МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
Кафедра технологий программирования

АНАЛИЗ ВРЕДНОСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ С ИСПОЛЬЗОВАНИЕМ DENSE PIXEL DISPLAY

Курсовая работа

Сечко Марины Юрьевны
студентки 3 курса,
специальность «компьютерная безопасность»

Научный руководитель:
кандидат физико-математических наук
В.В. Мушко

Минск, 2018

Аннотация

Сечко М.Ю. Анализ вредоносного программного обеспечения с использованием dense pixel display: Курсовая работа / Минск: БГУ, 2018.

В работе рассматриваются метод визуализации больших данных при помощи пиксельных графиков и его применимость к анализу вредоносного программного обеспечения. Реализовано построение пиксельных графиков по входным данным, интерактивное приложение для визуализации с использованием языка программирования Python.

Анатацыя

Сячко М.Ю. Аналіз шкоднаснага праграмнага забеспячэння з выкарыстаннем dense pixel display: Курсавая работа/ Мінск: БДУ, 2018.

У працы разглядаюцца тэхніка візуалізацыі вялікага набору дадзеных пры дапамозе піксельнага метаду і яе дастасавальнасць да аналізу шкоднаснага праграмнага забеспячэння. Рэалізавана пабудова піксельных графікаў па уваходным дадзеных, інтэрактыўная праграма для візуалізацыі з выкарыстаннем мовы праграмавання Python.

Annotation

Sechko M.Y. Malware analysis using dense pixel display: Course work / Minsk: BSU, 2018.

The work deals with the technique of visualizing a large data set using a pixel method and its applicability to the analysis of malicious software. Drawing pixel plots on the input data and interactive visualization application have been developed using the Python programming language.

РЕФЕРАТ

Курсовая работа, 39 стр., 32 рис., 4 источника.

Анализ вредоносного программного обеспечения с использование dense pixel display

Ключевые слова: визуализация данных, анализ вредоносного ПО, пиксельные методы визуализации, dense pixel display.

Объект исследования – вредоносное программное обеспечение.

Цели работы – разработка приложения, основанного на методе пиксельных графиков, с поддержкой импорта данных и организацией экспорта в форматы png, jpg, для анализа вредоносного программного обеспечения.

Результат работы – приложение для анализа вредоносного программного обеспечения, основанного на методе пиксельных графиков, с поддержкой импорта данных и организацией экспорта в форматы png, jpg.

Область применения – анализ данных о вредоносном программном обеспечении, анализ больших наборов данных.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Визуализация данных – один из методов, используемых для передачи данных или информации путем их представления в виде визуальных объектов (например, точек или линий), содержащихся в графике.

Первичной целью визуализации данных является четкая и эффективная передача информации с помощью статистической графики, графиков и информационной графики (инфографики).

Визуализация данных – это и искусство, и наука. Как эстетичность и красота, так и функциональность должны идти рука об руку, обеспечивая понимание довольно разреженного и сложного набора данных, передавая его ключевые аспекты более интуитивно понятным способом.

Отличным примером важности визуализации данных является так называемый квартет Энскомба — четыре набора числовых данных, у которых простые статистические свойства (среднее значение, дисперсия, корреляция) идентичны, но их графики существенно отличаются.

Уже из этого можно сделать вывод, что при анализе данных визуализация данных не только желательна, а даже необходима.

Область применения визуализации достаточно велика – она используется в научных и статистических исследованиях (в частности, в прогнозировании, интеллектуальном анализе данных, бизнес-анализе), в педагогическом дизайне для обучения и тестирования, в новостных сводках и аналитических обзорах. Сфера анализа вредоносного программного обеспечения не стала исключением в потребности визуализации.

1. АНАЛИЗ ВРЕДОНОСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Из-за растущей угрозы со стороны вредоносного программного обеспечения исследование уязвимых систем становится все более важным. Необходимость регистрации и анализа активности охватывает сети, отдельные компьютеры, а также мобильные устройства. Хотя существуют различные автоматические подходы и методы для обнаружения и идентификации вредоносных программ, фактический анализ постоянно растущего числа подозрительных образцов является трудоемким процессом для аналитиков вредоносного ПО. Использование визуализации может помочь поддержать этот процесс анализа в отношении исследования, сравнения и обобщения образцов вредоносных программ.

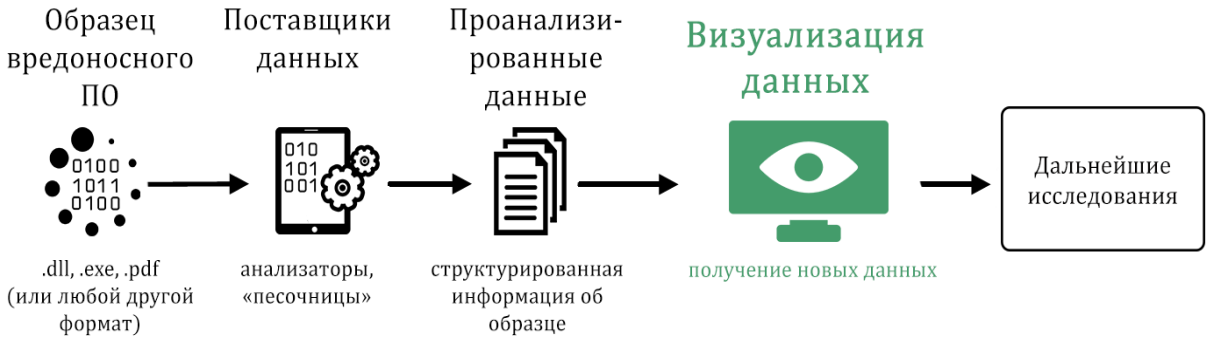

Рисунок 1.1. Схема анализа вредоносного программного обеспечения

В общем случае, схему анализа любой вредоносной программы можно представить в виде, представленном на рисунке 1.1. В качестве входных данных может выступать сам исполняемый файл или какая-либо информация о нём – отчёт о его функциях, обращениях к библиотекам, время выполнения и так далее. Далее этот файл проходит через один из так называемых «поставщиков данных» – любой инструмент, который статически или динамически анализирует входной файл и возвращает собранную информацию для дальнейшей обработки. После этого наступает этап визуализации полученных данных с целью выявлений новых интересных свойств или закономерностей. Полученные результаты могут использоваться для дальнейших исследований и/или получения выводов о входном файле.

1.1 Типы анализа вредоносного ПО

Существует два основных подхода к анализу вредоносного ПО: статический и динамический.

- *Статический анализ* описывает методы, которые не требуют фактического выполнения проверяемого образца. В зависимости от глубины анализа файл может быть проверен на его основные свойства (например, тип файла, контрольная сумма), легко извлекаемая информация (например, строки, информация об импортах DLL библиотек) или полностью разобран.
- *Динамический анализ* выполняет файл в главной системе, при этом различные инструменты контролируют выполнение проверяемого образца и записывают соответствующую информацию в лог выполнения. В зависимости от глубины анализа, это могут быть как простые операции файловой системы, так и целые инструкции, захваченные через отладчик.

Заметим, что несмотря на то, что динамический анализ может показать больше при базисном (имеющем небольшую глубину) анализе, при более продвинутом анализе вредоносного ПО используют именно статический анализ – при помощи дизассемблеров, методов обратной разработки и так далее. Причиной этому является зависимость динамического анализа от среды анализа: данные, полученные при этом типе анализа, могут являться неполными в силу текущих свойств соответствующей системы. Однако статический анализ проводится медленнее, так как аналитику приходится вручную разбираться в структуре исполняемого файла.

Выделим некоторые свойства, которые могут быть интересны для анализа вредоносного программного обеспечения.

1. Хэш-код (hash code) файла. Двоичный код образца проходит через процесс хеширования и полученный хэш сопоставляется с другими, хранящимися в базе поставщиков данных, чтобы определить, является ли весь файл (контрольная сумма) или части кода злонамеренными. Некоторые инструменты позволяют по полученному хэшу сразу вычислять и семейство вредоносной программы.
2. Информация о заголовках в файле описывает фактический тип файла (несмотря на возможное изменение типа файла) и секции его кода.
3. Импорты библиотек и функций могут указывать на функциональность самого файла. Как правило, библиотеки содержат набор функций, относящихся к определенной области работы. Таким образом, можно будет предположить, какую цель преследует исполняемый файл.
4. Инструкции ЦПУ (центрального процессорного устройства) несут в себе информацию об основной функциональности программы. Они могут быть получены при помощи процесса обратной разработки и

дизассемблирования кода либо при динамическом отслеживании значений регистров.

5. Информация об API-вызовах и системных вызовах может помочь распознать, к примеру, создание новых файлов или модификации в регистре.

6. Операции с файловой системой, регистрами, операции с сетью и так далее.

1.2 Анализ кибер-атак

Вредоносные программы не представляли бы никакой угрозы, если бы всегда оставались только на компьютерных системах, где они были разработаны. Однако существуют методы для внедрения их на другие системы.

Кибератака – это любой тип наступательного маневра, который совершается отдельными лицами, группами или организациями и который атакует компьютерные информационные системы, компьютерные сети или персональные компьютеры с целью внедрения вредоносного ПО.

Выделяют следующие виды атак в интернете:

- Вандализм и пропаганда — порча интернет-страниц, замена содержания другими элементами; рассылка обращений пропагандистского характера или вставка пропаганды в содержание других интернет-страниц.
- Сбор информации — взлом частных страниц или серверов для сбора информации и/или её замены на фальшивую.
- Отказ сервиса — атаки с разных компьютеров для нарушения функционирования сайтов или компьютерных систем.
- Вмешательства в работу оборудования — атаки на компьютеры, которые занимаются контролем над работой какого-либо оборудования, что приводит к его отключению или поломке.
- Атаки на пункты инфраструктуры — атаки на компьютеры, обеспечивающие жизнедеятельность городов, их инфраструктуры, таких как телефонные системы, водоснабжения, электроэнергии, пожарной охраны, транспорта и так далее.

Выделим «полезные» свойства кибер-атак, благодаря которым можно их анализировать:

1. IP-адреса могут дать информацию, откуда или куда была направлена атака.

2. Время атаки или атак может указывать на часовой пояс, в котором находится злоумышленник.

3. Протоколы, которые использовались при передаче данных.

Обеспечивать безопасность систем с каждым днём становится всё сложнее из-за появления новых типов вредоносных программ. Понимание моделей атаки дает более глубокое понимание уязвимости сети, которое, в свою очередь, может использоваться для защиты сети от будущих атак.

1.3 Источники данных о вредоносном ПО

Определим термин «источники данных» как инструменты или программы, которые статически или динамически анализируют вредоносного ПО и возвращают собранную информацию для дальнейшей обработки или анализа.

Инструменты визуализации используют эти данные в качестве первичного ввода, что делает качество предоставляемой информации важным фактором для сохранения выразительности полученной визуализации. Каждый поставщик данных работает в своей среде, извлекает данные на определенном уровне анализа и возвращает их в определенном виде.

Среда анализа является основой фактической реализации соответствующей системы анализа вредоносных программ.

В зависимости от возможностей и требований источника данных эти среды могут быть физическими машинами, виртуальными машинами эмуляторами.

- *Физические машины* - это компьютеры, которые выполняют образец вредоносного ПО непосредственно в предустановленной операционной системе. Потенциально вредоносный образец может напрямую обращаться к оборудованию, на котором он работает (обычно через слой абстракции, предоставляемый операционной системой). Также важно иметь в виду, что переустановка или перезагрузка физической машины занимает больше времени, чем перезагрузка эмулируемой среды. Источники данных должны запускаться непосредственно на операционной системе.

- Под *виртуальной машиной* понимается как программная или аппаратная система, эмулирующая аппаратное обеспечение некоторой платформы. Для классических виртуальных машин монитор виртуальной машины управляет доступом к аппаратным средствам и представляет эту виртуальную копию исполняемому программному обеспечению. Такой подход предотвращает прямое взаимодействие программы с реальным

оборудованием, но может усложнить анализ вредоносного ПО, использующего методы уклонения для предотвращения выполнения на виртуальных машинах. Как и физические машины, виртуальные машины ограничены той же архитектурой, что и хост-машина; выбор операционной системы, однако, не ограничивается хост-машиной. В список примеров виртуальных машин входят продукты VMwar, проект Xen и Oracle VM VirtualBox.

- *Эмуляторы* - комплекс программных, аппаратных средств или их сочетание, предназначенное для копирования (или эмулирования) функций одной вычислительной системы на другой, отличной от первой, вычислительной системе таким образом, чтобы эмулированное поведение как можно ближе соответствовало поведению оригинальной системы. Поскольку необходимо также эмулировать саму операционную систему, необходимо воссоздать всю функциональность - библиотеки, службы и т.д., необходимые для корректного запуска образца. Вредоносные программы могут использовать методы уклонения от анализа в эмулируемой среде. Кроме того, эмуляция намного более ресурсоемкая, чем виртуализация, и значительно медленнее, чем физическая машина. Однако, поскольку эмуляция обеспечивает полный доступ к системе извне, все данные о выборке могут быть собраны непосредственно из эмулятора.

1.4 Цели анализа данных

Выделяют три цели, для которых используется анализ вредоносных элементов.

- *Индивидуальный анализ* используется для анализа отдельного образца вредоносного ПО, выявления целей его выполнения и пути их достижения.
- *Классификация* вредоносного ПО. Здесь анализ используется для определения семейства вредоносных программ, к которому относится заданный образец.
- *Анализ выборки* объектов. В этом случае анализ используется для выявления схожести в поведении большого количества образцов вредоносного ПО с целью получения какой-либо статистики.

1.5 Выводы

Первая глава содержит краткое изложений сведений, касающихся существующих типов и целей анализа вредоносного ПО, а также программных комплексов для получения данных о вредоносных программах, сравнительный анализ этих комплексов, позволивший выявить их достоинства и недостатки; определена роль визуализации в анализе данных.

2. ПИКСЕЛЬНЫЕ МЕТОДЫ ВИЗУАЛИЗАЦИИ

2.1 Общие сведения

Методы визуализации приобретают все большее значение при изучении и анализе больших объемов многомерной информации. Одной из важных техник визуализации, которая особенно интересна для визуализации больших наборов многомерных данных, является класс пиксельных методов – так называемые плотностные пиксельные графики (dense pixel display). Основная идея пиксельных методов визуализации состоит в том, чтобы одновременно представлять как можно больше объектов данных на экране путем сопоставления каждого значения данных с пикселем экрана, учитывая его цвет и расположение.

Первые очевидные плюсы такого метода визуализации заключаются в следующем:

- Отсутствие пустого места. Каждый объект представляется пикселем, так что заполнение графика происходит максимальным образом.
- Отсутствие перекрытий. Так как каждый объект занимает определенное место на графике, возможность перекрытия одним образцом другого пропадает – визуализация более понятна для восприятия.
- Цветовая интерпретация понятна и легко читаема человеком.
- Можно визуализировать действительно огромные наборы данных.

Минусы метода:

- Необходимо найти функцию расположения пикселей такую, чтобы между ними не было пересечений, но в то же время чтобы сохранялась понятность графика.
- В одном графике, как правило, представляется одно свойство набора данных. Возникает необходимость строить несколько графиков для охвата всех интересующих атрибутов.

Пример использования графика представлен на рисунке 2.1, на котором отображены данные о суточном водном потоке в городе Аламида за промежуток времени длиной в сто лет.

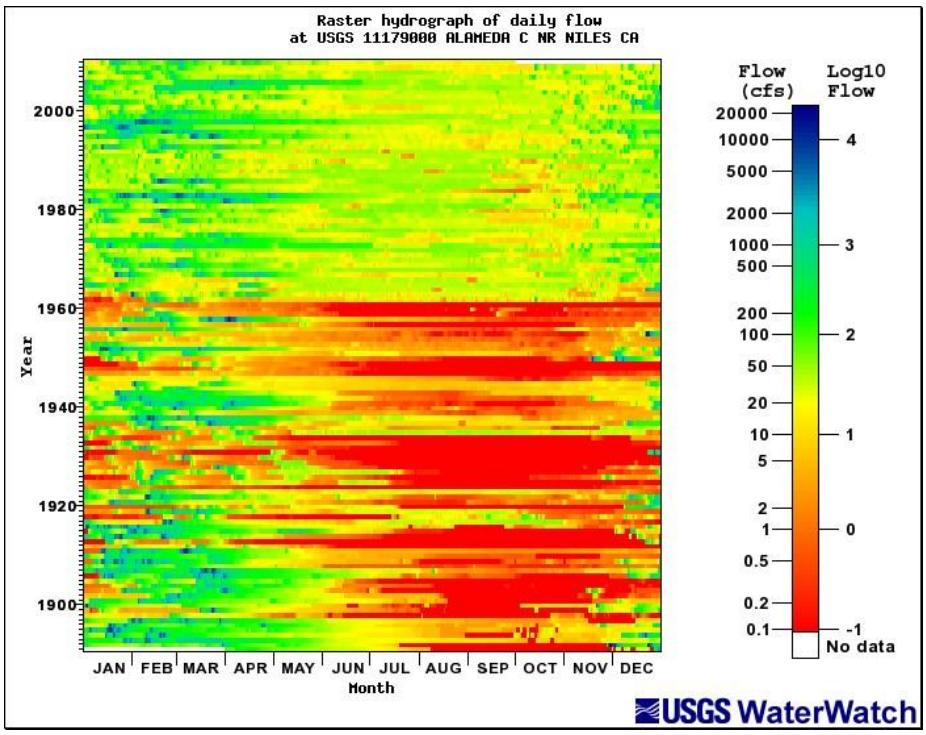

Рисунок 2.1. Пример использования пиксельных графиков

2.2 Свойства пиксельного графика

Все пиксельные методы визуализации разделяют экран на несколько областей – окон. Для набора данных размерности m (то есть каждый образец из данных имеет m измерений (атрибутов, свойств)) экран разделяется на m окон – по одному на каждое измерение. В некоторых случаях происходит разделение на $m+1$ окно – дополнительное $(m+1)$-ое окно используется для отображения какой-либо результирующей функции по всем остальным свойствам. Внутри окон значения данных упорядочены в соответствии с заданной общей сортировкой (рис.2.2). Корреляции, функциональные зависимости и другие интересующие отношения между измерениями могут быть обнаружены путем связывания соответствующих областей в нескольких окнах.

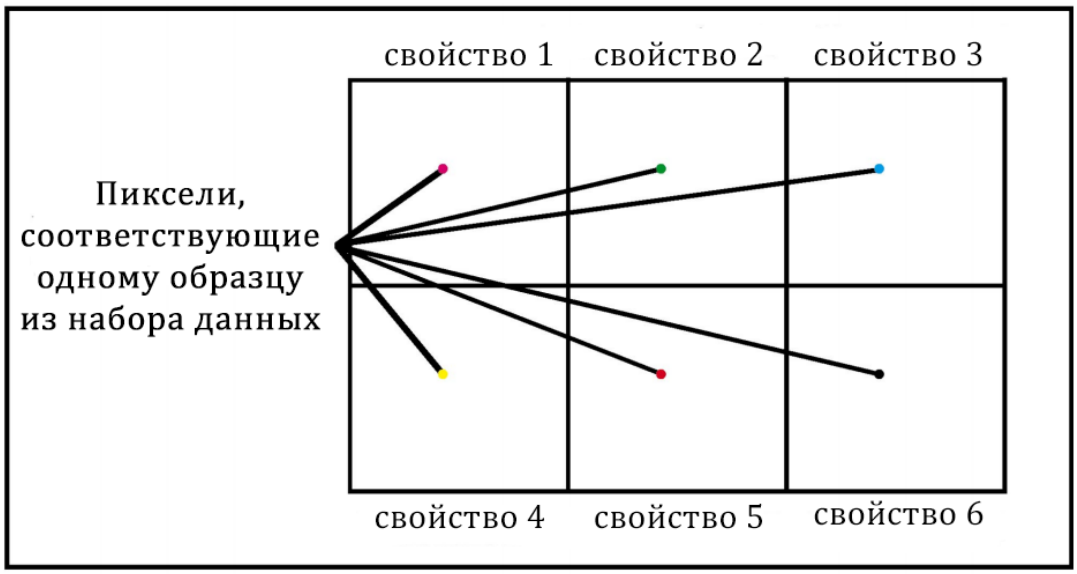

Рисунок 2.2. Представление образца данных на пиксельном графике

При этом возникает несколько вопросов в построении графика:

1. Сопоставление значений данных с цветами: нужно тщательно спроектировать отображение значений на графике, чтобы оно стало интуитивно понятным.
2. Расположение (компоновка) пикселей внутри окон. Решение этого вопроса зависит от входных данных и задачи визуализации. В большинстве случаев, проблема расположения может быть формально описана как проблема оптимизации, и различные техники визуализации подходят для различных задач оптимизации.
3. Форма окон. Например, при прямоугольной форме окон, как показано на рисунке 2.1, для наборов данных с большим количеством атрибутов окна для разных измерений иногда расположены далеко друг от друга, из-за чего становится трудно найти какие-либо связи между этими атрибутами. Как и в предыдущем пункте, проблема формы окон может быть представлена как проблема оптимизации.
4. Упорядочивание окон для измерений. В большинстве случаев не существует определенного порядка расположения окон измерений на графике. Чтобы обнаружить зависимости и корреляции между атрибутами, представленными в подокнах, нужно разместить связанные измерения рядом друг с другом. Из этого снова следует проблема оптимизации.

В этой работе мы остановимся на более подробном рассмотрении только второй проблемы.

2.2.1 Расположение пикселей

В этом пункте рассмотрим проблему расположения пикселей более подробно. Данный вопрос является одним из наиболее важных, поскольку только хорошее расположение позволит обнаружить кластеры или корреляции между измерениями. Для рассмотрения проблемы компоновки нужно различать наборы данных, которые имеют естественный порядок объектов (например, зависящие от времени данные) и наборы данных без естественного порядка.

- Определим проблему расположения пикселей в пиксельном графике для набора данных с естественным порядком объектов.

Проблема компоновки пикселей для множества объектов $\{a_1, a_2, \dots, a_n\}$, расположенных в соответствии с их естественным порядком, заключается в нахождении отображения этого множества на окно размера $(w \times h)$, т.е. биективного отображения $f: \{1, 2, \dots, n\} \to \{1, 2, \dots, w\} \times \{1, 2, \dots, h\}$, такого, чтобы минимизировать функцию

$\sum_{i=1}^{n} \sum_{j=1}^{n} \left| d\big(f(i), f(j)\big) - d\left((0,0), \left(w \cdot \sqrt{\frac{|i-j|}{n}},\ h \cdot \sqrt{\frac{|i-j|}{n}}\right)\right)\right|$, где

$d\big(f(i), f(j)\big)$ - L^p-расстояние между пикселями, соответствующими элементам a_i и a_j.

Таким образом, мы должны найти функцию, при которой минимизируется суммарное расстояние между получившимися пикселями относительно минимального расстояния двух пикселей в прямоугольном окне размера $(w \times h)$.

Задача отображения упорядоченных одномерных наборов данных в двумерное измерение рассматривалась математиками еще в прошлом веке. Так называемые *заполняющие пространство кривые* решают в точности выше указанную задачу оптимизации и обеспечивают минимизацию заданной функции.

Семейство заполняющих пространство кривых, или *кривые Пеано* – семейство параметрических кривых, образ которых содержит квадрат. Другими словами, кривая из этого семейства представляют собой непрерывное отображение из единичного отрезка в единичный квадрат, то есть проходит через любую точку этого квадрата. Пример итераций кривой Пеано представлен на рисунке 2.3.

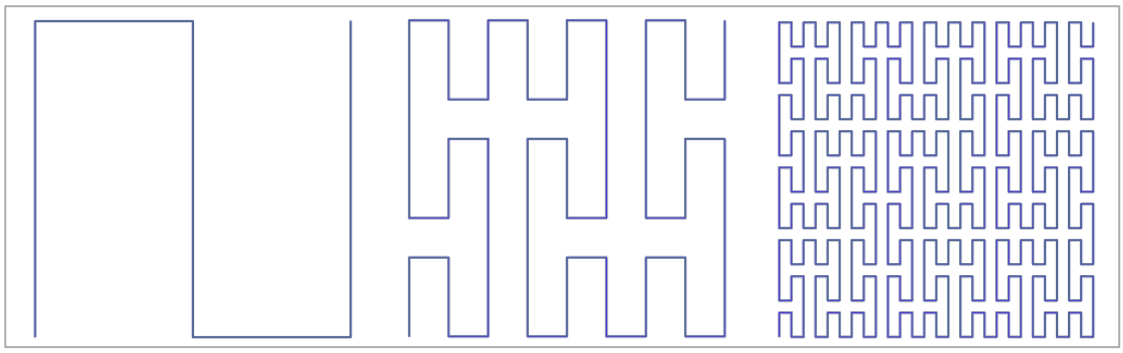

Рисунок 2.3. Три итерации кривой Пеано

Это значит, что если обозначить через (x, y) координаты точки в единичном квадрате, а через d расстояние вдоль кривой, на котором эта точка достигается, то точки, имеющие близкие к d значения, будут иметь также близкие значения и к (x, y). Обратное не всегда верно — некоторые точки, имеющие близкие координаты (x, y), имеют достаточно большую разницу в расстоянии d.

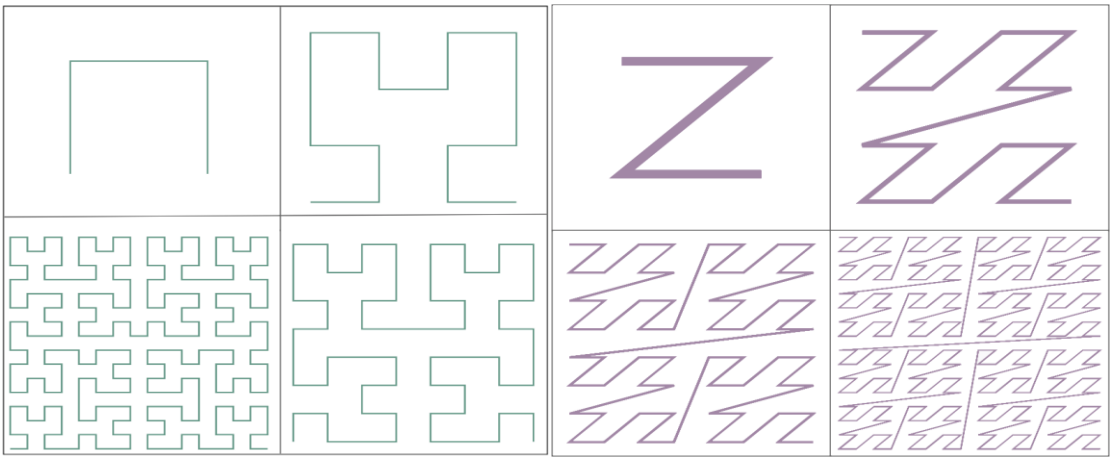

Рисунок 2.4. Четыре итерации кривой Гильберта (слева) и кривой Мортона (справа)

Примерами заполняющих пространство кривых являются кривые Пеано, Мортона и Гильберта (рис 2.4).

В двумерном пространстве линия, образующая участок кривой Гильберта заполняет 4 ячейки сетки, то есть блок 2x2 и имеет форму буквы U, а линия, образующая кривую Мортона форму буквы Z. Каждый последующий уровень кривой Гильберта или Мортона делит каждую из ячеек сетки на 4, а линии, формирующие участки кривых, имеют форму повернутых букв U и N, что является свойством самоподобия.

Все кривые Пеано являются фрактальными кривыми, что означает, что для их реализации можно использовать рекурсию.

- Для данных, которые не имеют естественного порядка, предлагается следующий выход. В этом случае главной целью визуализации является представление на графике только тех значений, которые удовлетворяют некоторому входящему *запросу*. Простой запрос для m-размерного набора данных может соответствовать точке в m-мерном пространстве; если задан диапазон запроса, то ему будет соответствовать область в m-мерном пространстве. Результатом запроса являются объекты из набора данных, которые находятся в области запроса.

Результат запроса может оказаться очень большим или пустым – в большинстве случаев количество подходящих объектов предсказать невозможно. Чтобы получать более информативные результаты от запроса, на графике можно изображать не только объекты, которые входят в область запроса, но и объекты, которые расположены «близко» к этому запросу.

Понятие «близко» может быть определено дополнительной функцией, которая и будет определять расстояние от объекта до запроса. Если объект принадлежит области запроса, расстояние равняется 0. Для числовых и временных типов данных можно использовать обычные функции расстояния: модуль и, например, количество секунд от одной даты до другой.

Таким образом, мы можем добавить $(m + 1)$-ый атрибут к объектам – расстояние до входящего запроса. Теперь мы имеем набор данных с естественным порядком – в соответствии с полученным расстоянием. А значит, к нему можно применить техники из предыдущего пункта – кривые Пеано.

Проблема заключается в том, что интуитивно предполагается, что объекты, которые удовлетворяют запросу, должны находиться в середине графика, в то время как кривые Пеано располагают наиболее подходящие элементы в левом нижнем углу. Однако эту проблему можно решить, к примеру, следующим образом, который представлен на рисунке 2.5.

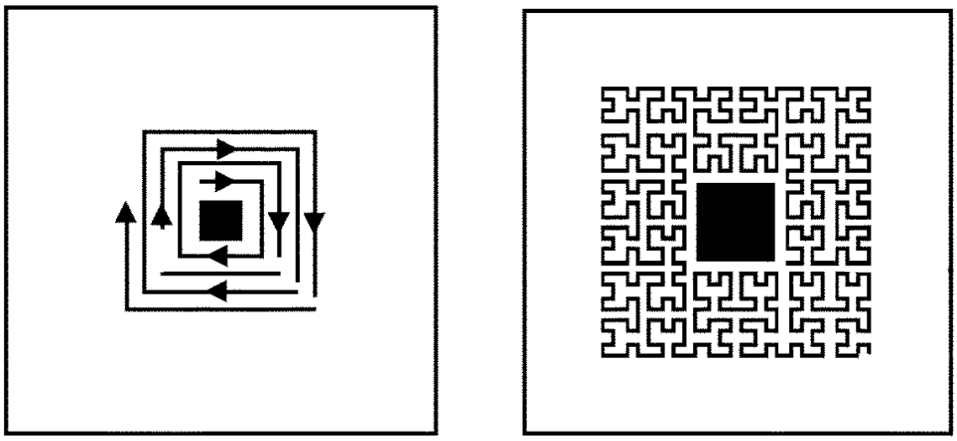

Рисунок 2.5. Варианты расположения объектов, удовлетворяющих запросу – спиральный обход и обход по кривой Гильберта

Стоит отметить, что вариант с простой спиральной компоновкой (на рис. 2.5 слева) недостаточно хорошо выполняет требование, сформулированное в предыдущем пункте – а именно, локальность объектов. Данные, которые в одномерном пространстве находились рядом, на графике могут располагаться далеко друг от друга. Поэтому решением может являться опять же использование кривых Пеано.

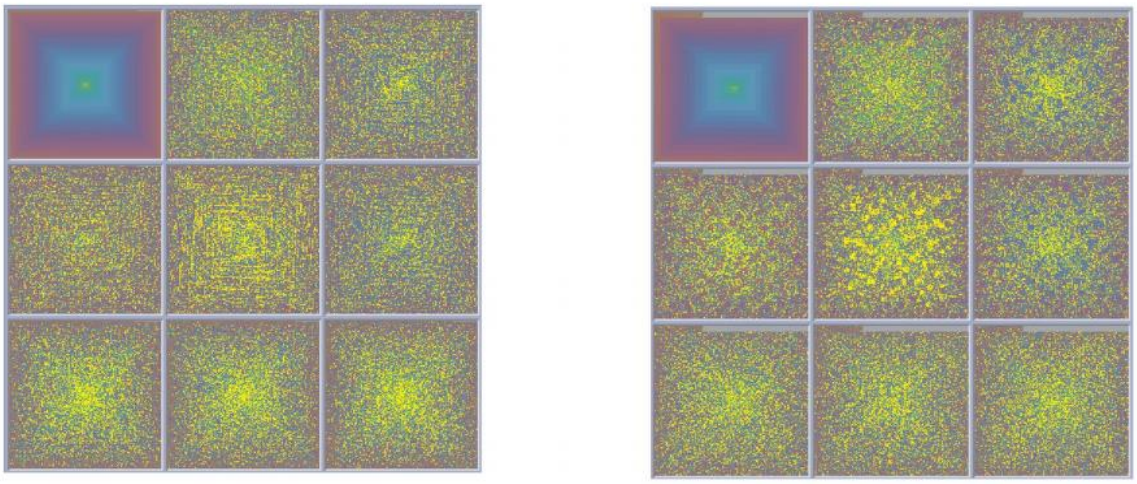

Рисунок 2.6. Сравнение обходов по спиральной кривой (слева) и по кривой Гильберта (справа)

Как видно на рисунке 2.6, кластеризация заметна более хорошо в случае использования заполняющих пространство кривых.

Стоит отметить, что в использовании кривых Пеано заключается следующий минус: для того, чтобы полностью заполнить квадрат, длина его стороны должна являться степенью числа 2. Это значит, что в плохих

случаях (когда количество данных немного больше степени числа 4) размеры окна становятся большими, но график в окне не заполняется полностью; либо не все объекты из набора будут визуализированы.

2.3 Выводы

В главе были подробно рассмотрен тип пиксельных графиков, их применение, достоинства и недостатки. Рассмотрен наиболее важный параметр для этого типа графиков – компоновка пикселей на графике, а также варианты решения проблемы компоновки при помощи кривых Пеано.

3. РЕАЛИЗАЦИЯ ПОСТРОЕНИЯ ПИКСЕЛЬНОГО ГРАФИКА

3.1 Этапы получения визуализации

Для того, чтобы визуализировать какой-либо набор данных, нужно провести несколько этапов подготовки:

1. Получение размеченных данных

Пиксельные методы визуализации используются для больших объемов данных, поэтому размеченные данные могу быть получены разными способами:

- как результат сбора статистики (например, компания собирает данные о проведенных кибер-атаках на свою систему);
- получены от источников данных в результате ответа на соответствующий запрос.

2. Предобработка данных

Для визуализации необходимо указать, по меньшей мере, некоторые особенности полученных данных:

- определить тип для каждого свойства данных: какие из свойств данных являются непрерывными, а какие – дискретными;
- задать параметры для визуализации – для пиксельных графиков это могут быть: тип кривой, параметры для выбранного типа кривой; цвета для каждого графика;
- в некоторых случаях приходится обрабатывать данные, чтобы они стали более понятными для анализа и для системы. Например, ip-адреса могут быть конвертированы в страны, которым они принадлежат.

3. Проведение расчетов

По полученным данным, их свойствам и параметрам, которые были введены пользователем, программа для каждого поля определяет положение и цвет всех элементов выборки на графике.

4. Построение графика

По полученным данным программа строит несколько пиксельных графиков – для каждого из свойств.

3.2 Параметры и реализация визуализации

Определим параметры реализуемой визуализации:

- Цвета для отображения значений будет зависеть от двух цветов, которые задает пользователь, – начального и конечного.
    - В случае, когда атрибут представляет собой дискретную величину, разделим цветовой диапазон от начального до конечного цвета на количество уникальных элементов во входных данных. Далее каждому значению из уникальных сопоставим цвет из получившегося градиента (рисунок 3.1, слева).
    - Для непрерывных величин пользователю будет предлагаться ввести количество шагов, на которые будет разбит диапазон значений. Цветовой диапазон разбивается на такое же количество шагов, и эти два разбиения сопоставляются между собой. Таким образом, если рассматриваемая величина попадает в i-ый шаг диапазона значений, она будет отображена i-ым цветом в градиенте (рисунок 3.1, справа).
- Прямоугольная форма окна.
- Упорядочивание окон для измерений будем проводить по порядку поступления атрибутов данных.
- Горизонтальное расположение пикселей и расположение по кривой Гильберта. При горизонтальном расположении пользователь может задать размеры графика; при выборе кривой Гильберта – длину данных, которые нужно визуализировать.
- Будем рассматривать наборы данных с естественным порядком.

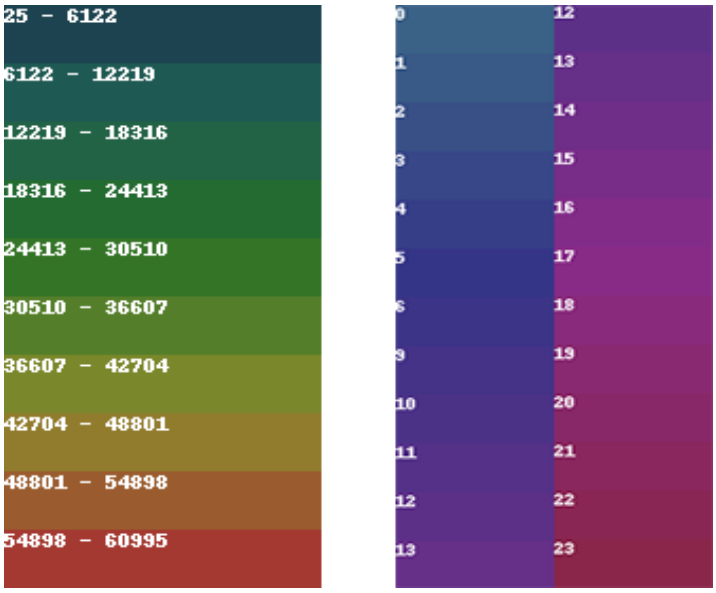

Рисунок 3.1. Пример цветовой легенды для непрерывной (слева) и дискретной (справа) величин

Покажем реализацию общей функции, которая используется для создания графика непрерывной и дискретной величин на рисунке 3.2.

```
def final_get_pict(data, size, max_len, xy_fun, color_fun, dict_colors=None):
    im = Image.new(mode='RGB', size=size, color='white')

    for i, elem in enumerate(data):
        xy_ = xy_fun(i)
        idx = color_fun(elem)
        color = dict_colors[idx]
        im.putpixel(value=color, xy=xy_)

        if i == max_len - 1:
            break

    return im
```

Рисунок 3.2. Общая функция для непрерывной и дискретной величин

Как можно видеть, параметрами в ней являются:

- список данных, отсортированных в нужном порядке;
- размер окна (для варианта горизонтальной компоновки);
- максимальный размер данных, которые будут отображаться;
- функция, которая определяет положение пикселя на графике по порядковому номеру в исходных данных; зависит от типа кривой компоновки;
- функция, которая определяет, каким образом сопоставить текущему элементу цвет из словаря цветов; зависит от типа величины;
- словарь цветов – список из цветов полученной ранее цветовой легенды.

Таким образом, для последующих дополнений функционала нужно будет указать только функцию компоновки пикселей на графике.

Покажем реализацию получения координат по порядковому номеру:

- Для горизонтального типа кривой (рис 3.3):

```
def get_horizontal_xy(width):
    def horizontal_xy(i):
        return i % width, int(i / width)

    return horizontal_xy
```

Рисунок 3.3 Реализация вычисления координат объекта для горизонтального типа кривой

- Для кривой Гильберта (рис. 3.4):

```
def coordinates_from_distance(h, p, N):
    x = _hilbert_integer_to_transpose(h, p, N)
    Z = 2 << (p-1)

    t = x[N-1] >> 1
    for i in range(N-1, 0, -1):
        x[i] ^= x[i-1]
    x[0] ^= t

    Q = 2
    while Q != Z:
        P = Q - 1
        for i in range(N-1, -1, -1):
            if x[i] & Q:
                # invert
                x[0] ^= P
            else:
                # exchange
                t = (x[0] ^ x[i]) & P
                x[0] ^= t
                x[i] ^= t
        Q <<= 1

    return x
```

Рисунок 3.4 Реализация вычисления координат объекта для кривой Гильберта

Таким образом, было реализовано получение координат и значение цвета каждого элемента по его порядковому номеру и по параметрам модели, заданным пользователем, а также получение изображения по полученным данным.

3.3 Примеры визуализации вредоносного программного обеспечения

3.3.1 Пример визуализации спам-атак

Для визуализации был использован набор данных о спам-атаках – пакеты трафика, которые поступали от нескольких коллекторов Netflow v9 в сети испанского провайдера за март – июнь 2016 года.

time	duration	source_ip	dest_ip	source_port	dest_port	protocol	tsp_flags	tos	packets_sent	bytes_sent	verdict
18.03.2016 13:16	0.184	192.143.84.56	42.219.156.215	25	46163	TCP	.AP.SF	0	3	230	anomaly-spam
18.03.2016 13:42	1.132	108.66.255.194	42.219.156.213	25	44328	TCP	.AP.SF	0	13	1267	anomaly-spam
18.03.2016 13:42	1.648	192.143.84.56	42.219.156.212	25	48987	TCP	.AP.SF	0	15	1371	anomaly-spam
18.03.2016 13:42	0.122	108.66.255.194	42.219.156.215	25	46433	TCP	.AP.SF	0	13	1267	anomaly-spam
18.03.2016 14:16	0.184	108.66.255.199	42.219.156.214	25	55753	TCP	.AP.SF	0	3	234	anomaly-spam
18.03.2016 14:33	0.264	192.143.87.120	42.219.156.223	25	60148	TCP	.AP.SF	0	3	234	anomaly-spam
18.03.2016 15:17	0.26	108.66.255.255	42.219.156.223	25	37275	TCP	.AP.SF	0	3	234	anomaly-spam
18.03.2016 15:39	0.148	192.143.87.120	42.219.156.223	25	34250	TCP	.AP.SF	0	3	234	anomaly-spam
18.03.2016 16:59	0.124	192.143.87.90	42.219.156.214	25	38997	TCP	.AP.SF	0	3	234	anomaly-spam
18.03.2016 17:38	1.316	108.66.255.250	42.219.156.223	25	53436	TCP	.AP.SF	0	12	1233	anomaly-spam
18.03.2016 17:42	0.156	192.143.87.124	42.219.156.223	25	34412	TCP	.AP.SF	0	3	230	anomaly-spam

Рисунок 3.5. Часть входных данных о спам-атаках

Пример набора показан на рисунке 3.5, в котором присутствуют поля:

- Время начала потока данных
- Продолжительность потока данных
- IP-адрес отправителя и получателя
- Порт источника для UDP или TCP, 0 для других протоколов
- Порт назначения для UDP или TCP, тип и код для ICMP или 0 для других протоколов
- IP протокол
- Флаги, включенные при TCP протоколе
- Тип обслуживания
- Количество переданных пакетов и байтов
- Вердикт

Была создана выборка размером в $4^7 = 16348$ элемента, состоящая из случайных записей из каждого месяца из указанного промежутка. Данные считаются упорядоченными по времени начала потока данных. Используется компоновка по кривой Гильберта.

1. Часы в течении дня, когда начинались потоки данных – рисунок 3.6. По графику можно сделать вывод, что в большинстве случаев атаки выполнялись либо в промежутке времени от 20 до 23 часов, либо в промежутке от 11 до 13 часов. Меньше всего атак происходило в районе полуночи. Кластеры в данном случае не несут в себе пользы, так как скорее всего они образовались из-за последовательных отправлений пакетов в одно и то же время. Справа на рисунке 3.6 представлена легенда для графика.

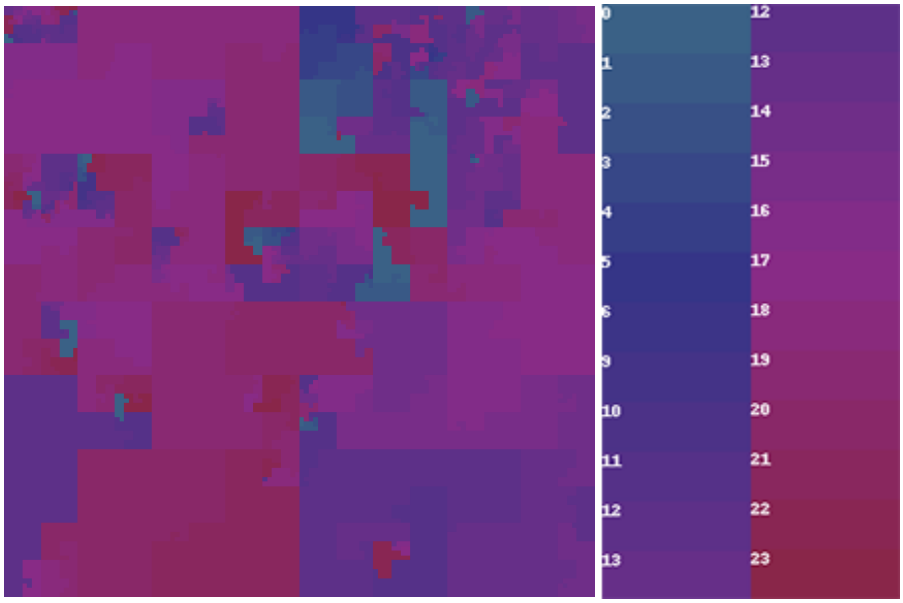

Рисунок 3.6. Визуализация часов начала потока данных

2. Страны отправителя и получателя представлены на рисунке 3.7. При помощи пакета ipwhois из колонок «source_ip» и «dest_ip» были получены коды стран, из которых проводилась атака. Для этих двух графиков был создана общая цветовая легенда (рис. 3.8) для лучшего восприятия.

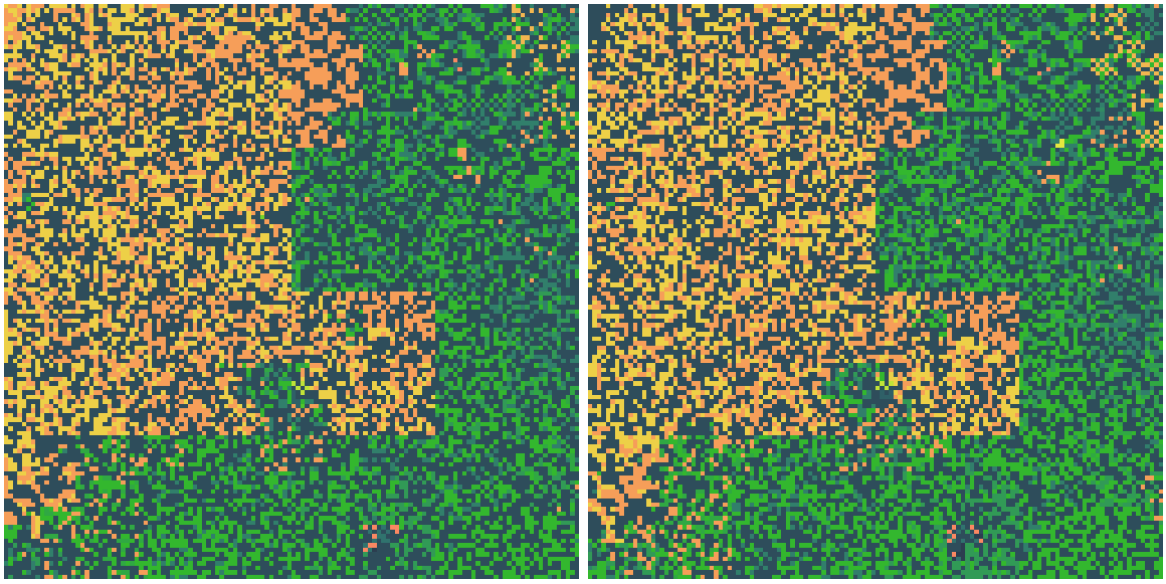

Рисунок 3.7. Визуализация стран: отправителя (слева) и получателя (справа)

Легко можно заметить, что графики очень похожи – это значит, что в большинстве случаев атака направлялась в ту же страну, откуда и исходила. Можно также отметить, что за весь промежуток времени стабильно атаковался (и атаковал) Китай, в первой половине набора много атак из Южной Африки и США, а во второй их заменили Корея и Германия.

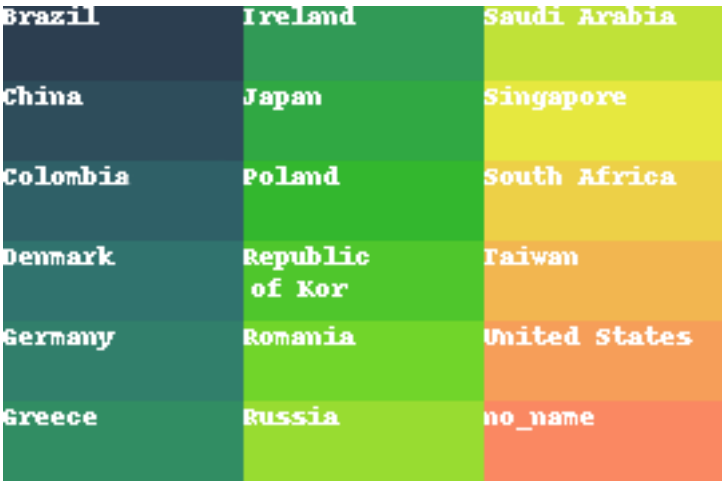

Рисунок 3.8. Цветовая легенда для стран отправителей и получателей

4. Продолжительность потоков данных представлена на рисунке 3.9. В этом пункте данные представляют собой непрерывную величину. Было выбрано число шагов градиента, равное 50. Даже при таком большом количество шагов для относительно маленького диапазона (от 0 до 340 мс) почти весь график заполнен одним цветом, что означает преобладание низкой продолжительности потока. Существует только небольшой участок, где и достигается максимальная продолжительность потока данных.

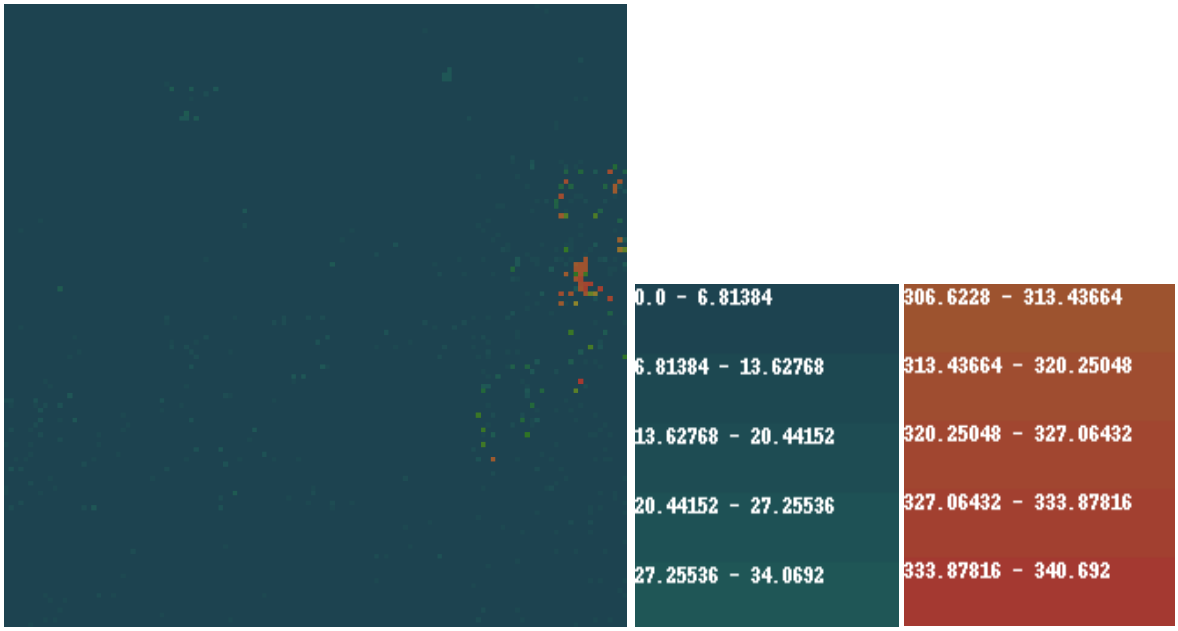

Рисунок 3.9. Визуализация продолжительности потока данных

5. Использование флагов при TCP/IP протоколе. Как было вычислено, все из представленных образцов использовали TCP протокол при передаче данных. Теперь можно посмотреть на флаги, которые выставлялись при этих

передачах, используя график, представленный на рисунке 3.10. Но прежде ознакомимся с ними:

- URG — поле «Указатель важности» задействовано;
- ACK — поле «Номер подтверждения» задействовано;
- PSH —инструктирует получателя протолкнуть данные, накопившиеся в приёмном буфере, в приложение пользователя;
- RST — оборвать соединения, сбросить буфер (очистка буфера);
- SYN — синхронизация номеров последовательности;
- FIN — флаг, будучи установлен, указывает на завершение соединения.

На графике можем заметить такое же разделение, как и на предыдущих, что значит, что для каждой половины набора данных характерны определенные свойства, отличающиеся от другой половины. Вверху заметен особенный участок, в котором просматривается узор – скорее всего, эти пакеты посылались почти в одно время и этот промежуток времени является одной продолжительной атакой.

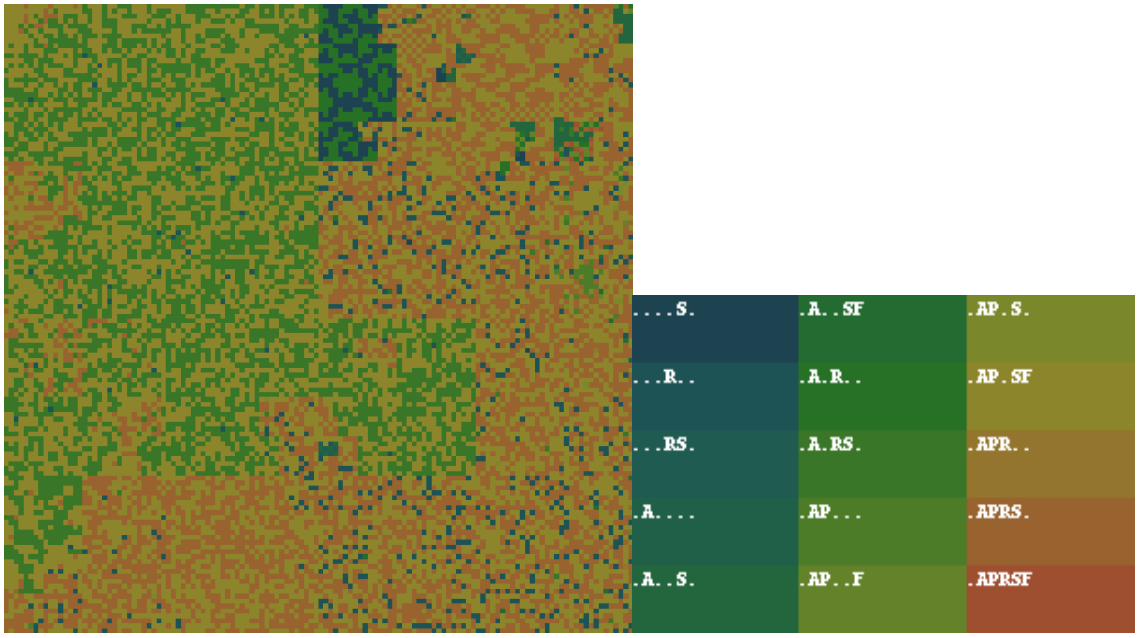

Рисунок 3.10. Визуализация установленных флагов при TCP соединении

Теперь составим пиксельный график в таком виде, каким он является по определению – состоящим из окон, каждое из которых соответствует одному свойству. Пять первых окон отображают выше указанные атрибуты, шестое – количество отправленных пакетов. Получившийся результат можно увидеть на рисунке 3.11.

Сравнивая окна между собой, можно найти некоторые корреляции между, например, первыми тремя, а также между четвертым и шестым.

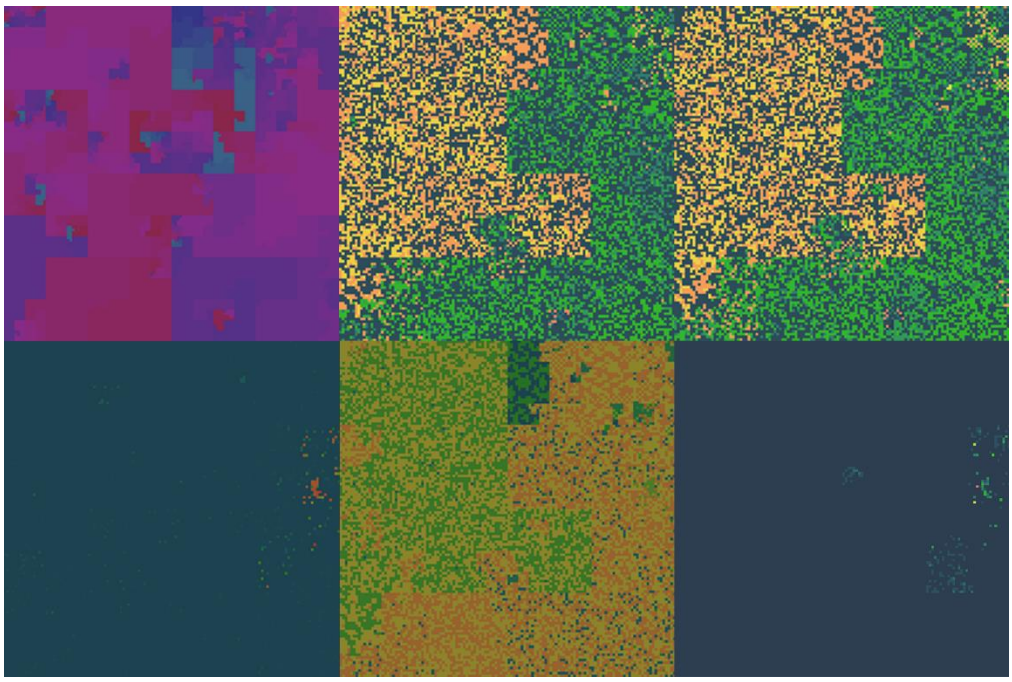

Рисунок 3.11. Пиксельный график, построенный по набору данных о спам-атаках

3.3.2 Пример визуализации файлов вредоносных программ

Для визуализации исполняемых файлов вредоносного ПО был использован набор данных, содержащий в себе последовательность вызовов функций или API-вызовов при исполнении соответствующей программы (пример показан на рисунке 3.12). В данном наборе содержались как вредоносные программы, так и безвредные.

	A	B	C	D	E	F	G	H	I	J
1	1,LoadLibraryW HeapAlloc HeapAlloc HeapFree HeapAlloc HeapFree HeapFree NtOpenKey LoadLibraryW									
2	1,RegOpenKeyExW LoadLibraryA GetProcAddress GetProcAddress GetProcAddress GetProcAddress GetP									
3	1,HeapAlloc HeapFree HeapAlloc HeapAlloc HeapFree HeapFree IsBadReadPtr IsBadReadPtr IsBadReadP									
4	1,HeapAlloc HeapFree HeapAlloc HeapAlloc HeapFree HeapFree IsBadReadPtr IsBadReadPtr IsBadReadP									
5	1,HeapAlloc HeapFree HeapAlloc HeapAlloc HeapFree HeapFree IsBadReadPtr IsBadReadPtr IsBadReadP									

Рисунок 3.12. Образец входных данных

Для всех объектов из файла была построена общая цветовая легенда, чтобы визуализация была информативной. Так как уникальных названий вызовов очень много, градиент получился очень плавный, что мешает четкому разделению вызовов по цветам. Однако даже при этом условии визуализация дала интересные результаты.

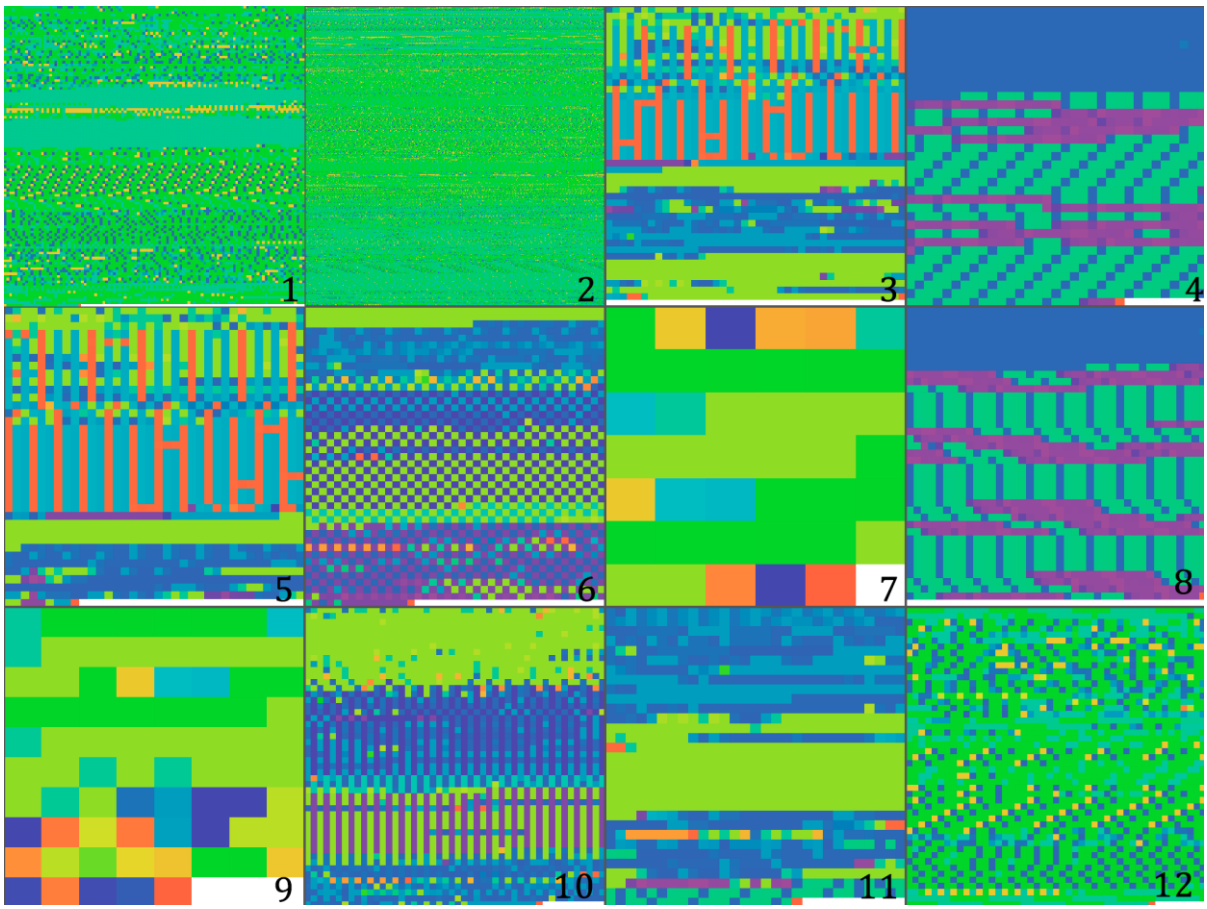

Рисунок 3.13. Результаты визуализации системных вызовов вредоносных программ

Для этого примера была использована компоновка пикселей горизонтальными линиями, чтобы показать и её пользу. К тому же, компоновку по кривой Гильберта использовать было бы неэффективно, потому что количество вызовов в каждом образце сильно отличается друг от друга, соответственно, получались графики значительно разных размеров и не заполненные до конца. Ширина окна для каждого образца выбиралась примерно равной квадратному корню из размера набора его вызовов (чтобы в итоге получался квадрат).

На рисунке 3.13 представлено несколько полученных образцов. Здесь можно увидеть пары очень похожих вредоносных программ – таких, как 3 и 5 или 4 и 8. Таким образом, без особых усилий можно найти семейства вредоносных программ, т.е. провести кластеризацию данных. При выборе фиксированной длины, можно было бы узнать, какие программы начинают своё выполнение одинаково, а какие нет.

Анализируя отдельный объект, можно легко понять, какие функции вызывались чаще всего и в каком порядке. Можно найти закономерности вызовов, а также составить общую картину действий программы.

Представленные два примера показывают, что пиксельный график может быть полезным при любом из типов анализа – индивидуальном, кластеризации или анализа большого набора данных – если правильно подобрать входные данные.

3.4 Выводы

В главе были рассмотрены этапы подготовки визуализации данных. Были представлены два примера, построенные при помощи реализованных методов, которые показывают, что пиксельный график может быть полезным при любом из типов анализа – индивидуальном, кластеризации или анализа большого объема данных.

4. РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ ДЛЯ ВИЗУАЛИЗАЦИИ ПРИ ПОМОЩИ ПИКСЕЛЬНЫХ ГРАФИКОВ

Для того, чтобы сделать анализ вредоносных программ более эффективным и удобным, необходимо реализовать интерактивность построенных графиков, а также графический интерфейс для выбора параметров модели. Поэтому к четырем пунктам процесса (глава 3 пункт 1) визуализации можно добавить пятый:

5. Обеспечить интерактивность графика во время исполнения программы.

Таким образом, теперь программа работает по сценарию, представленному на рисунке 5.1.

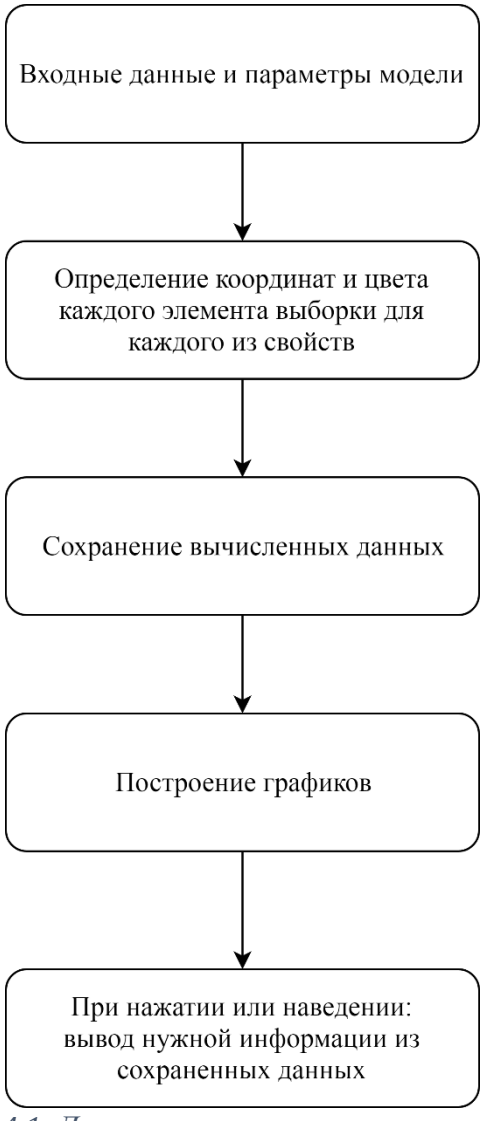

Рисунок 4.1. Диаграмма процесса визуализации

4.1 Описание взаимодействия пользователя с приложением

Исходя из структуры этапов реализации визуализации данных, представленных выше, для удобства пользователя были спроектированы следующие возможности.

В начале пользователю предоставляется окно для выбора месторасположения файла с данными, а также для выбора типа кривой, по которой будет строиться график (рис 4.2). На нынешнем этапе развития приложения считается, что на вход подаются уже обработанные пользователем данные в формате .csv.

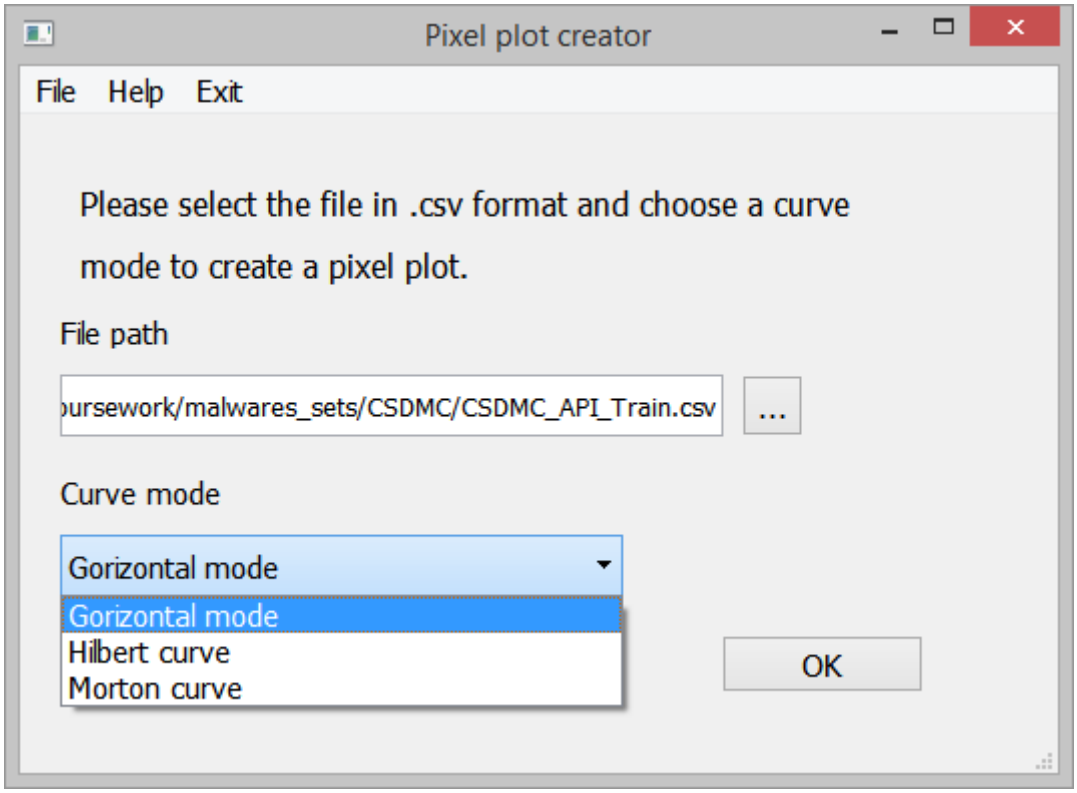

Рисунок 4.2. Окно выбора параметров визуализации

Можно выбрать файл при помощи диалогового окна (рис. 4.3).

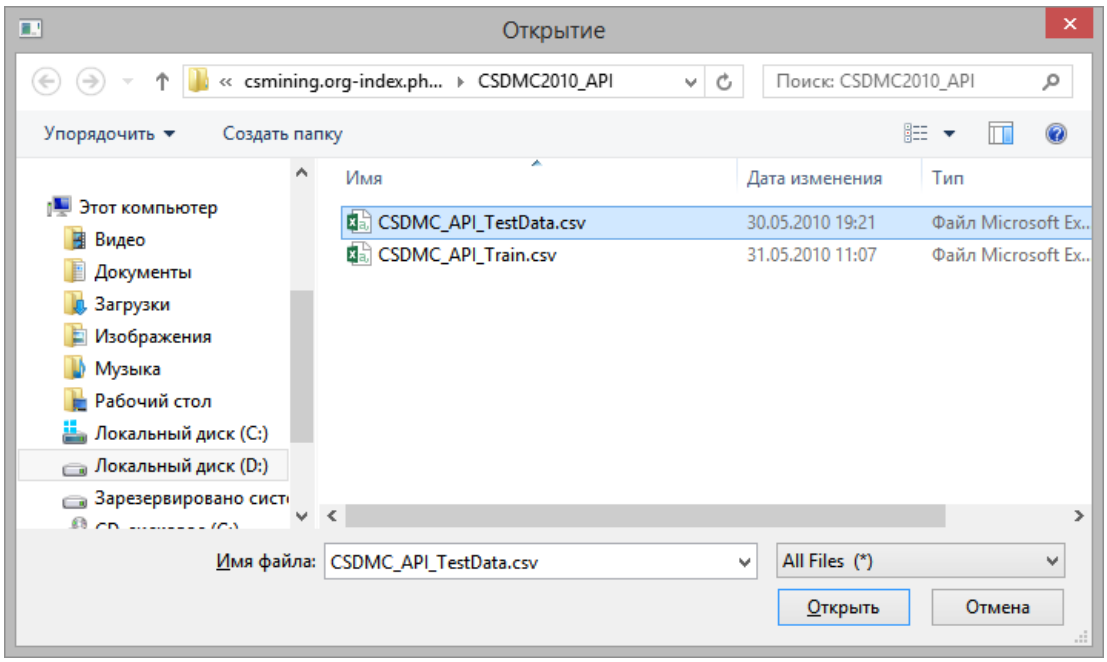

Рисунок 4.3 Диалоговое окно для выбора файла с данными

После этого пользователь должен определить тип свойств в полученных данных (рис 4.4). Если свойство есть непрерывная величина, то также указать количество разбиений промежутка значений.

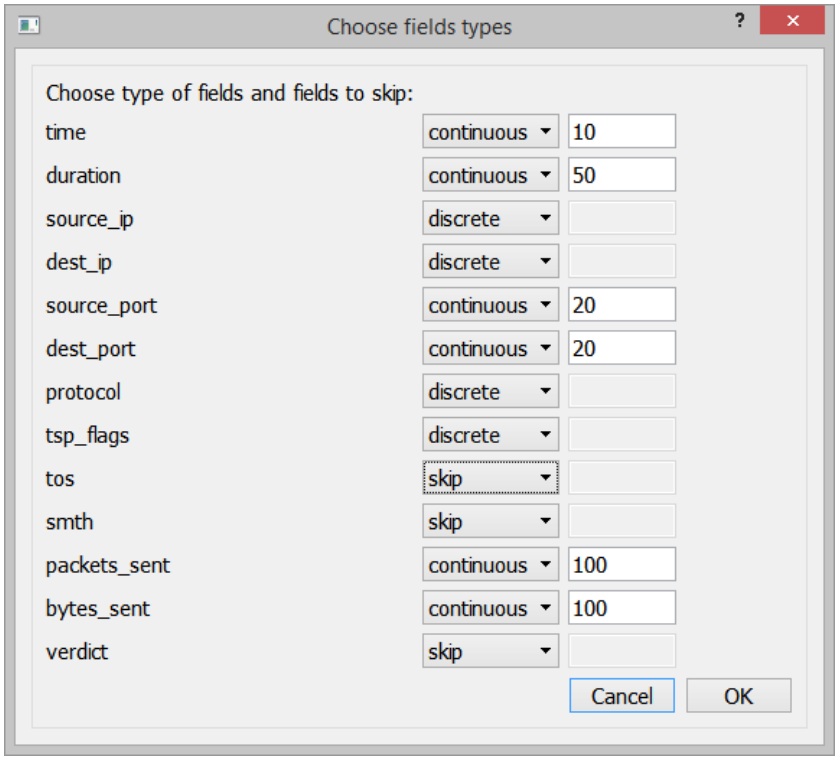

Рисунок 4.4. Диалог определения свойств атрибутов данных

После нажатия кнопки «ОК» появляется окно с графиками (рис. 4.5).

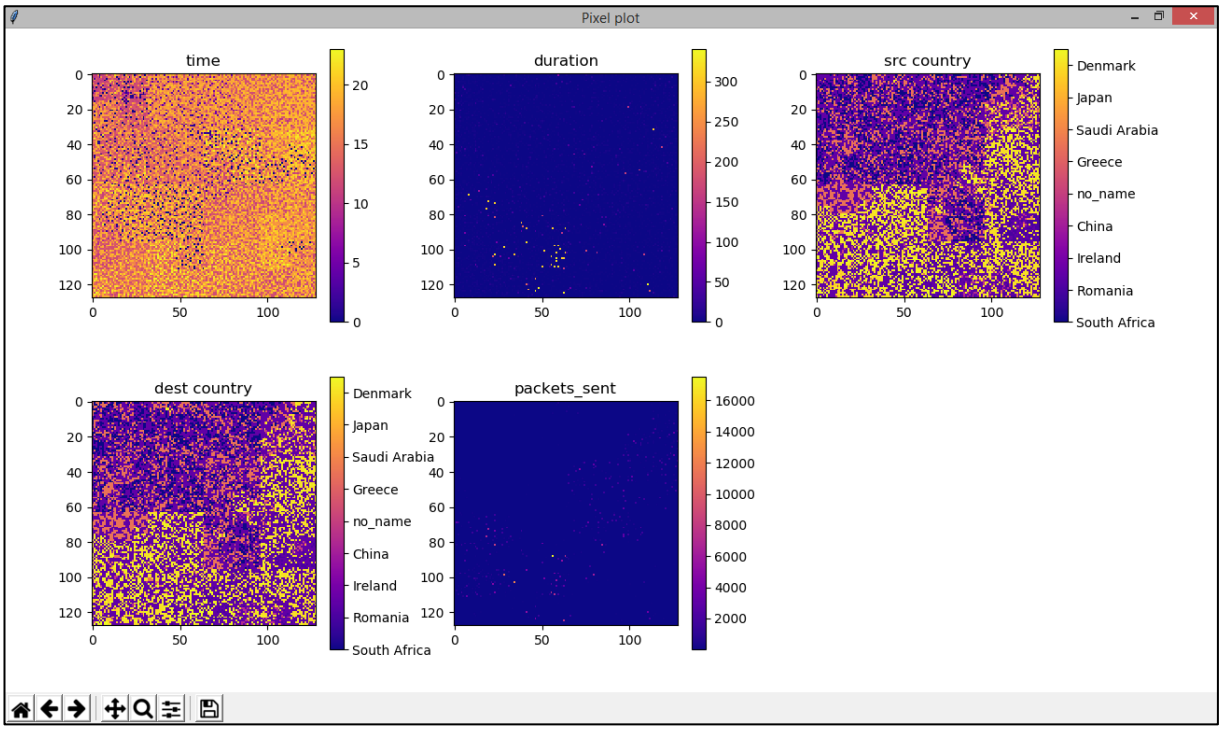

Рисунок 4.5. Окно программы с построенными графиками

Каждое из окон на графике показывает одно из полей полученных данных.

Существуют следующие возможности:

- Возле каждого свойства приведена своя цветовая легенда.
- При наведении на определенный пиксель (элемент) на любом из окон в статус-баре отображаются его координаты на пиксельном графике, а также вычисленное значение свойства, которому соответствует окно с текущим положением курсора, этого элемента (рис. 4.6).

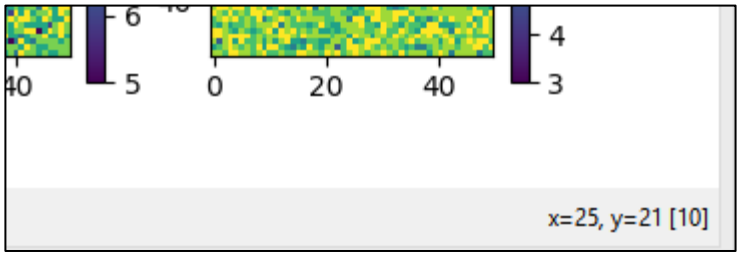

Рисунок 4.6. Статус-бар программы

- За курсором следует маленькое окно с указанием порядкового номера элемента в исходной выборке (рис. 4.7).

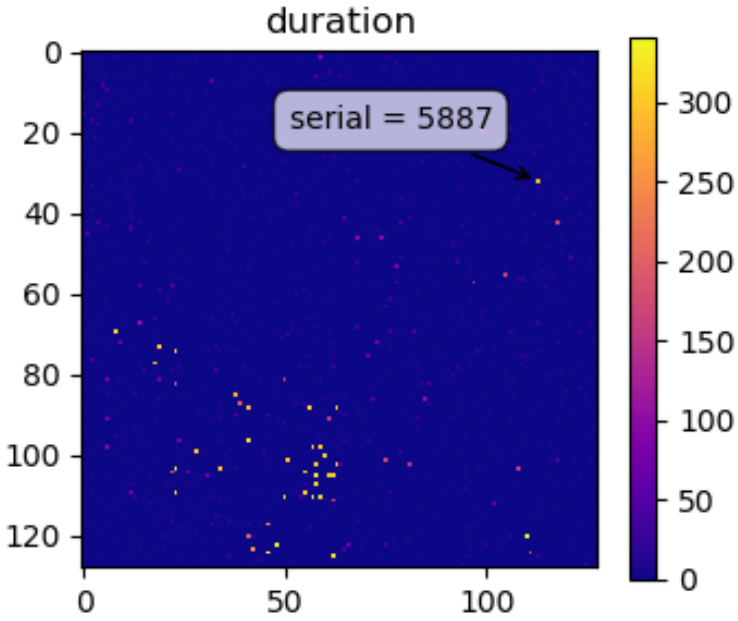

Рисунок 4.7. Курсор с окошком информации

- При нажатии на элемент появляется отдельное окно с полной информацией по данному элементу: его расположение в выборке, координаты на пиксельном графике и значения всех его свойств. Можно открыть несколько таких окон для разных элементов одновременно (рис 4.8).

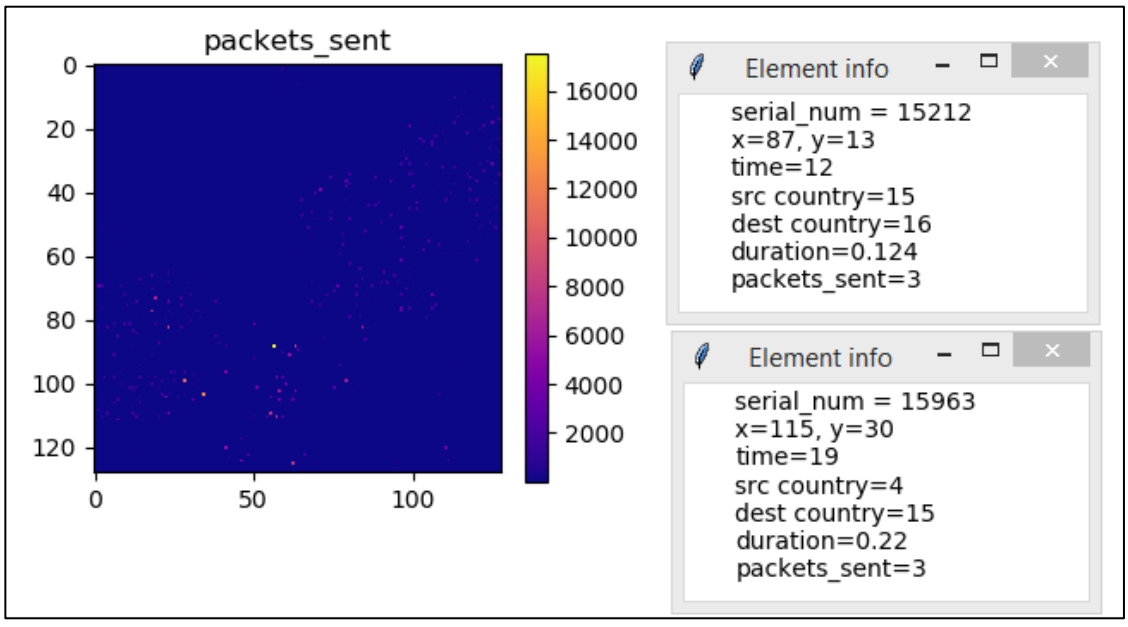

Рисунок 4.8.Окна с полной информацией об элементах

- В статус-баре расположены кнопки дополнительной функциональности, показанные на рисунке 4.9. При нажатии на значок лупы

можно изменить масштаб каждого графика независимо и приблизить (или отдалить) нужную часть графика прямоугольной формы (рис. 4.10).

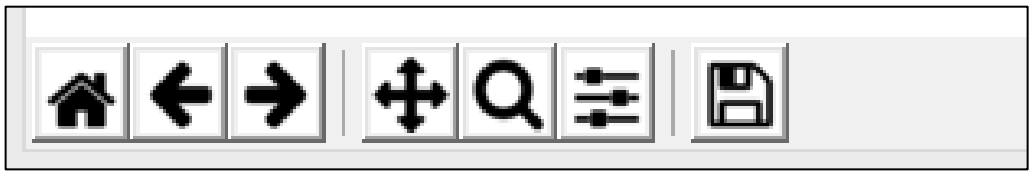

Рисунок 4.9. Кнопки в статус-баре программы

- При нажатии на значок четырех стрелок можно перетаскивать определенный график, если он находится в измененном масштабе.

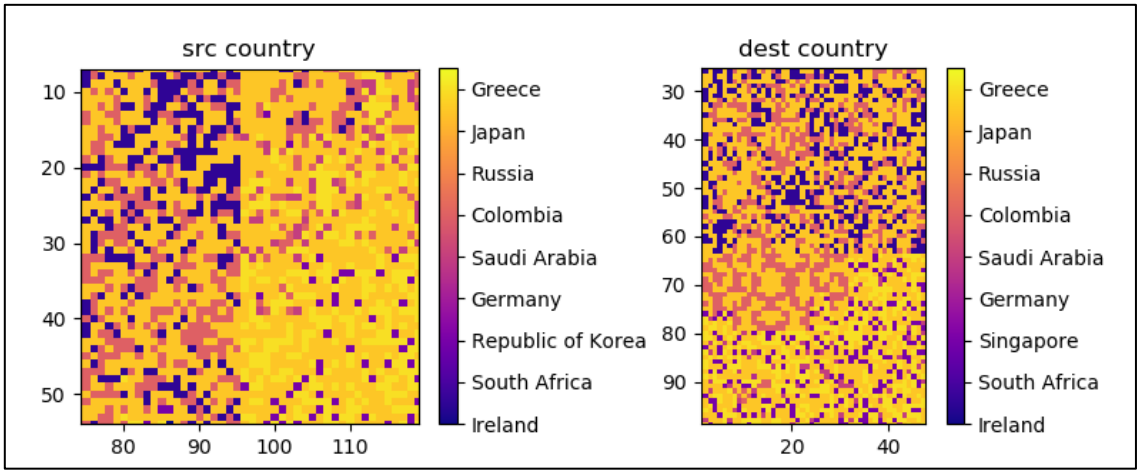

Рисунок 4.10. Окна с масштабированием разных частей графика

- При нажатии на значок домика все настройки масштаба возвращаются к исходным. По кнопкам с изображением стрелок можно переходить к предыдущему и следующему изменению масштаба или перемещения.
- Также можно настраивать расстояние между окнами, их общий масштаб.
- Окно приложения масштабируется также.
- Существует возможно сохранить полученный график в форматах png, jpg, svg и других (рис. 4.11).

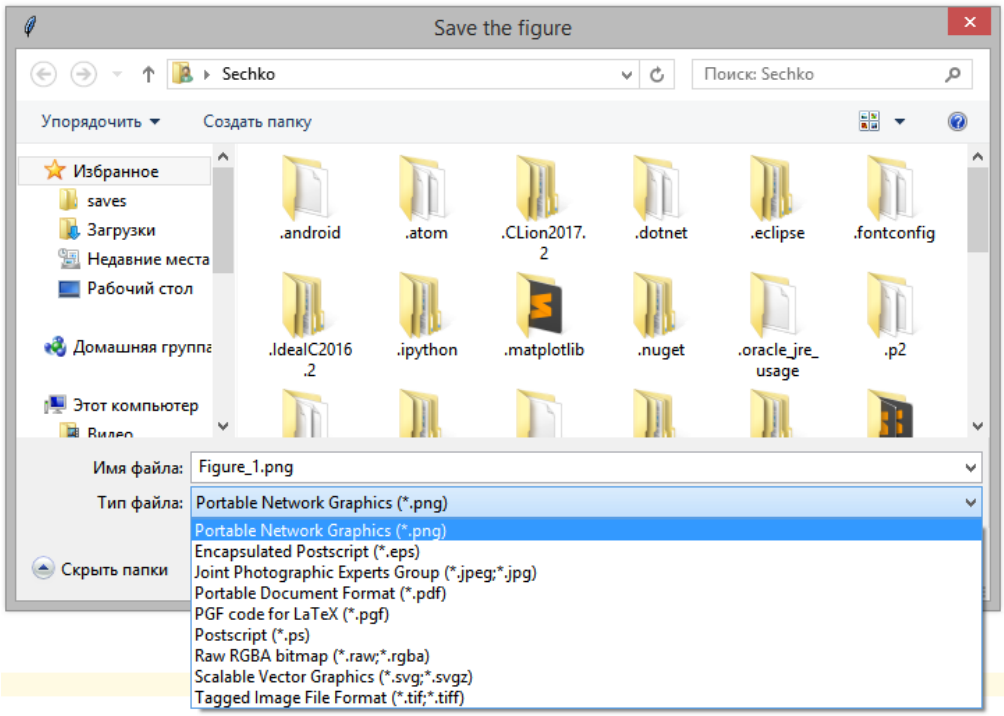

Рисунок 4.11. Диалог сохранения графика

4.2 Программная реализация

Для реализации приложения был использован графический фреймворк Qt для языка программирования Python и библиотеки matplotlib и pandas языка Python.

Qt представляет собой кросс-платформенную платформу приложений для разработки графических пользовательских интерфейсов (GUI) и кросс-платформенных приложений, работающих на всех основных операционных системах.

Интерактивность приложения обеспечивается следующим образом. При наведении на какой-либо элемент графиков по координатам курсора на всём холсте определяются координаты в одном из окон, тем самым идентифицируя элемент, который пользователь просматривает. После этого по запросу координат на графике из сохраненных данных возвращается вся информация о текущем элементе. Для реализации были переопределены методы ActionEvent для наведения курсора и нажатия кнопки мыши (рис. 4.12).

```
def onclick(event):
    x, y = xy_by_plot(event.xdata, event.ydata)
    ord_id = i_by_xy[x][y]

    info_fig = plt.figure(figsize=(2, 2))

    info_ax = info_fig.add_subplot(111)
    info_fig.canvas.toolbar.pack_forget()

    info_ax.text(0, 0, infos[ord_id], family='sans-serif')
    info_fig.show()
    pass
```

Рисунок 4.12 Переопределенный метод для вызова окна с информацией

Около каждого графика выделяется место для создания своей цветовой легенды.

Для возможности масштабирования были использованы стандартные методы фреймворка PyQT, с переопределением пространства для масштабирования – чтобы сделать возможным изменение масштаба в разных окнах графика независимыми.

Для обработки входного файла в формате .csv используется библиотека pandas с набором методов класса DataFrame.

Сложности, возникшие при реализации приложения:

1. Определение элемента под курсором. Так как стандартное приложение может вернуть только координаты пикселя на всём холсте, определение выбранного элемента стало зависеть от масштаба всего окна и масштаба текущего окна, расположения курсора и других параметров.

2. Эффективность приложения. На текущий момент приложение работает относительно медленно ввиду большого объема данных, которые хранятся во время исполнения программы. Также, даже небольшое изменение положения курсора влечет перерасчет данных для вывода их пользователю, что замедляет работу программы.

4.3 Выводы

В данной главе было описано реализованное приложение для визуализации, описаны схемы взаимодействия с пользователем, его

возможности и недостатки; кратко описана программная реализация; приведены примеры построенных пиксельных графиков.

ЗАКЛЮЧЕНИЕ

В процессе реализации курсовой работы были изучены основы анализа вредоносного ПО, различные техники визуализации данных, роль визуализации в анализе вредоносных программ.

Был подробнее рассмотрен тип пиксельных графиков, их применение, достоинства и недостатки; изучен наиболее важный параметр для этого типа графиков – компоновка пикселей на графике, а также варианты решения проблемы компоновки при помощи кривых Пеано.

Реализовано вычисление координат для компоновки пикселей в зависимости от типа кривой, построение пиксельных графиков по заданным параметрам; приложение с графическим интерфейсом для визуализации набора данных с выбором параметром для неё; интерактивное взаимодействие приложения с пользователем, позволяющее получать данные об элементах по запросу. Так же приведены примеры, созданные при помощи реализованной программы и показывающие пиксельный график может быть полезным при любом из типов анализа – индивидуальном, кластеризации или анализа большого объема данных.

В дальнейшем планируется добавить еще большую интерактивность: подсветку на всех окнах графика выбранного пикселя, возможность выделять область на графике с её подсветкой; возможность выбора цветов для отображения в реальном времени. Целью также стоит повышение эффективности работы приложения и использования памяти.

СПИСОК ЛИТЕРАТУРЫ

1. M. Wagner, F. Fischer, R. Luh, A Survey of Visualization Systems for Malware Analysis, 2015

2. Daniel A. Keim, Designing Pixel-Oriented Visualization Techniques: Theory and Applications, 2000

3. Набор данных для анализа [Электронный ресурс] – 2017. – Режим доступа: https://nesg.ugr.es/nesg-ugr16/ - Дата доступа 10.12.2017

4. Набор данных для анализа [Электронный ресурс] – 2017. – Режим доступа: http://csmining.org/index.php/malicious-software-datasets- – Дата доступа 14.12.2017