враховувати різнорідну природу ознак, які одержують із бінарних файлів. Частина ознак є прапорами присутності, частина - статистики. Такі, наприклад, як ентропії секцій, а частина поля PE структур. Також ми беремо до уваги швидкість роботи алгоритму. У таких умовах варто передусім розглянути алгоритми, що базуються на вирішальних дерев (випадковий ліс, градієнтний бустинг). Існуючі дослідження в галузі детектування шкідливого програмного забезпечення також підтверджують це припущення. Опишемо аналізовані алгоритми.

2.1 Дерево рішень

Алгоритм «вирішальне дерево (decision tree) тут згадаємо лише тому, що на ньому ґрунтуються складніші, представлені тут алгоритми, і слід розуміти принцип побудови базових алгоритмів для використання їх у системі.

Дерево рішень, загалом, є способом подання правил в ієрархічній послідовній структурі, де кожному об'єкту відповідає єдиний вузол, що дає рішення.

Як правило, розглядають бінарне вирішальне дерево.

Процес пошукувузла представлений на Рис. 3 з відповідним лістингом - Алгоритм 1.

Введемо позначення:

V- Багато вузлів у дереві. βv - предикат, функція у вершині v, що повертає {0, 1}. Y - безліч класів, у нашому випадку - 2 (pure - чистий файл, infected - шкідливий).

Вибір класу виконується так:

$\forall v \in V_{\text{всередину}} \rightarrow$ предикат $\beta v: X \rightarrow \{0, 1\}, \beta \in B$

$\forall v \in V_{\text{лист}} \rightarrow$ ім'я класу cv ∈ Y

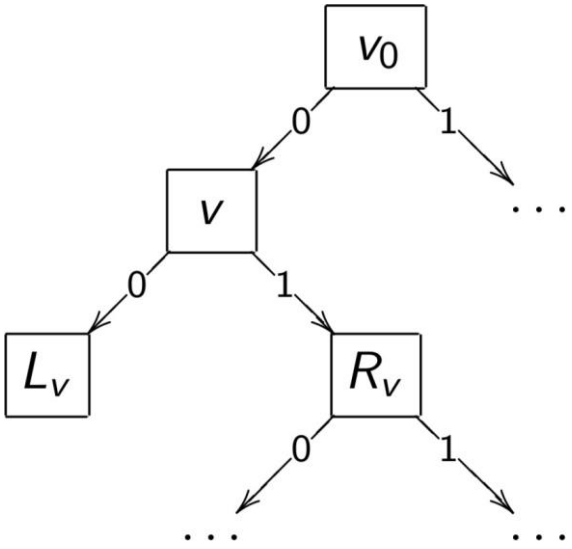


Рисунок 3 –Вирішальне дерево

while $v \in V_{\text{всередину}}$ do
    if βv(x) = 1 then
        go to right;
        v:= Rv;
        else
        go to left;
        v:= Lv;
        end return cv
        end

Алгоритм 1: Пошук вузла

Насправді використовуються одномірні предикати, які порівнюють значення однієї з ознак з порогом.

Для побудови дерева існує багато алгоритмів. Конкретний метод побудови вирішального дерева визначається:

1. видом предикатів у вершинах

функціоналом якості (для прийняття рішення про розбиття навчальної вибірки в розглянутій вершині. Як правило, використовується критерій Джині або ентропійний критерій)

критерієм зупинки

методом обробки пропущених значень

методом стрижки (видалення деяких вершин з метою зниження складності та підвищення узагальнюючої здатності)

Можливість обробляти пропущені значення є корисною і в нашій задачі. Вибрані імена секцій або імпортовані назви бібліотек, функцій можуть бути не знайдені. У такому разі для такого файлу частина значень ознак буде пропущена. Вирішальне дерево може бути навчене і таких даних. А саме (тут, U - безліч об'єктів навчання, Gain - деякий обраний критерій для максимізації в кожній вершині, Sv - функція, при 0 - Lv, 1 - Rv)

На стадії навчання:

bv(xi) не визначено => xi виключається з U для Gain(bv, U)

$q_{vk} = \frac{|Uk|}{|U|}$ - Оцінка ймовірності k-й гілки, v ∈ Vвнутр

$P(y|x, v) = \frac{1}{|U|} \sum_{x \in U} [yi = y]$ для всіх $v \in V_{лист}$

На стадії класифікації:

bv(x) визначено => з дочірньоїв s = Sv(bv(x)) взяти P (y|x, v) = P (y|x, s)

bv(x) не визначено => пропорційний розподіл:

$P(y|x, v) = \sum_{k \in D_v} qvkP(y|x, Sv(k))$

Остаточне рішення - найімовірніший клас:

$a(x) = \arg\max_{y \in Y} P(y|x, v_0)$

Для побудови дерева зазвичай використовується жадібний алгоритм з оптимізацією в кожному вузлі заданого функціоналу якості (зазвичай розглядають критерій Джині, ентропійний критерій). Вибір функціоналу якості також дуже впливає на кінцевий результат.

Як готовий алгоритм вирішальне дерево сьогодні не використовується. Алгоритм схильний до перенавчання, сильний вплив на шуми. Також, не завждидерево будує найкращий поділ (знаменитий приклад із завданням

побудови XOR функції). Рис. 4відповідний приклад.

Велика увага приділяється ансамблям дерев рішень. Кожне з дерев здатне до перенавчання, сильного впливу на шумові об'єкти, але використання їх ансамблів (бустинг, бегінг) перетворює на один з найбільш успішних і застосовних на практиці інструментів машинного навчання.

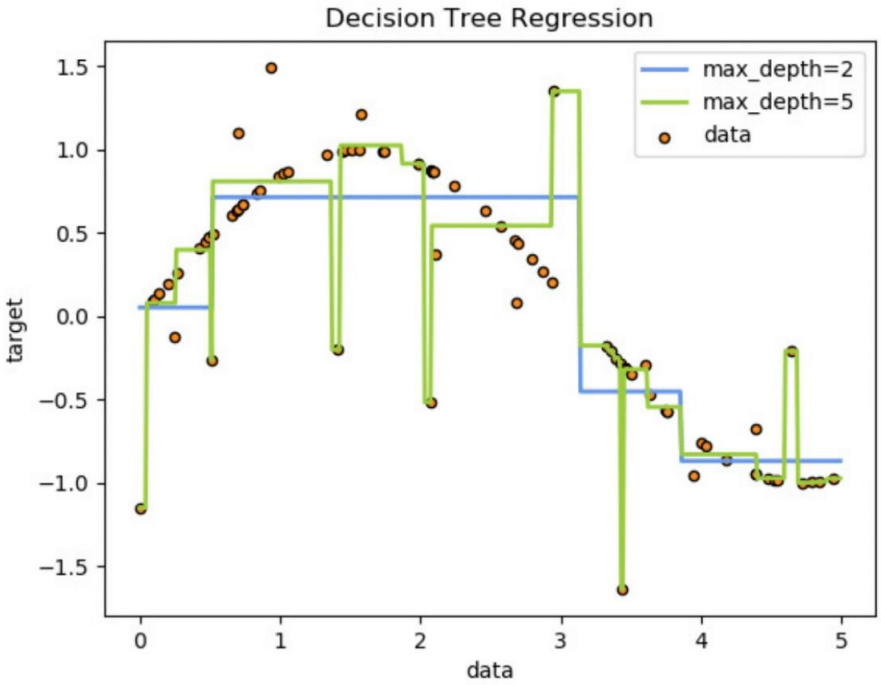


Рисунок 4 - Схильність до перенавчання вирішального дерева

Побічний ефект від побудови дерева рішень полягає у можливості оцінка важливості ознак. Важливість ознаки розраховується на підставі того, наскільки покращується обрана метрика при поділі вибірки за ознакою в черговому вузлі. Це може нам дати уявлення, які ознаки для побудови дерева виявилися найбільш вирішальними, а які не були використані. Потрібно розуміти, що важливість ознак є оцінною.

Далі будуть наведені алгоритми, що ґрунтуються на деревах рішень. Багато властивостей, вказаних тут, вірні і для ансамблів. Наприклад, ми можемо отримати важливість ознак через усереднення по кожному дереву.

2.2 Випадковий ліс

Випадковий ліс (Random forest) є ансамблем дерев рішень, які навчаються незалежно[25].Кожен із вирішальних дерев, як правило, роблять простим. Для ухвалення остаточного рішення обирається значення, за яке проголосувала більшість дерев. Кожне дерево окремо дає низьку якість, але за рахунок їх великої кількості результат виходить добрим. Рис.5 можна бачити приклад випадкового лісу із двох дерев.

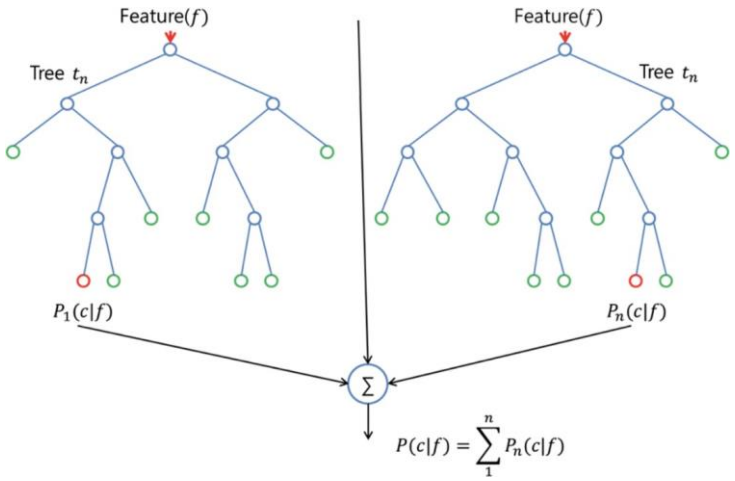


Рисунок 5 –Випадковий ліс із 2 дерев

У побудові дерев на даних є свої особливості. Вкажемо традиційний підхід у побудові дерева. Нехай N - кількість прикладів на навчанні. Тоді:

1. вибирається підвибірка навчальної вибірки розміру N (можливо, з поверненням)

2. будується дерево. Під час пошуку розбиття при побудові кожного вузла розглядаються не всі ознаки, а лише частина з них (sklearn реалізація - Корінь з кількості ознак)

3. дерево будується до повного вичерпання підвиборки, або на основі будь-яких інших критеріїв

Кожне з дерев намагаються зробити якомога простіше. При ухваленні рішення обирається клас, за який проголосувала більшість дерев рішень.

Випадковий ліс дає порівняно погані результати під час поділу вибірки у метричному просторі. У тому випадку, коли очікується деякий «правильний» вид розділяючої гіперплощини. Тим не менш, у випадкового ліса є велика кількість переваг, які можуть використовуватися в розглянутій задачі визначення шкідливості файлу. Алгоритм не оперує метриками, що дозволяє вільно працювати з ознаками різної природи. Можна обробляти дані з пропущеними значеннями ознак. Висока швидкість роботи алгоритму, незалежна побудова вирішальних дерев.

2.3 Градієнтний бустинг

При побудові бустингу базові алгоритми будуються послідовно, а не паралельно, компенсуючи помилку на попередній ітерації.[27].Загалом алгоритм навчання градієнтного бустингу представлений нижче (Алгоритм2). Тут L(a, y) - довільна функція втрат, bi - і базовий алгоритм. fi є і-е наближення.

Маємо:

Лінійну комбінацію базових алгоритмів: a(x) = ΣT $_{t=1}$ αtbt(x). Це і є алгоритм градієнтного бустингу. Уявляє лінійну комбінацію базових алгоритмів. Весь фокус полягає в тому, як побудувати такий набір алгоритмів та підібрати ваги.

Функціоналякості з довільною функцією втрат L(a, y):

$$Q(\alpha, b; Xl) = \sum_{l} \sum_{i=1}^{T-1} L\left(\sum_{t=1} \alpha tbt(xi) + \alpha b(xi), yi\right) \min_{\alpha,b}$$

f_{T-1}= ($f_{T-1,i}$)$'_{i=1}$ - поточне наближення

fT= (fT,i)$_{l_1}$ - Наступне наближення

На кожній ітерації при пошуку чергового базового алгоритму мінімізується даний функціонал Q(α, b; Xl).

Data: навчальна вибірка Xl, параметр T;

Result: базові алгоритми та їх ваги αtbt, t = 1, ..., T;

ініціалізація: fi := 0, i = 1, ..., l;

for $t \leftarrow 1$ **to** T **do**

базовий алгоритм, що наближає градієнт:

$$bt := \arg\min_{b} \sum_{i=1}^{l} (b(xi) + L'(fi, yi))2;$$

завдання одномірної мінімізації:

$$\alpha t := \arg\min_{\alpha>0} \sum_{i=1}^{l} L(fi + \alpha bt(xi), yi);$$

оновлення вектора значень на об'єктах вибірки:

$$fi := fi + \alpha tbt(xi); \; i = 1, \ldots, l;$$

end

Алгоритм 2: Алгоритм градієнтного бустингу

У разі градієнтного бустингу на вирішальних деревах, очевидно, базові алгоритми є деревами рішень. Параметр T - кількість дерев, що вибудовуються.

Найбільш популярні реалізації градієнтного бустингу на вирішальних деревах: XGBoost[17],LightGBM[27],CatBoost[28].

Xgboost представляє лише найбільш вдалу реалізацію ідей, запропонованих раніше.

LightGBM алгоритмреалізовувався з метою прискорити існуючі реалізації бустингу, а саме XGBoost.

Основний час на побудову алгоритму йде на пошук кращої точки розбиття тренувальних даних у вузлі за деякою ознакою. У XGBoost реалізовано 2 механізми:

1. для знаходження кращого розбиття ознаки пресортуються і знаходиться

найкраще розбиття

2. алгоритм з урахуванням гістограми. ознаки об'єктів розбиваються на групи

LightGBM пропонує 2 техніки для багаторазового прискорення швидкості роботи алгоритму:

1. Gradient-based One-Side Sampling (GOSS). Для побудови чергового дерева використовуються не всі об'єкти. Видаляються з розгляду об'єкти з малими градієнтами
2. Exclusive Feature Bundling (EFB). Упаковка ознак. За великої кількості ознак дані з високою ймовірністю розріджені і є можливість зменшити кількість ефективних ознак

У статті про LightGBM було проведено порівняння алгоритму з XGBoost як за швидкістю роботи, так і за якістю на кількох наборах даних. Розбиралися завдання класифікації та ранжирування. Було продемонстровано від 2 - 20 кратне збільшення швидкості роботи алгоритму бустингу за тієї ж точності на відкладеній вибірці.

CatBoost ставить своїм завданням ефективну роботу з категоріальними ознаками (catboost == categorial boosting), а також пропонує нову схему для підрахунку значень у вузлах, що дозволяє зменшити перенавчання. Ще одне нововведення в тому, що при побудові чергового дерева поділ увузлі може відбуватися за сукупністю двох категоріальних ознак. Підтримка GPU. У статті проводиться порівняння з XGBoost, LightGBM, H2O реалізаціями. Показано, що CatBoost навіть з параметрами за умовчанням дозволяє досягти кращої якості (Logloss) на 8 наборах даних, що розглядаються. Показано результати порівняння часів прогнозування алгоритмів CatBoost, XGBoost, LightGBM. CatBoost виявився на порядок швидшим як в однопоточному режимі, так і в багатопотоковому. CatBoost знаходиться у вільному доступі та активно розвивається.

З огляду на те, що в нашому завданні присутні категоріальні ознаки, доцільно дослідити також і цю реалізацію.

РОЗДІЛ 3. ТЕСТУВАННЯ ДЕТЕКТУ ШКІДЛИВИХ ПРОГРАМ

3.1 Отримання даних

Для того, щоб навчити класифікатор, потрібно мати розмічені дані. Для нашого завдання потрібно отримати набір чистих та шкідливих файлів PE формату. Проблеми отримання чистих файлів немає. Ви можете використовувати файли системи Windows відразу після встановлення. Шкідливі файли можна отримати з інших джерел. Було обрано сервіс VirusTotal. Сервіс дозволяє отримати вердикти 109 антивірусів за досліджуваним файлом. Для досліджуваного файлу не за всіма антивірусами може бути отриманий вердикт. Це відбувається з кількох причин. Наприклад, файл опинився в основі зовсім недавно і ще не був оброблений усіма антивірусами. Сервіс надає можливість як завантажити файл, так і отримати раніше завантажені файли з аналізом антивірусів, використовуючи платний API. Рис.6 показано статистику щодо завантажених файлів. Крім PE формату, можна завантажувати текст в разних форматах, архіви. На сьогоднішній день у базі міститься понад мільйон файлів PE формату.

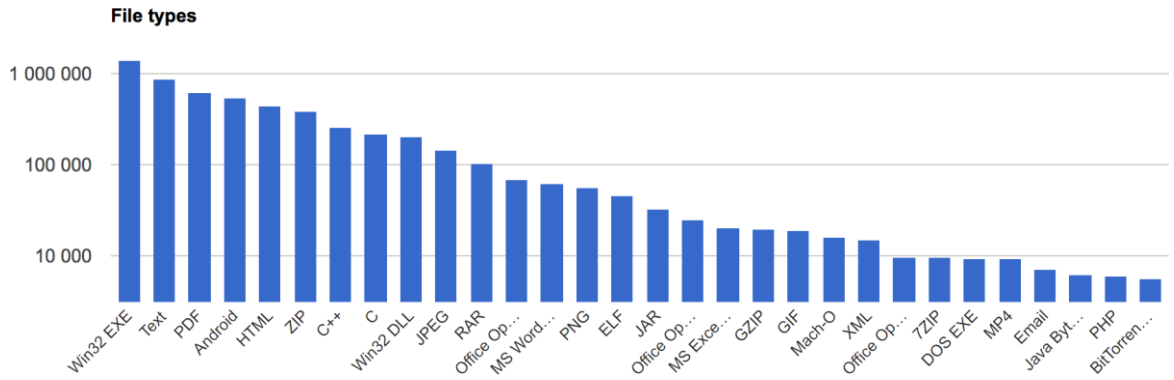


Рисунок 6 –Статистика сервісу VirusTotal по завантаженим файлам.

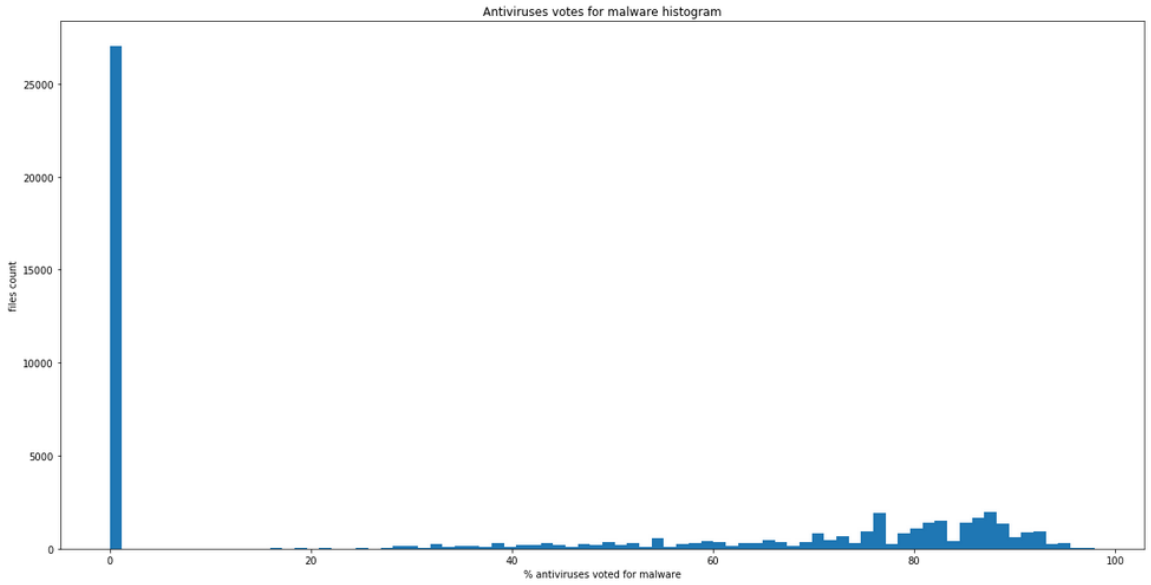
Було відзначено, що для досліджуваного файлу не завжди є вердикти щодо всім

антивірусам. Тому введемо поняття «індекс шкідливості». «Індексом шкідливостік називатимемо відношення числа антивірусів, які проголосували за шкідливість файлу, до кількості антивірусів, що проголосували за цим файлом. Таким чином, «індекс шкідливості k приймає значення від 0 до 1, де 1 означає, що всі антивіруси, аналізовані файл, вважають його вірусом.

За допомогою VirusTotal було отримано 55 432 PE-файлів (54 299 exe файлу, 1133 dll). Для кожного з них отримані вердикти щодо антивірусів. Рис.7 показано розподіл голосів антивірусів для отриманих файлів. Для кожного файлу було підраховано «індекс шкідливості».

Розподіл файлів за індексом шкідливості бачимо на гістограмі

Рисунок 7 –Розподіл файлів за голосами всіх антивірусів. По осі X відношення числа антивірусів проголосували за шкідливість файлу до всіх антивірусів, що голосували



Антивіруси влаштовані по-різному і видають не завжди узгоджені вердикти на досліджуваних файлах, що видно на Рис.7. Тобто, з одного боку, є антивіруси, які помилилися, вважаючи чисті файли вірусами (false positive), і є ті, хто помилився, вважаючи вірус чистим файлом (false negative). При ідеальних детектуючих алгоритмах антивірусів на гісто-грамі не було б «хвоста правіше 100%. Виділяється проблема - які файли вважати шкідливими під час навчання алгоритму машинного навчання.

Для найбільш точного передбачення було обрано топ-8 популярних і надійних антивірусів (Paloalto, SentinelOne, McAfee, BitDefender, Kaspersky, CrowdStrike, Avast, Symantec). Індекс шкідливості підраховується лише цих антивірусів. Відповідна гістограма Рис.8.При такому підході близько 5000 файлів, як і раніше, залишаються сумнівними (сіра зона) — близько половини з надійних антивірусів проголосували за шкідливість.

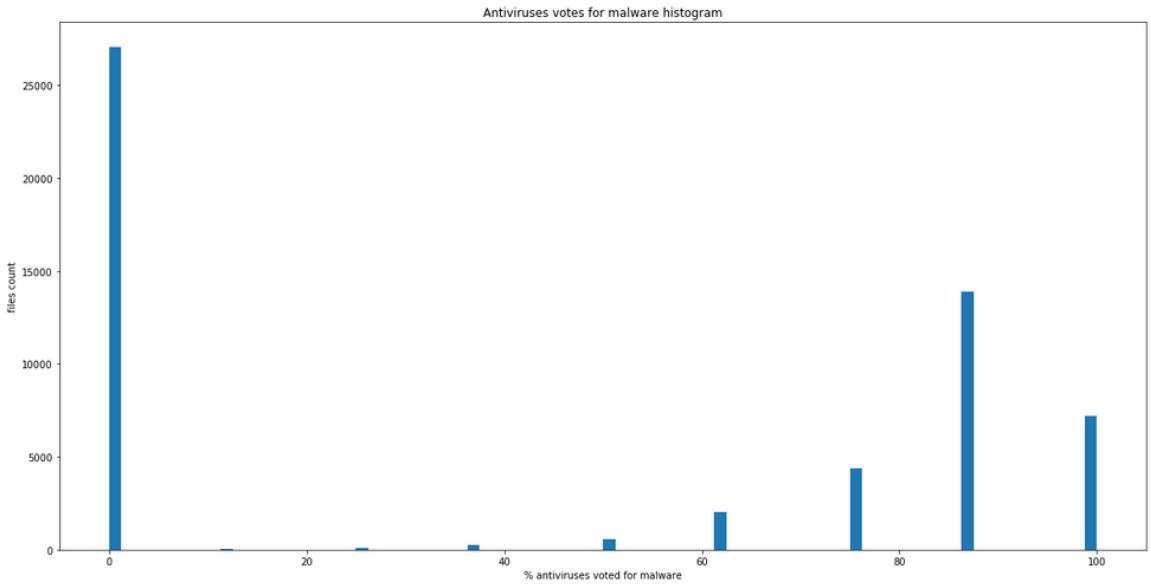


Рисунок 8 - Розподіл файлів за голосами від надійних антивірусів. По осі X відношення числа антивірусів проголосували за шкідливість файлу до всіх антивірусів, що голосували

Визначимо файл шкідливим, якщо хоча б один із топ-8 антивірусів визначив його таким. За такого підходу ми намагаємося максимально посилити детектування вірусів, зменшуючи кількість помилково негативних спрацьовувань. Зауважимо, що більшість алгоритмів класифікації підтримують можливість крім індексу передбаченого класу видавати ймовірність віднесення об'єкта до класу, тобто, ми маємо ще один інструмент впливу на передбачення вже після навчання алгоритму (вважати шкідливим файлом, якщо ймовірність більша за 30%, а не більше) 50%).

Таким чином, ми отримуємо такі дані (розмір підібраний так, щоб дані були збалансовані) у таблиці 2.

Таблиця 2 Статистика по зібраним файлам

Усі файли	55432
Чисті файли	27049
Шкідливі файли	28383

Рис.9 наведено гістограму розподілу розмірів файлів завантажених VirusTotal як для чистих, так і для шкідливих файлів. Бачимо,

що в основному файли мають розмір 500-1000 кілобайт і дуже мало файлів мають розмір близько 10 мегабайт. Також варто відзначити, що шкідливе ПЗ, як правило, має менший розмір файлів.

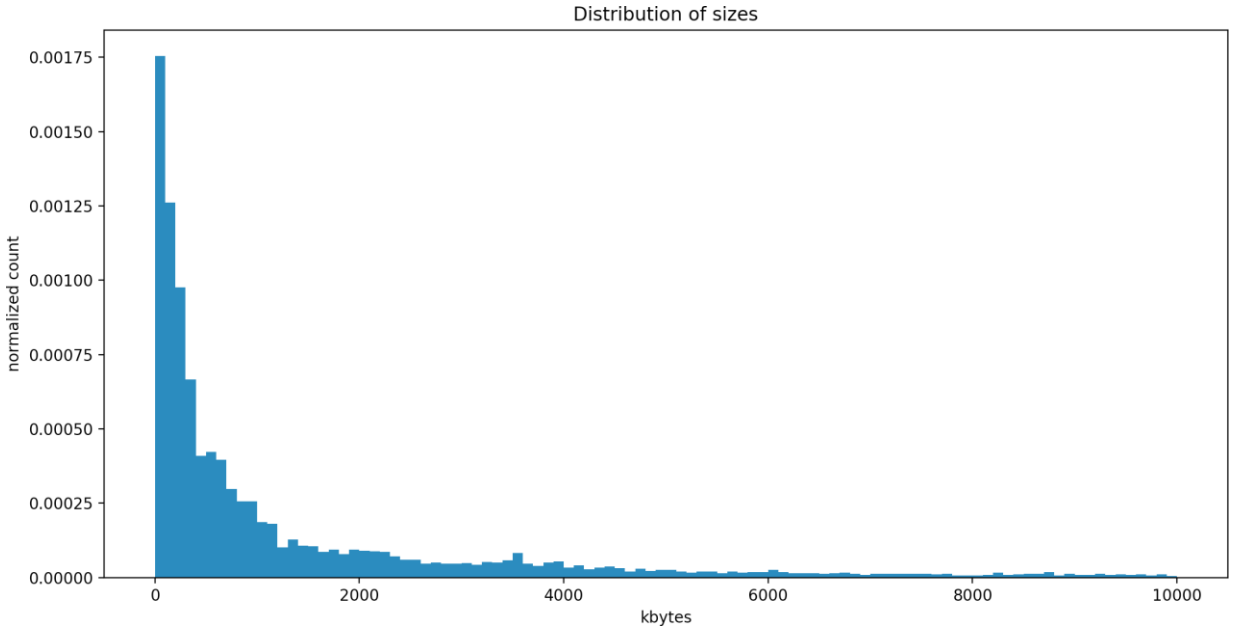


Рисунок 9 –Гістограма розподілу розмірів одержаних файлів. По осі X розмір файлу в кілобайтах

3.2 План тестування

Опишемо процес проведення експериментів. Перед проведенням експериментів розіб'ємо дані (файли) на навчальну вибірку та тестову щодо 5 до 1 збалансовано (з однаковим ставленням кількості шкідливих файлів до чистих). Підбір ознак, гіперпараметрів виконуватимемо на навчальній вибірці. Порівняння алгоритмів машинного навчання проводитимемо на результатах з тестової вибірки, що дозволить найбільш чесно провести експеримент, уникаючи перенавчання на тестових даних.

Будь-який алгоритм машинного навчання містить гіперпараметри - деякий набір параметрів, що впливає на модель, і, зрештою, якість навчання. Так, наприклад, для випадкового лісу це кількість дерев у лісі, глибина дерев, кількість ознак, що розглядаються під час розбиття вибірки у черговому вузлі дерева та інші. Для пошуку гіперпараметрів скористаємося 3-fold крос-валідацією на навчальній вибірці. Тобто, розіб'ємо

всю навчальну вибірку на 3 збалансовані частини, тричі навчимо алгоритм на 2 частинах і обчислимо значення метрик на одній, що залишилася. Середнім отримані значення. Приклад для k = 10 подано на Рис.10.Для кожного алгоритму з найкращим набором гіперпараметрів обчислюватимемо accuracy (точність) і false positive rate (частку хибно позитивних спрацьовувань) на тестовій (відкладеній) вибірці для порівняння моделей між собою. В силу того, що нам вдалося отримати збалансовані дані, accuracy є вдалою метрикою для оцінки якості алгоритмів. Інакше, варто було б вибрати інші способи оцінки, наприклад, f1 міру або площу під AUC кривою. Мотивація розрахунку false positive rate (FPR) полягає в тому, що нам важливо знати кількість чистих файлів, які помилково віднесені до шкідливих. Таких файлів має бути якнайменше. Таке помилкове визначення може призвести до псування файлів, втрати даних.

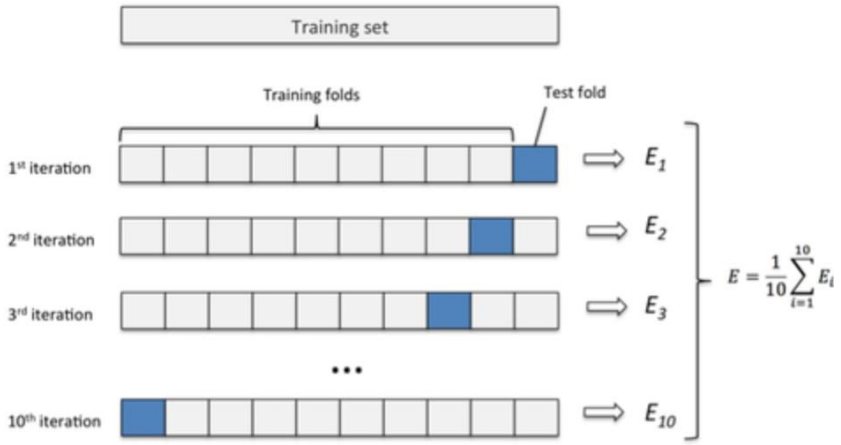


Рисунок 10 –Приклад k-fold крос-валідації при k = 10

3.3 Відбір ознак

Раніше (PE формат) ми визначили яким чином і якого типу атрибути витягуватимемо з PE файлу. Витягнемо максимально можливе число ознак із файлу, а потім відбір будемо проводити жадібним алгоритмом. Тобто, на кожному кроці будемо додавати до наявного той атрибут, який дає найбільший приріст точності (accuracy) моделі/класифікатору. Так робитимемо доти, доки якість перестане зростати. Виконуємо відбір на навчальній вибірці 3-fold крос-валідацією. Для відбору ознак як алгоритм використовуватимемо випадковий ліс.

Додатково до атрибутів обчислюватимемо ентропію секцій. Ентропія розраховується так (ентропія Шеннона)[29]:

$$H = -\sum_{i=1}^{256} pi \log pi$$

Основу логарифму беремо рівним 256 - число можливих значень байта. Таким чином, ентропія H прийматиме значення від 0 до 1.

Висока ентропія секції може сигналізувати про стиснення секції, використуванні пакувальників, шифрувальників для приховання шкідливого коду.

Зібрані атрибути з файлів:

1. чисельні значення атрибут, взяті з полів PE файлу безпосередньо
2. прапори присутності api, секцій та їх характеристики. Значення: {0, 1}
3. ентропія секцій

Після роботи жадібного алгоритму ми отримали набір із 115 ознак, на основі яких можна детектувати шкідливе ПЗ.

Повний перелік зібраних атрибут файлів представлений у Додатку 1. Збираємо прямі атрибути структури PE файлу з MS-DOS заглушки, заголовка coff, опціонального заголовка. Для фіксованої кількості dll-бібліотек та імпортованих api були виділені запропоновані в додатку назви. Вирішили виділити 5 секцій з назвами: text, data, rsrc, rdata, reloc і отримати характеристики — права секції (shared, execute, read, write) та його энтропию. Раніше зазначалося, що назви секцій для завантажувача не мають значення, проте дані, «стандартні» секції зустрічаються у більшості файлів і містять, як правило, те, на що і вказують. Такі відхилення двигун відловлюватиме. Разом із безпосередньо вилученими параметрами збиратиме загальну інформацію щодо самих структур. Наприклад, кількість імпортованих бібліотек, назв функцій, секцій, символи. Назвемо категорію таких ознак - general.

Можна виділити групи атрибутів залежно від структури їх вилучення. Окремо розглядатимемо характеристики секцій та його ентропії. Після виділення таких груп можна побудувати графік їх важливості. Важливість групи вважається як сума важливості ознак, що входять до групи.

Рис.11 представлено значимість груп ознак. За графіком видно, що ентропія секцій разом із ознаками опціонального заголовка грає вирішальну роль детекції вірусу, зараження файла. Найменш важлива інформація про DOS заголовок, coff заголовок і характеристики секцій.

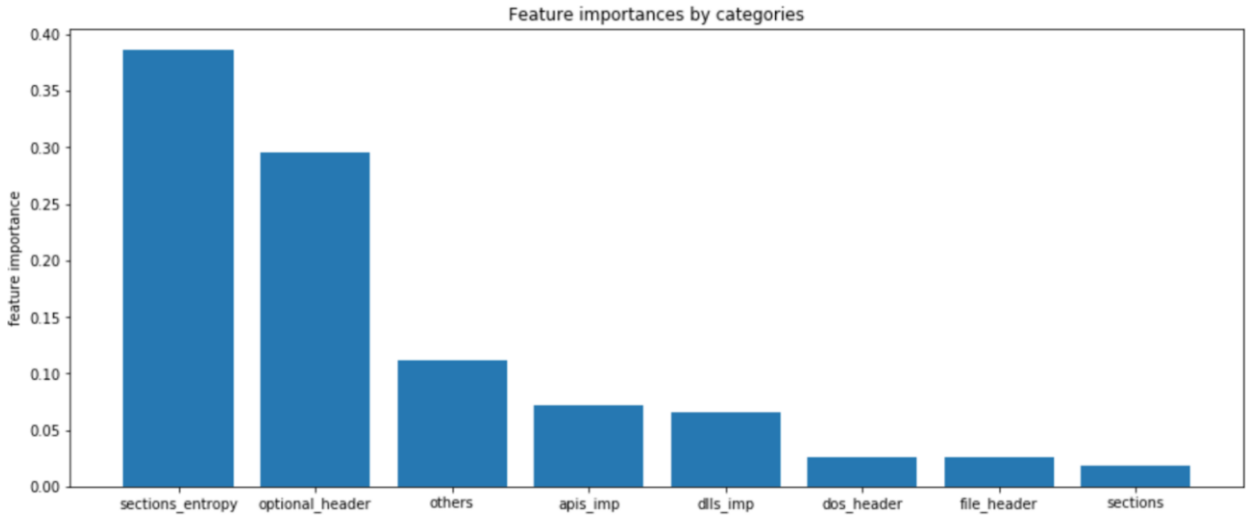


Рисунок 11 –Значимість ознак, розрахована за випадковим лісом

3.4 Порівняння алгоритмів

Після того, як знайдено визначальні ознаки та зафіксовано дані, потрібно знайти найкращу модель, найкращий алгоритм класифікації. Розглянемо 4 обрані моделі: випадковий ліс (Random Forest), XGBoost, LightGBM, CatBoost. Всі експерименти проводимо мовою Python у середовищі ipython notebook з відповідними інструментами та бібліотеками. Для кожної моделі пошук гіперпараметрів здійснюємо по 3-fold крос валідації. Після підбору гіперпараметрів навчаємо модель на всій наданій тренувальній вибірці та обчислюємо accuracy та false positive rate на тестових даних. Результати, що виходять, наведені в таблиці 3.

На додаток до зведеної таблиці якості алгоритмів, наведемо залежність якості алгоритмів від розміру навчальної вибірки - Рис.12.За кривими можна помітити (апроксимація), що збільшення даних, що навчаються, може покращити якість детектування вірусів (криві не досягли насичення).

Найкращим із розглянутих алгоритмів у застосуванні до цього завдання

Таблиця 3 Порівняння алгоритмів

Алгоритм	Accuracy %	False positive rate %
Випадковий ліс	95.9	1.9
XGBoost	96.3	1.4
LightGBM	96.5	1.4
CatBoost	96.3	1.5

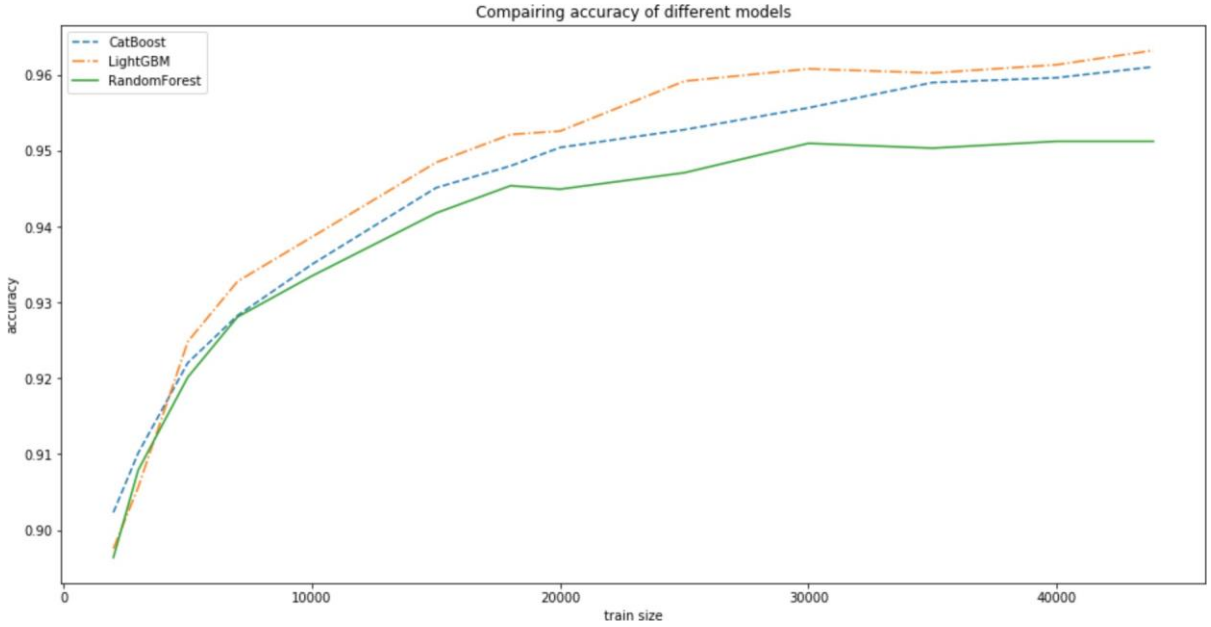


Рисунок 12 –Залежність якості алгоритму від даних

виявився градієнтний бустинг із LightGBM реалізацією. Вдалося досягти кращої, ніж необхідна точності (accuracy) і хорошої false positive rate. CatBoost навіть з підбором гіперпараметрів не виявився найкращим.

Взявши за основу алгоритм градієнтного бустингу з LightGBM реалізацією був реалізований двигун на C + + інтегрований у продукт. Крім якості роботи алгоритму, інша важлива характеристика - швидкість його роботи. Була виміряна швидкість роботи двигуна для перевірки задоволення вимог у реальних умовах. Результати представлені в таблиці 4. Швидкість обчислювалася на 50 тисячах файлів, розміщених на HDD диску. При запуску на SSD диску швидкість роботи класифікатора не змінилася, що логічно, а PE-парсер почав працювати в середньому за 5 мс на одному файлі.

Рис.13 представлено розподіл часу обробки двигуном файлів. Бачимо, що в основному файл обробляється менш ніж за 20 мс. Також проаналізуємо частку файлів, що обробляються за потрібний час. Зокрема, частку файлів, оброблюваних протягом менше 30 мс, якби обмеження тимчасово було жорстким і застосовувалося до кожного файлу. У цьому випадку буде оброблено 90% всіх файлів, а 10% - проігноровано. Варто розуміти, що в реальних умовах обмеження вводяться не на один файл, а на кілька, виходячи із середнього часу, а також з того, що більшість вірусів, інфікованих файлів мають малий розмір файлу і, відповідно, малий час роботи алгоритму. Середній час обробки файлу – 21 мс, що є хорошим показником та задовольняє вимогу.

Рис.14 представлена залежність швидкості роботи двигуна від розміру файлу. Бачимо, що залежність часу обробки файлу з його розміру лінійна. Виймаючи атрибути з PE файлу, ми повинні пройтись по таблиці імпорту, по секціях для отримання ентропій. Логічно припускати, що чим більший розмір файлу, тим більше стають секції та таблиці імпорту, що містять назви dll і api. Маючи розмір досліджуваних файлів, можна оцінити час їх обробки. Так, на 100 GB даних потрібно близько 10 хвилин.

Таблиця 4 Середній час роботи складових частин двигуна

	Час роботи, ms
PE-парсер	16.0
Класифікатор	0.5
Загальний час	16.5

Результати оптимістичні і показують застосовність двигуна на практиці. Також цікаво подивитися за такою характеристикою як false negative rate. Частка вірусів, які двигун не зміг виявити. Раніше вважалося, що файл є вірусом, якщо за це проголосував хоча бодин із надійних антивірусів. Тепер побудуємо розподіл індексу шкідливості від усіх антивірусів для false negative файлів. Гістограма наведена

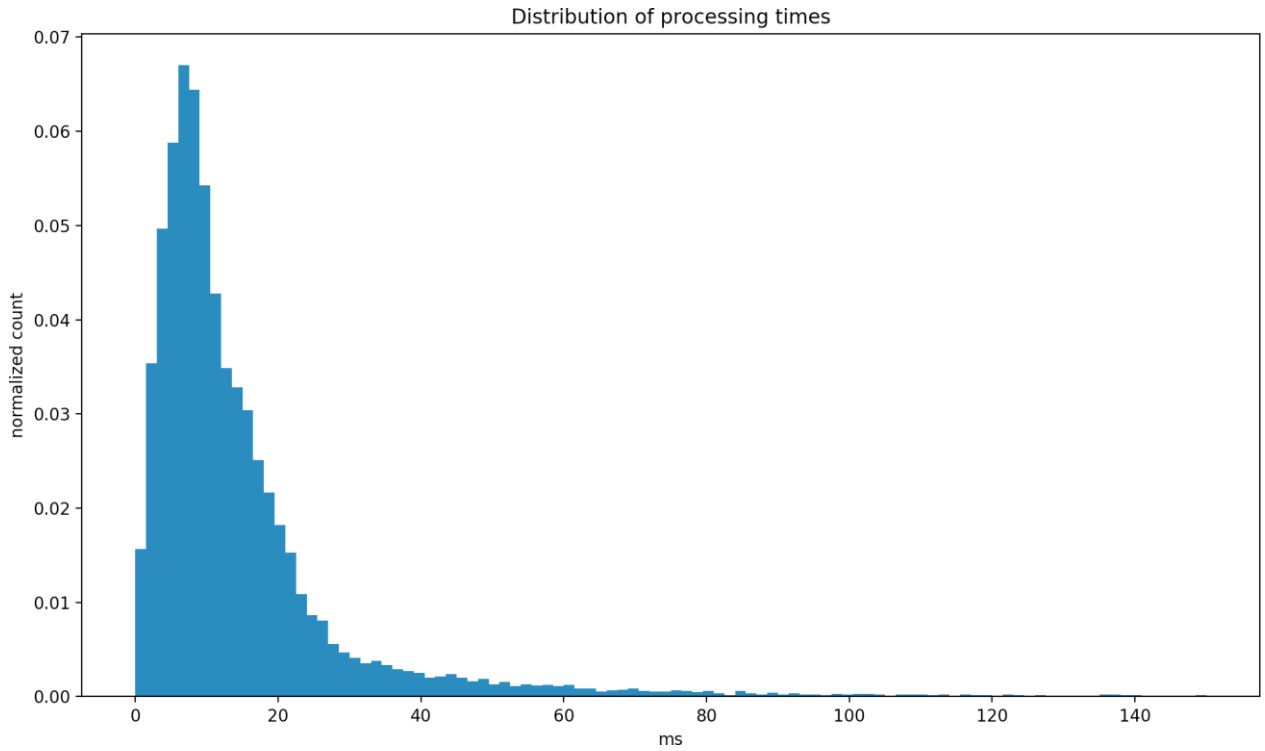


Рисунок 13 –Розподіл часу роботи движка для файлів різного розміру

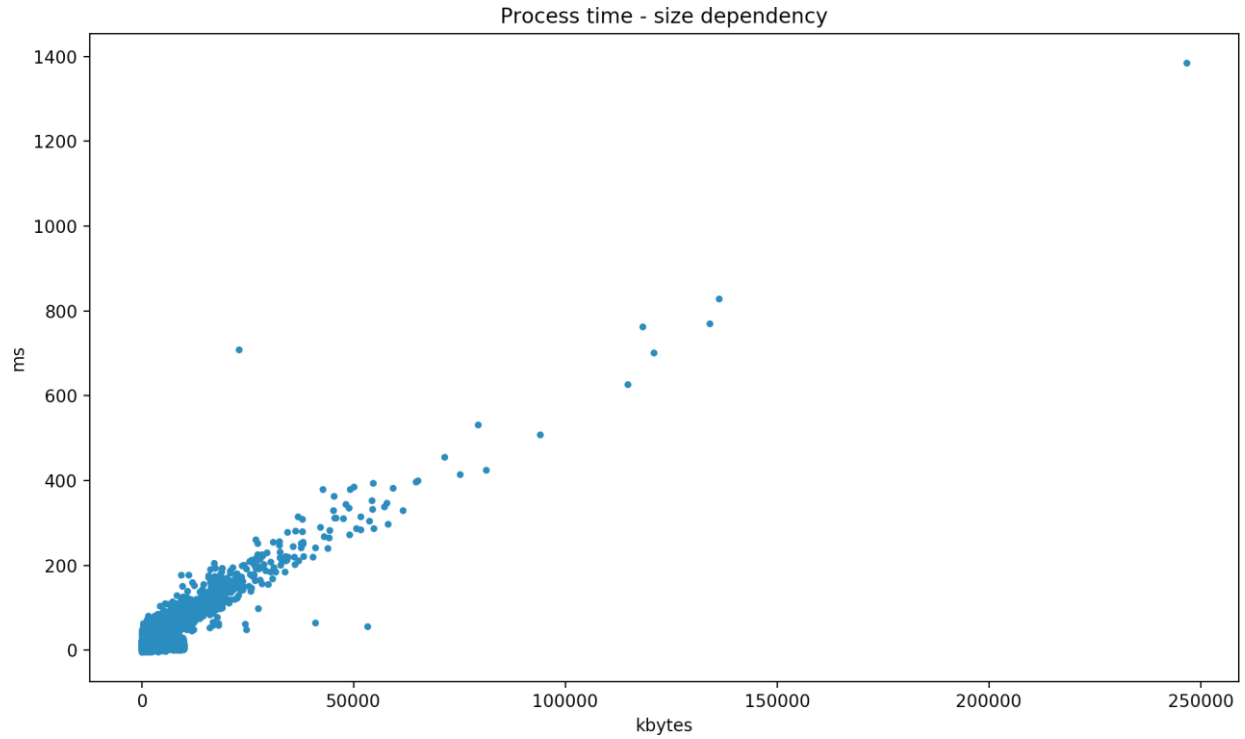


Рисунок 14 –Залежність швидкості роботи движка від розміру файлу

Рис.15. Можна порівняти цей розподіл із представленим раніше для всіх файлів (Рис.7). Рис.15 розподіл більш рівномірний. попри

на загальне зміщення вправо (центр мас приблизно в 0.6), багато антивірусів виявилися нездатними правильно визначити ці випадки. Для покращення якості алгоритму можна сконцентруватися на таких сірих файлах та визначити їх особливості.

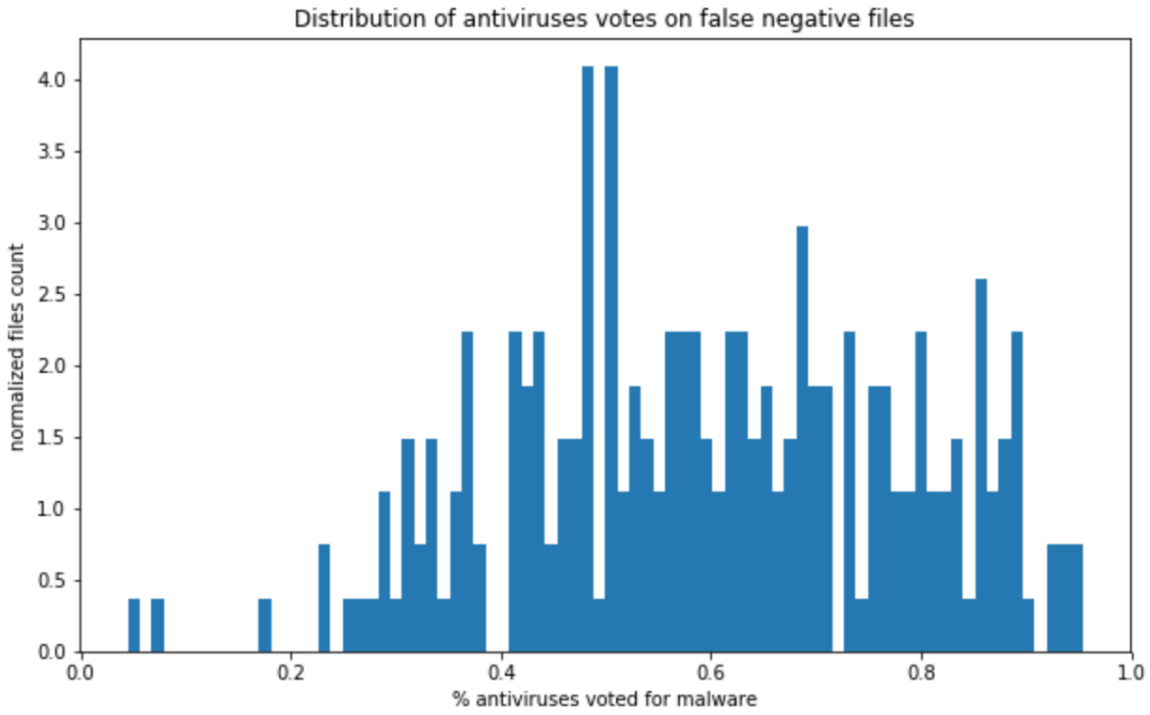


Рисунок 15 –Розподіл голосів антивірусів по false negative файлам

ВИСНОВКИ

У ході роботи було отримано низку результатів:

1. Виконано огляд застосовуваних методів статичного аналізу визначення шкідливих програм. Область є активно розвивається в даний час, відбувається перехід від евристик до застосування методів машинногонавчання. Виділено основні засади побудови статичного аналізу. Існує велика кількість робіт зі створення евристик для детекції шкідливого коду. У багатьох роботах не висвітлюється практичне застосування алгоритмів машинного навчання

2. Отримано дані для аналізу (понад 50000 файлів), побудови двигуна статичного аналізу. Наведено метод розбиття файлів на чисті та шкідливі.

3. Проведено відбір найбільш значущих ознак. З повним списком одержаних ознак можна ознайомитись у Додатку 1

4. Зроблено порівняння сучасних алгоритмів машинного навчання в рамках даного завдання в умовах обмежених ресурсів та часу. Вибрано найкращий алгоритм з метрики точності (accuracy), що задовольняє висунутим вимогам якості

5. Реалізовано двигун детектування шкідливого коду в рамках програмного продукту для кінцевих користувачів, що задовольняє вимоги швидкості роботи, що висуваються.

Цінним практичним результатом роботи стало створення ефективного за швидкістю та якістю двигуна машинного навчання.

Проектування та створення двигуна виконувалося на основі результатів проведених досліджень з відбору найбільш значущих ознак, вибору відповідного алгоритму. Вимоги до швидкості роботи моделі накладали обмеження на вибір алгоритму, кількість ознак, що витягуються. У ході роботи було отримано уявлення про значущість окремих ознак та груп ознак для детектування шкідливого ПЗ.

Отриману експертизу можна використовувати для побудови складніших систем детектування вірусів. Наведемо тут лише невеликий перелік можливих подальших кроків щодо розвитку системи статичного аналізу захисту даних користувача:

1. Перехід до моделі клієнт-сервер. Модель призначена для перенесення складної роботи з даними на сервер. Користувач надсилає на сервер набір ознак для розрахунку загрози файлу. Це дозволяє зняти обмеження на вибір алгоритму

2. Додавання можливості розшифрування файлів. Шкідливий код може бути зашифрований. Можливість отримати вихідний код призведе до збільшення детекції моделлю вірусів

3. Застосування глибинних методів машинного навчання. У разі клієнт-серверної архітектури стають доступні складніші методи аналізу файлів. Деякі дослідження вже були виконані у цьому напрямку. Дослідження можливості застосування згорткових, рекурентних мереж для детектування шкідливого ПЗ

Список літератури

1 Symantec intelligent report. - 2015. - URL: https://www.dimt.it/wp-content/uploads/2014/10/www.symantec.com_content_en_us_enterprise_other_resources_b-intelligence_report_05-2014.en-us.pdf

2 *Jinrong Bai, Junfeng Wang, Guozhong Zou*. A Malware Detection Scheme Based on Mining Format Information // The Scientific World Journal. - 2014. - Vol. 2014 року.https://www.hindawi.com/journals/tswj/2014/260905/

3 Eureka: A Framework for Enabling Static Malware Analysis / Sharif M. [et al.] - 2008. - Vol. 5283. - Pp. 481-500.https://www.covert.io/research-papers/security/Eureka%20-%20A%20framework%20for%20enabling%20static%20malware%20analysis.pdf

4 *Hahn K.*Robust Static Analysis of Portable Executable Malware // HTWK Leipzig. - 2014.https://papers.put.as/papers/malware/2014/masterthesiskatja.pdf

5 Novel Feature Extraction, Selection and Fusion for Effective Malware Family Classification / Ahmadi M. [et al.] // In Proceedings of the 6 ACM Conference on Data and Application Security and Privacy / ACM. - 2016. - Pp. 183-194.https://arxiv.org/pdf/1511.04317.pdf

6 An intelligent PE-malware detection system based on association asociation mining / Ye Y. [et al.] // Journal in Computer Virology. - 2008. - Vol. 4, no. 4. - Pp. 323-334. https://www.researchgate.net/publication/238420750_An_intelligent_PE-malware_detection_system_based_on_association_mining

7 PE-Miner: Mining Structural Information to Detect Malicious Executables in Realtime / Shafiq M. Z. [et al.] // Recent Advances in Intrusion Detection, Lecture Notes in Computer Science — 2009 — Vol. 5758 — Pp. 121-141.https://www.researchgate.net/publication/221427459_PE-Miner_Mining_Structural_Information_to_Detect_Malicious_Executables_in_Realtime

8 *Tabish SM, Shafiq MZ, Farooq M.*Malware detection використовуючи статистичну analysis of byte-level file content // In Proceedings of ACM SIGKDD Workshop на CyberSecurity and Intelligence Informatics / ACM. - 2009. - Pp. 23-

31.https://web.cs.ucdavis.edu/~zubair/files/kdd09-momina.pdf

9 Griffin K. [et al.] // In Proceedings, Recent Advances in Intrusion Detection. - 2009. - Pp. 101-120.

https://www.researchgate.net/publication/242504646_Recent_Advances_in_Intrusion_Detection_7th_International_Symposium_RAID_2004_Sophia_Antipolis_France_September_15-17_2004_Proceedings

10 Large-scale malware indexing using function- call graphs // In Proceedings of the 16th ACM Conference on Computer and Communications Security / ACM. — 2009. — Pp. 611-620.

https://www.researchgate.net/publication/221609147_Large-scale_malware_indexing_using_function-call_graphs

11 Malware detection based on mining api calls / Sami A. [et al.] // In Proceedings of the 2010 ACM Symposium on Applied Computing / ACM. - 2010. - Pp. 1020–1025.https://www.researchgate.net/publication/221000581_Malware_detection_based_on_mining_API_calls

12 Manjunath. Malware images: Visualization and automatic classification / Nataraj L. [et al.] // У процесі 8-го Міжнародного симпозісу на Visualization for Cyber Security / ACM. - 2011. - Pp. 41-47.https://www.researchgate.net/publication/228811247_Malware_Images_Visualization_and_Automatic_Classification

13 A static, packer-agnostic filter to detect similar malware samples / Jacob G. [et al.] // In Proceedings of the 9th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. - 2013. - Pp. 102-122 .https://link.springer.com/book/10.1007/978-3-642-37300-8

14 Opcode sequences as representation of executables for data-mining-based unknown malware detection / Santos I. [et al.] // Information Sciences. — 2013. — Vol. 231 — Pp. 64-82.
https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.473.9043&rep=rep1&type=pdf

15 Expert Systems with Applications. - 2014. - Vol. 41, no. 13. - Pp. 5843-

5857.https://www.elsevier.com/journals/expert-systems-with-applications/0957-4174?generatepdf=true

16 DLLMiner: structural mining for malware detection / Narouei M. [et al.] // Security and communication networks. — 2015. — Vol. 8, no.18. — Pp. 3311-3322. https://www.researchgate.net/publication/275385542_DLLMiner_Structural_Mining_for_Malware_Detection

17 *Chen T., C. Guestrin C.* XGBoost: A Scalable Tree Boosting System // Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. - 2016. - Pp. 785-794. https://typeset.io/pdf/xgboost-a-scalable-tree-boosting-system-48ocu6x4c7.pdf

18 *Breiman L.*Bagging predictors // Machine Learning. - 1996. - Vol. 24, no. 2 - Pp. 123-140. http://www.machine-learning.martinsewell.com/ensembles/bagging/Breiman1996.pdf

19 A survey on Heuristic Malware Detection Techniques / Bazrafshan Z. [et al.] //5th Conference on Information and Knowledge Technology. - 2013. - Pp. 113-120.https://ieeexplore.ieee.org/abstract/document/6620049

20 *Ucci D., Aniello L., Baldoni R.* Survey on the Usage of Machine Learning Techniques for Malware Analysis - 2018.https://arxiv.org/pdf/1710.08189v2.pdf

21 Microsoft Корпорація. PE Формат specification. - URL: https://www.researchgate.net/publication/275385542_DLLMiner_Structural_Mining_for_Malware_Detection

23 Portable Executable Wiki. -URL: https://en.wikipedia.org/wiki/Portable_Executable

24 Microsoft Corporation. What is a DLL? - 2007 - URL: https://learn.microsoft.com/en-US/troubleshoot/windows-client/deployment/dynamic-link-library

25 *Воронцов К.* Курс лекцій. - URL: http://www.machinelearning.ru/wiki/images/archive/9/97/20201031123349%21Voron-ML-Logic-slides.pdf

26 *Воронцов До.* Курс лекцій. презентації. - URL:http://www.machinelearning.ru/wiki/images/0/0d/Voron-ML-Compositions.pdf

27 LightGBM: A Highly Efficient Gradient Boosting Decision Tree. - 2017. https://www.microsoft.com/en-us/research/wp-content/uploads/2017/11/lightgbm.pdf

28 *Dorogush AV, Ershov V., Gulin A.*CatBoost: gradient boosting з категорічними features support. - 2017. https://arxiv.org/pdf/1810.11363.pdf

29 *Shannon C. E.* A Mathematical Theory of Communication // Bell System Technical Journal. — 1948. — Vol. 27. — Pp. 379-423. https://people.math.harvard.edu/~ctm/home/text/others/shannon/entropy/entropy.pdf

Додаток 1 (список ознак)

1. MS-DOS Stub/e_crlc
2. MS-DOS Stub/e_lfanew
3. Coff Header/SizeOfOptionalHeader
4. Coff Header/PointerToSymbolTable
5. Header/PointerToSymbolTable
6. Coff Header/NumberOfSections
7. Optional Header/SizeOfStackReserve
8. Optional Header/DllCharacteristics
9. Optional Header/BaseOfCode
10. Optional Header/SizeOfUninitializedData
11. Optional Header/SizeOfInitializedData
12. Coff Header/Characteristics
13. Optional Header/SizeOfHeapCommit
14. Optional Header/SizeOfHeapCommit
15. Optional Header/SizeOfHeapReserve
16. Optional Header/FileAlignment
17. Optional Header/SizeOfHeaders
18. Optional Header/SizeOfHeaders
19. Optional Header/SizeOfImage
20. Optional Header/SizeOfStackCommit
21. DLL Referred/Imm32.dll

22. DLL Referred/Uxtheme.dll
23. DLL Referred/Setupapi.dll
24. DLL Referred/Shlwapi.dll
25. DLL Referred/Gdiplus.dll
26. DLL Referred/Mscoree.dll
27. DLL Referred/Gdi32.dll
28. DLL Referred/Oleacc.dll
29. DLL Referred/Version.dll
30. DLL Referred/Userenv.dll
31. DLL Referred/Msvcrt.dll
32. DLL Referred/Shell32.dll
33. DLL Referred/Wintrust.dll
34. DLL Referred/Rpcrt4.dll
35. DLL Referred/Wininet.dll
36. DLL Referred/Iphlpapi.dll
37. DLL Referred/Winspool.drv
38. DLL Referred/Winmm.dll
39. DLL Referred/Comdlg32.dll
40. DLL Referred/Advapi32.dll
41. DLL Referred/Ole32.dll
42. DLL Referred/Wtsapi32.dll
43. DLL Referred/Ntdll.dll

44. DLL Referred/Mpr.dll
45. DLL Referred/Netapi32.dll
46. DLL Referred/User32.dll
47. DLL Referred/Oleaut32.dll
48. DLL Referred/Urlmon.dll
49. DLL Referred/Comctl32.dll
50. DLL Referred/Crypt32.dll
51. DLL Referred/Ws2_32.dll
52. DLL Referred/Psapi.dll
53. DLL Referred/Msimg32.dll
54. API Referred/Createfilew
55. API Referred/Createfilea
56. API Referred/Getmodulehandlew
57. API Referred/Getmodulehandlea
58. API Referred/Getmodulehandleexw
59. API Referred/Heapsetinformation
60. API Referred/Loadlibrarya
61. API Referred/Writeprocessmemory
62. API Referred/Reopenfile
63. API Referred/Movefilea
64. API Referred/Movefileexw
65. API Referred/Createprocessuserw

66. API Referred/Setunhandledexceptionfilter
67. API Referred/Getenv
68. API Referred/Createprocessuser
69. API Referred/Copyfilew
70. API Referred/Duplicatehandle
71. API Referred/Copyfilea
72. APIReferred/Virtualprotect
73. API Referred/Virtualprotectex
74. API Referred/Rtllookupfunctionentry
75. API Referred/Createdirectorya
76. API Referred/Createdirectoryw
77. API Referred/Movefilew
78. API Referred/Movefileexa
79. API Referred/Virtualalloc
80. API Referred/Openprocess
81. API Referred/Ntsetinformationfile
82. API Referred/Setfileinformationbyhandle
83. API Referred/Openprocesstoken
84. API Referred/Rtlvirtualunwind
85. API Referred/Openfile
86. API Referred/Ntcreatefile
87. API Referred/Loadicona

88. API Referred/Rtlcapturecontext
89. API Referred/Copyfileexw
90. API Referred/Findfirstfileexw
91. API Referred/Replacefilew
92. API Referred/Writefileex
93. API Referred/Ntsuspendprocess
94. API Referred/Ntsuspendthread
95. API Referred/Findfirstfilew
96. API Referred/Findfirstfileexa
97. API Referred/Copyfileexa
98. API Referred/Replacefilea
99. API Referred/Ntopenfile
100. API Referred/Writefile
101. API Referred/Deletefilew
102. API Referred/Suspendthread
103. API Referred/Deletefilea
104. API Referred/Findfirstfilea
105. Section/data : характеристики * 4, ентропія
106. Section/text : характеристики * 4, ентропія
107. Section/rsrc: характеристики * 4, ентропія
108. Section/rdata : характеристики * 4, ентропія

109. Section/reloc: характеристики * 4, ентропія

110. General/NumberOfReferredDLLs

111. General/NumberOfReferredAPIs

112. General/NumberOfReferredAPIOrdinals

113. General/NumberOfSections

114. General/NumberOfExportTableSymbols

115. General/NumberOfRelocSectionItems