Санкт-Петербургский государственный университет
Факультет прикладной математики и процессов управления

РИТТЕР Герман Андреевич

Выпускная квалификационная работа

Обнаружение программ-шифровальщиков на основе статического и динамического анализа с использованием методов машинного обучения

Уровень образования: бакалавриат

Направление 02.03.02 «Фундаментальная информатика и информационные технологии»

Основная образовательная программа СВ.5003.2018 «Программирование и информационные технологии»

Научный руководитель:
доцент кафедры технологий программирования,
к.т.н. Блеканов Иван Станиславович

Рецензент:
доцент кафедры компьютерных технологий и систем,
к.т.н. Погожев Сергей Владимирович

Санкт-Петербург
2022 г.

Оглавление

Введение

Актуальность

Последние несколько лет все большее внимание злоумышленников привлекает относительно новый класс вредоносного программного обеспечения – вымогатели-шифровальщики. Будучи запущенной на компьютере жертвы, программа шифрует файлы пользователя, а за восстановление доступа к ним вымогатели требуют выкуп.

Все чаще целью атак злоумышленников становятся крупные организации. Сами вымогатели при этом стали использовать тактику двойного выкупа, требуя деньги как за расшифровку файлов, так и за удаление украденных злоумышленниками данных с целью не допустить публикацию конфиденциальных сведений.

К решению проблемы подключились многие компании, работающие в сфере обеспечения информационной безопасности. Во многих современных антивирусных решениях можно обнаружить защиту от вымогательского ПО. Однако не стоят на месте и злоумышленники, использующие различные техники для маскировки вредоносной активности. Таким образом, актуальной стала задача обнаружения данного класса вредоносного ПО.

Цель работы

Разработать алгоритм распознавания программ-шифровальщиков. При этом могут использоваться как методы статического, так и методы динамического анализа.

Задачи работы

1. Определение набора признаков для обучения модели.
2. Создание обучающей и тестовой выборок модели на основе уже известных образцов программ-шифровальщиков.
3. Реализация и обучение модели, её тестирование.

Практическая значимость

Разработанный метод может использоваться как в личных, так и в корпоративных системах для их защиты от атак программ-вымогателей.

Обзор существующих решений

Методы на основе динамического анализа

Большинство современных решений для распознавания программ-вымогателей основаны на динамическом анализе. Предложенный в 2018 году метод [3] использовал в качестве входных данных историю API-вызовов, которая затем преобразовывалась в многомерный вектор. За счет детектирования на основе метода опорных векторов была получена точность распознавания 97,48%.

В описанном в [9] исследовании в качестве исходных данных так же использовалась история API-вызовов. Для детектирования вымогателей использовался многослойный перцептрон, для обучения использовалось 7 семейств вымогателей. Методом была получена точность 98%.

Опубликованный в [10] метод основывается на поиске последовательных шаблонов и использовании четырех различных алгоритмов классификации. В ходе эксперимента была получена точность распознавания 99%.

Так как в ходе динамического анализа происходит непосредственный запуск потенциально вредоносного образца, методы на его основе имеют высокую точность распознавания. Однако существенным недостатком подхода является относительно большое время, необходимое для анализа. Более того, многие современные программы-вымогатели (как и многое другое вредоносное ПО) способны обнаруживать виртуальные окружения и не выполнять в них злонамеренных действий.

Методы на основе статического анализа

Опубликованная в 2019 году статья [4] предлагала решение, целиком основанное на статическом анализе. Идея заключалась в преобразовании ассемблерных инструкций исполняемого файла в предложения. В ходе эксперимента были использованы пять различных классификаторов.

Наибольшая точность (91,43%) была получена при использовании классификатора на основе случайного леса.

На свойствах ассемблерных инструкций исполняемого файла была основана и описанная в [11] техника. Для классификации использовался метод опорных векторов. Исследователями была получена точность 96,5% в классификации пяти семейств вымогателей.

Стоит отметить, что предложенные методы не подходили для обнаружения ранее неизвестных семейств шифровальщиков, а лишь для классификации уже известных семейств. О том, почему статический анализ малоэффективен против новых семейств вредоносного ПО, более подробно написано в Главе 1 настоящей работы.

Методы на основе гибридного анализа

Некоторыми исследователями предлагаются методы, использующие преимущества как статического, так и динамического анализа.

В опубликованном в 2018 году исследовании [5] описывается метод детектирования программ-вымогателей, использующий гибридный подход при анализе приложений. Исследователями в ходе эксперимента была получена точность распознавания 98,25%. Для обучения использовалось 74 образца двенадцати различных семейств.

Гибридный подход применялся и для анализа приложений для операционной системы Android [12]. Учитывались такие признаки приложения, как использование приложением памяти и ЦПУ, сетевая активность и статистика системных вызовов.

Собственное решение для детектирования программ-вымогателей, получившее название RansomFlare, в 2017 году предложили специалисты компании F-Secure [6]. Отличительной особенностью решения стало большое число использованных для обучения семейств вымогателей – использовались образцы трехсот различных семейств. Полный список извлекаемых из образцов признаков, к сожалению, не раскрывается.

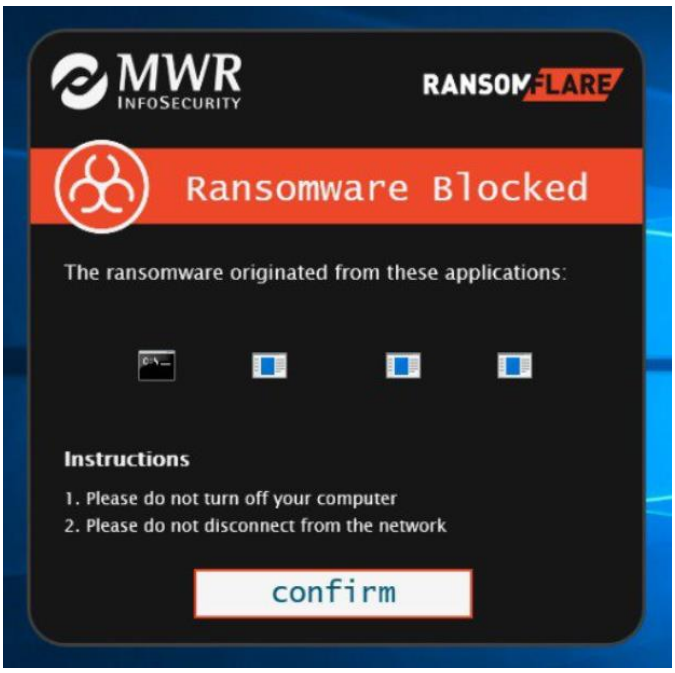


Рисунок 1. Окно программы RansomFlare

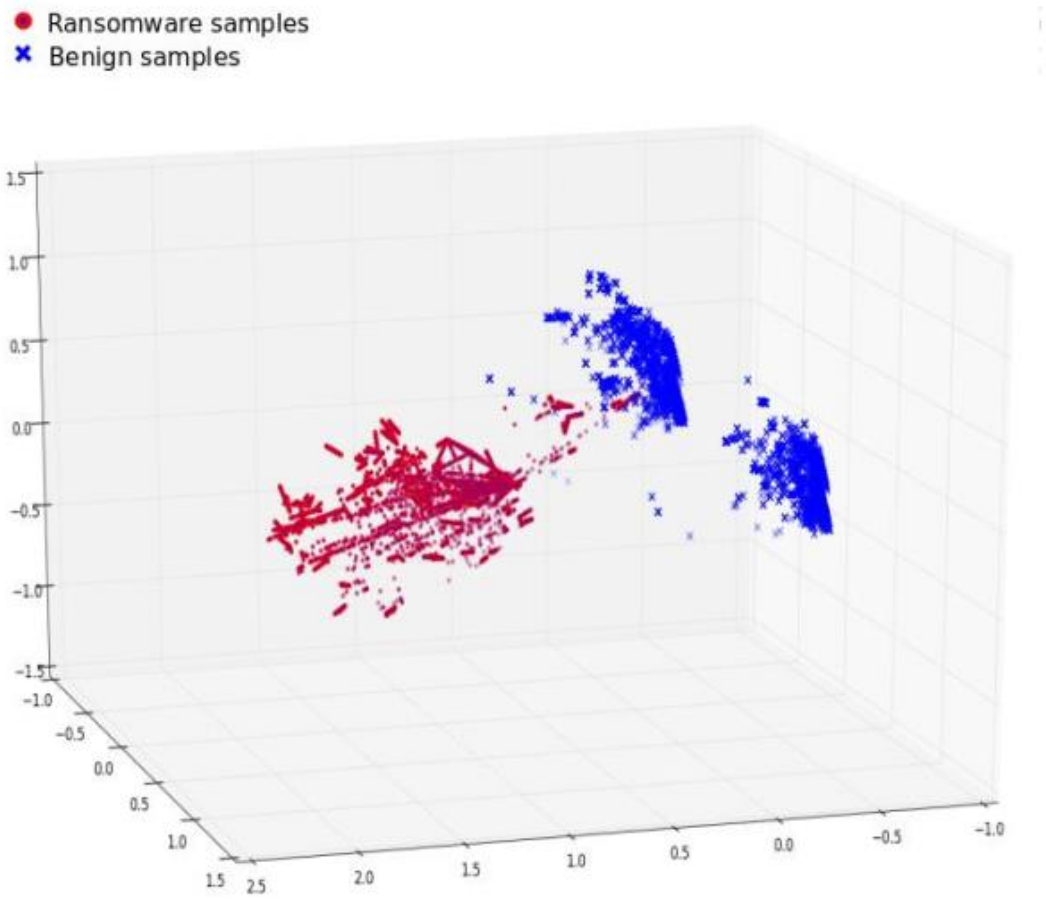


Рисунок 2. Визуализация датасета RansomFlare с помощью метода главных компонент

Глава 1. Составление списка критериев

1.1. Ограничения статического анализа

Главным недостатком использования лишь статического анализа для обнаружения вредоносных программ является его слабая приспособленность к обнаружению новых образцов и семейств вредоносного ПО. На это есть несколько причин:

1. **Слабая эффективность при использовании обфускации кода.** Для обхода сигнатурных правил разработчики вредоносного ПО часто используют различные техники обфускации, позволяющие каждый раз генерировать уникальные исполняемые файлы. Данные техники не влияют на общее поведение вредоносной программы, однако позволяют обойти системы защиты, полагающиеся лишь на статический анализ.
2. **Сигнатурный анализ неэффективен против часто обновляемого вредоносного ПО.** Новые образцы зачастую появляются быстрее, чем базы данных антивирусных вендоров пополняются сигнатурами старых образцов.
3. **Слабая эффективность при таргетированных атаках.** Вместо распространения все новых образцов вредоносного ПО, злоумышленники могут использовать новый образец при атаках на отдельные, специально выбранные для атак организации.

Таким образом, использование лишь статического анализа недостаточно для создания системы с высоким показателем распознавания шифровальщиков. Поэтому для распознавания будет использован как статический, так и динамический анализ.

1.2. Аномальное поведение программы

Шифровальщики имеют много общих с другими видами вредоносного ПО поведенческих шаблонов. Данные шаблоны могут быть разбиты на 6 групп [6]:

1. **Закрепление в системе.** Техники этой группы используются для возможности управления зараженным компьютером в том числе после перезагрузки операционной системы.
2. **Защита от восстановления.** Одной из распространенных тактик, применяемых программами-вымогателями, является уничтожение теневых копий данных для предотвращения восстановления файлов из их незашифрованной версии.
3. **Сокрытие вредоносного поведения.** Злоумышленниками используются различные техники для избегания обнаружения как пользователем, так и антивирусными сканерами. Одной из распространенных техник является инъекция кода в легитимный процесс.
4. **Изучение системного окружения.** Перед установкой вредоносная программа может провести различные проверки системного окружения. Обычно такие проверки осуществляются для того, чтобы убедиться, что программа запускается на компьютере реального пользователя, а не в песочнице или отладчиком с целью изучения аналитиком.
5. **Сетевая активность.** Многие образцы шифровальщиков требуют для работы интернет-соединение. Требуется оно для обмена использованными при шифровании файлов ключами.
6. **Повышение привилегий.** Выполнение определенных действий в системе требует повышенных привилегий. Примером может послужить программа-вымогатель Petya, использовавшая администраторские привилегии для перезаписи и шифрования MBR, необходимой для загрузки операционной системы.

1.3. Основные критерии

На основе приведенных выше тактик и техник был составлен список критериев для оценки вредоносности программы:

1. Частота изменения программой файлов на системе.
2. Наличие в сетевых адресах, к которым подключался исполняемый файл, адресов, использованных в предыдущих атаках шифровальщиков и другого вредоносного ПО. В качестве признака использовалась репутация адреса на сервисе AbuseIPDB.
3. Сетевая активность в процессе выполнения, то есть такие признаки, как:
    a. Объем отданных по сети данных
    b. Объем полученных по сети данных
    c. Количество сетевых адресов, к которым подключался исполняемый файл
4. Максимальная энтропия секций исполняемого файла. Высокая энтропия говорит об использовании сжатия и/или шифрования при генерации секции, что часто используется для сокрытия вредоносного кода.
5. Наличие в исполняемом файле фактов использования шифрования:
    a. Магические константы современных алгоритмов шифрования
    b. Сигнатуры популярных криптографических библиотек
    c. Наличие функций Windows CryptoAPI в таблице импортов исполняемого файла

    Для обнаружения данных сигнатур использовались сигнатурные правила Yara. Признаком выступало число сработавших правил.
6. Наличие в исполняемом файле свидетельств использования техник обхода детектирования антивирусом – техник обнаружения факта отладки и инъекции кода. Был составлен список подозрительных функций Windows API на основе

описанных в реестре MITRE ATT&CK [1] техник и документации Microsoft [2]:

a. Debugger Evasion (T1622) и ее подтехники
b. Process Injection (T1055) и ее подтехники

В качестве признака использовалось количество подозрительных функций, импортируемых исполняемым файлом.

7. Число запущенных исполняемым файлом процессов.

Ниже приведен пример данных признаков, извлеченных из образцов семейства шифровальщиков Ryuk.

	bad_imports	connections_num	crypto_usage	modification_rate	ip_reputation	max_entropy	bytes_download	bytes_upload	started_processes
0	4	9	0	5.541806	0	7.765381	0	0	20
1	3	0	1	5.515050	0	7.017339	0	0	15
2	3	0	1	10.571906	0	6.499513	0	0	37
3	3	0	1	68.725753	0	6.694101	0	0	2
4	3	10	1	107.870293	17	6.542796	93696	8419	12
5	4	0	0	0.186441	0	7.267143	0	0	20
6	4	0	0	0.220339	0	7.267143	0	0	22
7	4	0	1	0.067797	0	7.737801	0	0	4
8	3	3	1	133.420168	0	7.640575	0	306	22
9	4	0	2	0.694915	0	6.673520	0	0	20
10	4	0	0	7.070588	0	7.267143	0	0	23
11	3	0	1	10.237288	0	7.640575	0	0	22
12	4	41	0	20.156627	0	7.566996	0	6042	25

Рисунок 3. Набор признаков, извлеченных из образцов Ryuk

Глава 2. Сбор данных для обучения

2.1. Использованные инструменты

1. Источником образцов вредоносного ПО служила онлайн-песочница ANY.RUN. Возможности песочницы включают в том числе поиск вредоносных программ определенных видов и семейств.
2. Для извлечения признаков были использованы Python-библиотека pefile для парсинга PE-файлов и сигнатурный движок Yara.
3. Были также использованы правила для движка Yara для поиска криптографических примитивов (см. приложение).

2.2. Использованные семейства вымогателей

На основе определенных критериев были собраны как образцы вымогателей-шифровальщиков, так и безопасное программное обеспечение. Всего было собрано 200 безвредных программ и такое же количество образцов вымогателей десяти различных семейств:

1. Cerber (19 образцов)
2. Conti (15 образцов)
3. GandCrab (21 образец)
4. LockBit (25 образцов)
5. Maze (22 образца)
6. NetWalker (25 образцов)
7. Ragnar Locker (16 образцов)
8. Ryuk (15 образцов)
9. REvil Sodinokibi (20 образцов)
10. TeslaCrypt (22 образца)

Данные делились на обучающую и тестовую выборки одинакового размера. При этом для обучения и для тестирования использовались разные семейства вымогателей для того, чтобы избежать обучения модели на распознавание конкретных семейств вымогателей.

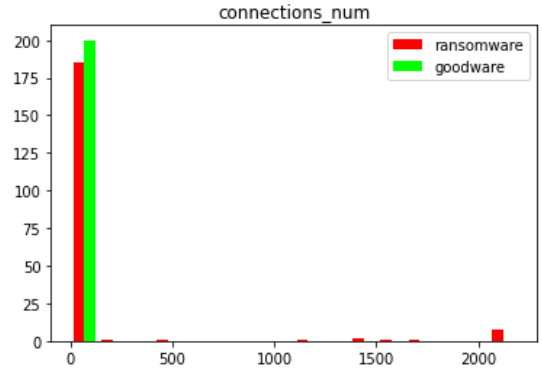
Итак, в обучающую выборку вошли образцы семейств: Cerber, GandCrab, Maze, Ragnar Locker, TeslaCrypt.

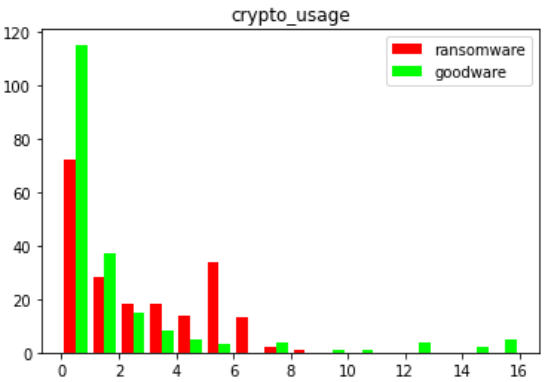
В тестовую выборку вошли образцы семейств: Conti, LockBit, NetWalker, Ryuk, REvil Sodinokibi.

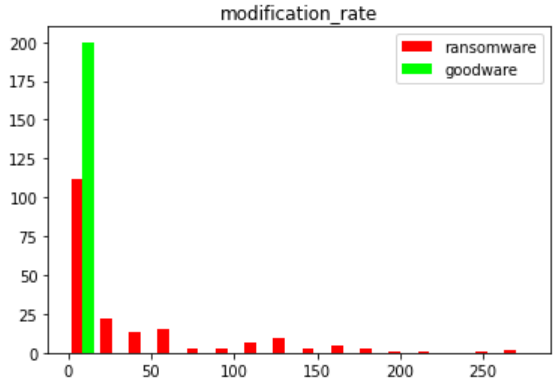
Глава 3. Обучение модели

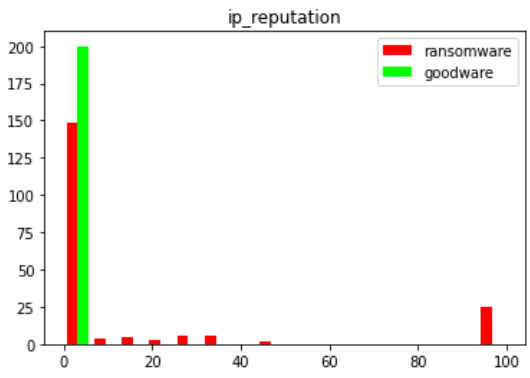
3.1. Визуализация данных

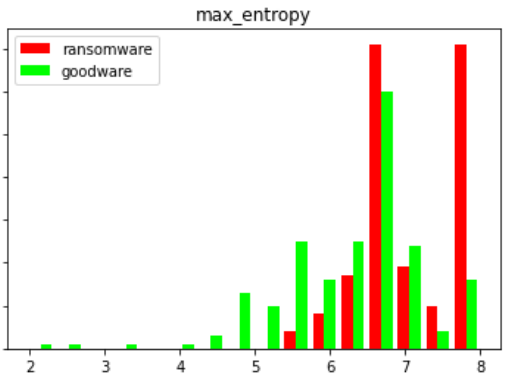
На рисунках ниже представлена гистограмма признаков для программ-вымогателей (красный цвет) и безопасных программ (зеленый цвет).











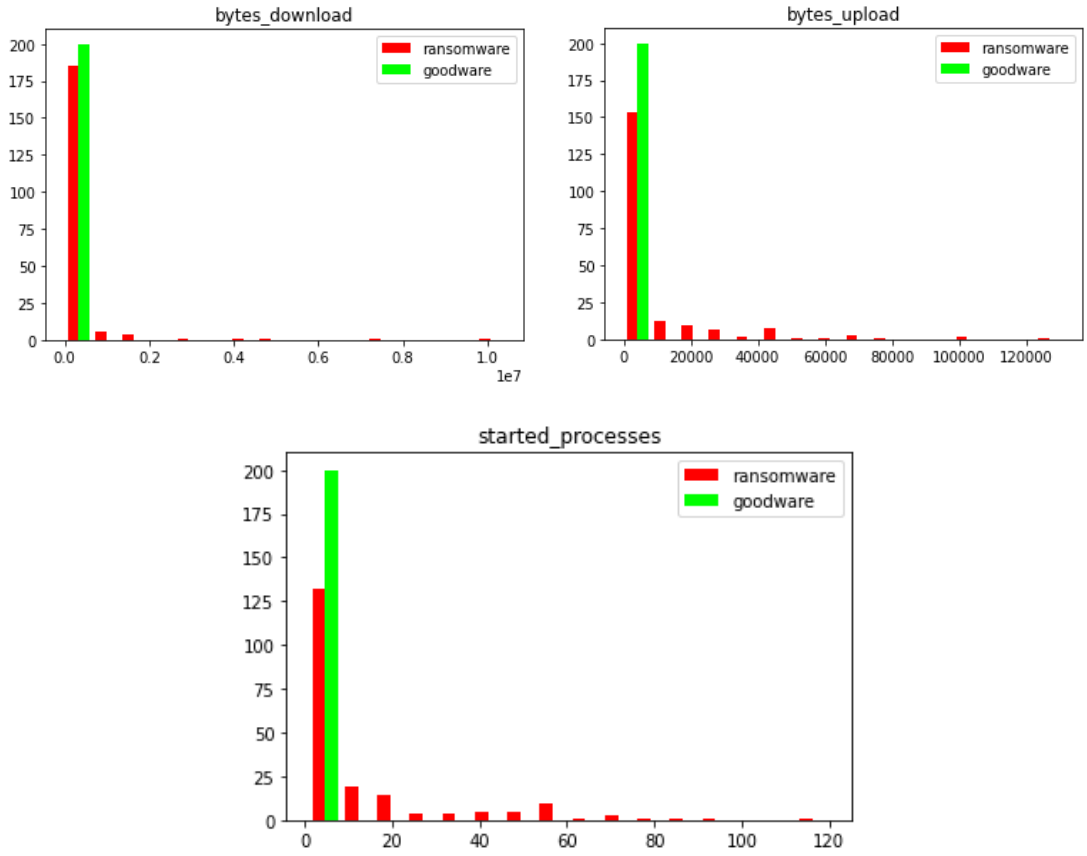


Рисунок 4. Гистограмма распределения признаков

Визуализируем данные с помощью алгоритма снижения размерности и визуализации многомерных данных t-SNE. Красными точками обозначены образцы-шифровальщики, зелеными – безопасные программы:

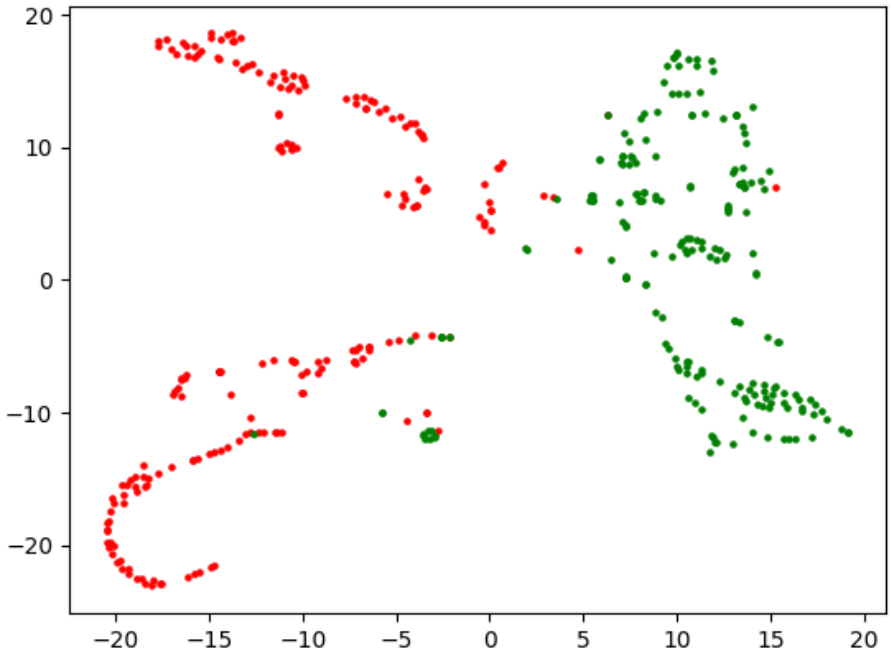


Рисунок 5. Визуализация данных с помощью t-SNE

За исключением единичных случаев, выборка на основе выделенных критериев оказалась линейно разделимой после снижения размерности.

3.2. Использованные метрики

Для оценки качества работы моделей были использованы стандартные метрики качества:

1. Recall = $\frac{TP}{TP+FN}$
2. Precision = $\frac{TP}{TP+FP}$
3. Accuracy = $\frac{TP+TN}{TP+FP+TN+FN}$

Где:

1. TP (True Positive) – число программ-вымогателей, классифицированных верно
2. TN (True Negative) – число безопасных программ, классифицированных верно
3. FP (False Positive) – число безопасных программ, ошибочно классифицированных как вредоносные
4. FN (False Negative) – число программ-вымогателей, ошибочно классифицированных как безопасные

3.3. Обучение модели на основе случайного леса

В качестве модели для обучения был использован классификатор на основе случайного леса, использовалась реализация библиотеки sklearn. Точность распознавания измерялась при различных значениях параметра n_estimators (число деревьев в случайном лесу).

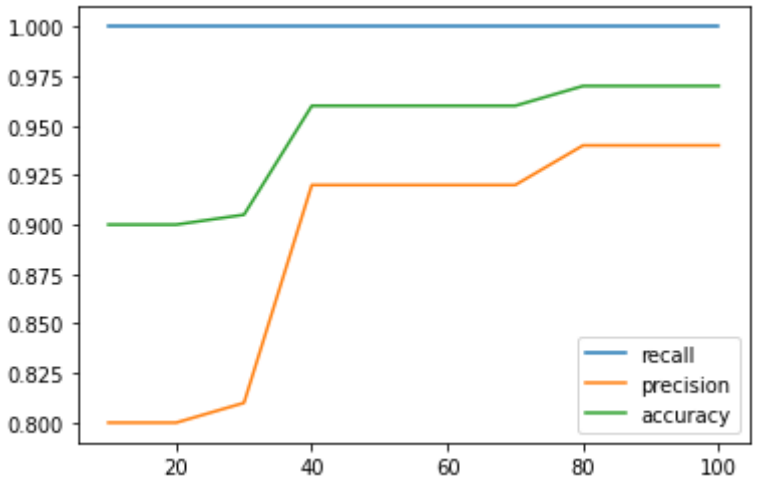


Рисунок 6. Зависимость различных метрик качества от числа деревьев в лесу

Метрика recall оказалась равной 1.0 при всех значениях числа деревьев, то есть классификатором были обнаружены все программы-вымогатели. Метрики accuracy и precision достигли оптимального значения при числе деревьев 80 в лесу.

3.4. Обучение модели на основе метода опорных векторов

Для реализации классификатора использовалась библиотека sklearn для языка Python. Значение параметра регуляризации – 1,0. Так как выборка линейно разделима, то использовалось линейное ядро.

Были получены следующие результаты обучения:

Таблица 1. Результаты обучения SVM

Метрика	Accuracy	Recall	Precision
Значение метрики	0,97	1,00	0,94

3.5. Результаты

Были получены следующие результаты обучения моделей на основе случайного леса и на основе метода опорных векторов:

Таблица 2. Результаты обучения Random Forest и SVM

	Accuracy	Recall	Precision
Random Forest	0,97	1,00	0,94
SVM	0,97	1,00	0,94

Заключение

В ходе работы были реализованы две модели для распознавания программ-шифровальщиков. Полученная точность классификации обеих моделей составляет 97%. При этом полнота моделей (recall) оказалась равной 100%, то есть системой верно распознаются все образцы программ-вымогателей тестовой выборки.

Отличием данной работы от многих других является использование малого числа признаков, необходимых для классификации, в сочетании с высокой точностью распознавания.

Дальнейшим направлением для развития результатов работы может являться интеграция модели с какими-либо песочницами (например, с песочницей Cuckoo Sandbox). Таким образом, результаты работы могут быть использованы для защиты как личных компьютеров, так и корпоративных систем.

Список литературы

1. MITRE ATT&CK – knowledge base of adversary tactics and techniques [Электронный ресурс]. – URL: https://attack.mitre.org/
2. Техническая документация Microsoft [Электронный ресурс]. – URL: https://docs.microsoft.com/
3. Takeuchi Y., Sakai K., Fukumoto S. Detecting ransomware using support vector machines //Proceedings of the 47th International Conference on Parallel Processing Companion. – 2018. – С. 1-6.
4. Zhang H. et al. Classification of ransomware families with machine learning based on N-gram of opcodes //Future Generation Computer Systems. – 2019. – Т. 90. – С. 211-221.
5. Shaukat S. K., Ribeiro V. J. RansomWall: A layered defense system against cryptographic ransomware attacks using machine learning //2018 10th International Conference on Communication Systems & Networks (COMSNETS). – IEEE, 2018. – С. 356-363.
6. Nieuwenhuizen D. A behavioural-based approach to ransomware detection //Whitepaper. MWR Labs Whitepaper. – 2017.
7. Khammas B. M. Ransomware detection using random forest technique //ICT Express. – 2020. – Т. 6. – №. 4. – С. 325-331.
8. Masum M. et al. Ransomware classification and detection with machine learning algorithms //2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC). – IEEE, 2022. – С. 0316-0322.
9. Vinayakumar R. et al. Evaluating shallow and deep networks for ransomware detection and classification //2017 international conference on advances in computing, communications and informatics (ICACCI). – IEEE, 2017. – С. 259-265.

10. Homayoun S. et al. Know abnormal, find evil: frequent pattern mining for ransomware threat hunting and intelligence //IEEE transactions on emerging topics in computing. – 2017. – Т. 8. – №. 2. – С. 341-351.
11. Baldwin J., Dehghantanha A. Leveraging support vector machine for opcode density based detection of crypto-ransomware //Cyber threat intelligence. – Springer, Cham, 2018. – С. 107-136.
12. Ferrante A. et al. Extinguishing ransomware-a hybrid approach to android ransomware detection //International Symposium on Foundations and Practice of Security. – Springer, Cham, 2017. – С. 242-258.

Приложение

1. Репозиторий с результатами исследования: https://github.com/exp101t/ransomware-detect.
2. Набор правил Yara для обнаружения криптографических примитивов в программах: https://github.com/Yara-Rules/rules/blob/master/crypto/crypto_signatures.yar.