**Аннотация**

Тема: Разработка программного продукта для анализа уязвимости хранения паролей в файлах.

Объем бакалаврской работы 60 страниц (40 без учета приложений), на которых размещены 4 иллюстрации. При написании работы использовались 20 источников.

Ключевые слова: информационная безопасность, информационная энтропия, пароль, криптографический ключ.

Цель работы: разработать программный продукт для обнаружения и анализа уязвимостей хранения незашифрованных паролей в файлах.

В качестве входных данных использовались открытые источники.

Результаты: создана программа с графическим интерфейсом, которая ищет текстовые файлы и анализирует в них текст на возможное содержание в нем секретных ключей. Проведено тестирование на текстах с разных содержанием и продемонстрировано, что программа успешно справляется со своей задачей.

Работа состоит из введения, основной части, состоящей из трех глав, и заключения.

В первой главе рассмотрены основные понятия и возможные способы обнаружения секретных ключей

Во второй главе рассмотрены более подробно три основных метода поиска и два дополнительных, их программная реализация и свойства, которые могут быть при помощи них обнаружены.

В третьей главе реализована и представлена готовая графическая программа, проведено тестирование и анализ результатов.

В заключении делается вывод о проделанной работе. Проведенная работа показала, что уязвимость хранения секретных ключей в незашифрованных текстовых файлах имеет большой риск обнаружения. В особенности благодаря развитию нейронных сетей, которые имеют высокую результативность.

# **СОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ** 7](#_Toc136663002)

[**Глава 1 ИССЛЕДОВАНИЕ СПОСОБОВ ИДЕНТИФИКАЦИИ ПАРОЛЕПОДОБНЫХ СТРОК И КЛЮЧЕЙ** 9](#_Toc136663003)

[**1.1 Понятие пароля** 9](#_Toc136663004)

[**1.2 Понятие криптографического ключа** 10](#_Toc136663005)

[**1.3 Теория информации** 11](#_Toc136663006)

[**1.4 Информационная энтропия** 13](#_Toc136663007)

[**1.5 Машинное обучение и нейронные сети** 14](#_Toc136663008)

[**1.6 Методы получения паролей и ключей** 18](#_Toc136663009)

[**Выводы по главе 1** 19](#_Toc136663010)

[**Глава 2 АНАЛИЗ МЕТОДОВ ПОИСКА ПАРОЛЕЙ И КЛЮЧЕЙ В ТЕКСТОВЫХ ФАЙЛАХ** 20](#_Toc136663011)

[**2.1 Применение энтропии и библиотеки Passwordmeter   
для поиска паролей и ключей** 20](#_Toc136663012)

[**2.2 Применение словаря и генератора словарей для поиска паролей** 21](#_Toc136663013)

[**2.3 Поиск по ключевым словам и регулярному выражению** 23](#_Toc136663014)

[**2.4 Применение нейронных сетей для вероятностного определения паролей** 24](#_Toc136663015)

[**Выводы по главе 2** 25](#_Toc136663016)

[**Глава 3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ДЛЯ ОПРЕДЕЛЕНИЯ ПАРОЛЕПОДОБНЫХ СТРОК В ТЕКСТОВЫХ ФАЙЛАХ** 27](#_Toc136663017)

[**3.1 Алгоритм работы и графическая реализация программы** 27](#_Toc136663018)

[**3.2 Тестирование на файлах разного содержания** 32](#_Toc136663019)

[**3.3 Анализ результатов тестирования** 35](#_Toc136663020)

[**Выводы по главе 3** 37](#_Toc136663021)

[**ЗАКЛЮЧЕНИЕ** 38](#_Toc136663022)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ** 39](#_Toc136663023)

[**ПРИЛОЖЕНИЕ А. КОД ПРОГРАММНОЙ РЕАЛИЗАЦИИ ДЛЯ ОПРЕДЕЛЕНИЯ ПАРОЛЕПОДОБНЫХ СТРОК И КЛЮЧЕЙ В ТЕКСТОВЫХ ФАЙЛАХ**  41](#_Toc136663024)

# **ВВЕДЕНИЕ**

Актуальность данной работы обусловлена растущим количеством угроз информационной безопасности, связанных с уязвимостями в процессе хранения паролей. С учётом того, что пароли являются одним из основных средств аутентификации и защиты данных, любые угрозы, связанные с их утечкой или кражей, могут иметь серьезные последствия.

Важность обеспечения безопасности хранения паролей подтверждается постоянным увеличением числа кибератак, направленных на получение доступа к личной информации и конфиденциальным данным. Одна из известных проблем безопасности является компрометация ключей доступа AWS через общедоступный репозиторий кода[*1*], [*2*], [*3*].

В этом контексте разработка программного продукта, способного анализировать и выявлять уязвимости хранения паролей в файлах, является актуальной и важной задачей.

В современном цифровом мире, где защита конфиденциальности и безопасности данных имеет важное значение, одной из наиболее распространенных и опасных уязвимостей является несанкционированный доступ к паролям, хранящимся в файлах. Уязвимость хранения паролей может привести к серьезным последствиям, включая несанкционированный доступ к аккаунтам пользователей, утечку конфиденциальных данных и нарушение информационной безопасности.

Разработка программного продукта для анализа уязвимости хранения паролей в файлах включает в себя несколько ключевых аспектов. Во-первых, необходимо провести анализ существующих методов хранения паролей и выявить их уязвимости. Это может включать исследование распространенных алгоритмов хэширования паролей, методов соления (salt) и хэширования множественного раунда.

Во-вторых, необходимо создать набор критериев и метрик для оценки безопасности хранения паролей в файлах. Эти критерии могут включать длину пароля, сложность, использование соли, применение сильных алгоритмов хэширования и другие факторы, которые влияют на безопасность паролей.

Третий этап включает разработку и реализацию ПО, которое будет осуществлять анализ уязвимости хранения паролей в файлах. Эта программа должна быть способна сканировать и проверять текстовые файлы на возможное содержание в них паролеподобных строк.

Целью данной работы является разработка программного продукта, способного анализировать уязвимости хранения незашифрованных приватных данных в файлах, для обеспечения усиления защиты данных и поддержания информационной безопасности.

Задачи работы:

1. Изучить и проанализировать существующие методы и программные продукты для анализа уязвимостей хранения паролей в файлах;
2. Изучить основные уязвимости, которые могут возникнуть при хранении паролей в файлах;
3. Разработать концепцию программного продукта, который будет выявлять уязвимости в процессе хранения паролей в файлах;
4. Разработать алгоритмы для выявления и анализа уязвимостей;
5. Разработать программное обеспечение на основе реализованного алгоритма и произвести отладку разработанного программного продукта;
6. Провести тестирование программного продукта на различных типах файлов, содержащих пароли;
7. Проанализировать полученные результаты и предложить рекомендации по улучшению безопасности хранения паролей.

# **ИССЛЕДОВАНИЕ СПОСОБОВ ИДЕНТИФИКАЦИИ ПАРОЛЕПОДОБНЫХ СТРОК И КЛЮЧЕЙ**

## **1.1 Понятие пароля**

Пароль — это секретный ключ в виде уникальной строки символов, которую пользователь использует для подтверждения своей личности или для получения доступа к ресурсу[*4*]. Пароли являются ключевым элементом в системах аутентификации и обеспечения безопасности информации. С технической точки зрения, пароль — это обычно последовательность символов, которая может включать буквы, цифры и специальные символы.

Одним из ключевых аспектов в контексте нашего исследования является **сложность пароля[***5***]**. Сложность определяется через разнообразие используемых символов и длину пароля. Чем больше длина и разнообразие символов (то есть включение строчных и прописных букв, цифр и специальных символов), тем труднее взломать пароль. Поэтому, в контексте разработки программы, которая анализирует слова и определяет, является ли слово паролем, важно учесть эти аспекты.

Говоря о сложности, стоит выделить 2 вида паролей[*5*]:

* **Пользовательский пароль** — это секретный ключ, который выбирает или создает сам пользователь для защиты своего аккаунта или доступа к системе. Это может быть любая комбинация символов, выбранная пользователем в соответствии с определенными требованиями безопасности. Пользовательский пароль обычно выбирается таким образом, чтобы быть легко запоминающимся для пользователя, но достаточно сложным для угадывания или взлома.
* **Сгенерированный (случайный) пароль** — это секретный ключ, создаваемый с использованием программных инструментов или парольных менеджеров, которые генерируют случайную последовательность символов. Этот тип пароля обычно содержит комбинацию букв (как прописные, так и строчные), цифр и специальных символов. Генерируемые пароли считаются более безопасными, так как они труднее угадать или взломать методами перебора. Однако такие пароли могут быть сложными для запоминания, поэтому рекомендуется использовать парольные менеджеры для сохранения и управления сгенерированными паролями.

Стоит также сказать, что принципиальное **различие между паролем и обычным словом** заключается в его уникальности и сложности. В отличие от обычных слов, которые часто имеют заранее определенные значения и структуру, пароли предназначены для обеспечения безопасности и должны быть сложными и трудно угадываемыми. При этом многие пользователи, в угоду удобству, используют обычные слова в качестве паролей, что снижает их безопасность.

## **1.2 Понятие криптографического ключа**

Криптографический ключ — это особый вид секретного ключа, используемый в криптографии для шифрования и дешифрования данных[*6*]. Криптографический ключ обычно является случайной последовательностью битов или символов, имеет фиксированную длину и используется в определенных криптографических алгоритмах.

Криптографические ключи обеспечивают высокий уровень безопасности для шифрования данных, поэтому они используются повсеместно и не менее ценны, чем пароли для аутентификации пользователей.

Стоит также отметить, что аутентификационные секретные ключи могут хранится не в открытом виде, а в виде хэшей – строки фиксированной длины. Хотя главная особенность хэшей то, что их нельзя вернуть в первоначальный вид, однако можно попробовать найти соответствующий пароль, применяя атаку методом перебора. Это процесс попыток применения различных паролей и проверки их хэшей на соответствие известному хэшу. А с учетом того, что хэш имеет схожие особые признаки, как у криптографического ключа, то его поиск будет иметь те же методы.

Существуют два основных типа криптографических ключей: симметричные и асимметричные[*6*][*7*].

* Симметричные ключи. В этом случае один и тот же ключ используется для шифрования и дешифрования данных. Симметричные алгоритмы, такие как AES (Advanced Encryption Standard) и DES (Data Encryption Standard), используют симметричные ключи. Один из основных аспектов использования симметричных ключей — это безопасное распространение ключей между отправителем и получателем, чтобы обеспечить конфиденциальность данных.
* Асимметричные ключи (также известные как открытые ключи), используют пару ключей: открытый и закрытый. Открытый ключ используется для шифрования данных, а соответствующий закрытый ключ - для их дешифрования. Это обеспечивает асимметричную систему шифрования, где один ключ является общедоступным, а другой - секретным. RSA (Rivest-Shamir-Adleman) – один из известных и повсеместно применяемом алгоритме, который использует асимметричные ключи. Асимметричные ключи также используются для реализации цифровых подписей, аутентификации и обмена ключами в протоколах, таких как SSL/TLS.

## **1.3 Теория информации**

Теория информации — это математическая дисциплина, которая исследует количественные аспекты информации, включая ее измерение, хранение и передачу[9].

Теория информации играет значительную роль в информационной безопасности и анализе уязвимостей хранения паролей в файлах. Она предоставляет нам не только понимание количественных аспектов информации, но и помогает оценить ее качество, сложность и безопасность. При разработке и обеспечении безопасности систем важно учитывать принципы и методы теории информации, чтобы гарантировать сохранность конфиденциальных данных и защиту от несанкционированного доступа.

В области информационной безопасности, теория информации играет важную роль в анализе и оценке уязвимостей, особенно при рассмотрении хранения паролей в файлах. Безопасное хранение паролей является критическим аспектом в обеспечении конфиденциальности данных и предотвращении несанкционированного доступа к системам.

При анализе уязвимостей хранения паролей в файлах, теория информации может быть использована для определения степени безопасности используемых методов хранения паролей. Одной из ключевых метрик является энтропия паролей, которая отражает степень неопределенности и сложности паролей[9][*10*]. Чем выше энтропия пароля, тем сложнее его подобрать методами перебора или атаки.

Оценка энтропии паролей может помочь в идентификации слабых паролей, которые могут быть легко угаданы или подобраны. Важно учитывать, что сложность паролей не зависит только от их длины, но и от использования различных символов, чисел, заглавных и строчных букв, а также случайных комбинаций. Теория информации позволяет провести анализ и оценку различных схем хранения паролей и выявить их сильные и слабые стороны.

Более безопасным подходом к хранению паролей является использование хэширования[*11*]. Хэш-функции преобразуют пароль в уникальный хэш-код, который затрудняет обратное преобразование. Однако, при использовании хэш-функций также важно учитывать их энтропию и стойкость к атакам посредством перебора или словарных атак.

При проектировании систем хранения паролей, теория информации может помочь в оценке степени безопасности выбранного метода и определении необходимости применения дополнительных мер безопасности, таких как соль (salt) и многократное хэширование (iterative hashing). Соль представляет собой случайную дополнительную информацию, которая добавляется к паролю перед хэшированием[*11*], что затрудняет использование таблиц радужных хэшей (rainbow tables) для взлома хэшей. Многократное хэширование предполагает применение хэш-функции несколько раз к паролю, увеличивая время вычисления и усложняя атаки перебором[*11*].

Одной из основных угроз при хранении паролей в файлах является возможность их несанкционированного получения третьими лицами[*12*]. В этом контексте, теория информации предоставляет инструменты для оценки степени безопасности системы хранения паролей и определения потенциальных уязвимостей. Это позволяет разработчикам и администраторам систем принимать соответствующие меры для обеспечения безопасности паролей, такие как использование сильных алгоритмов хэширования, установка ограничений на длину пароля, требования к сложности пароля, а также обновление и периодическая смена паролей.

## **1.4 Информационная энтропия**

Информационная энтропия является ключевым понятием в теории информации[9]. Она измеряет степень неопределенности или неожиданности информации. Чем больше энтропия, тем меньше информации несет сообщение или событие.

Теория информационной энтропии позволяет оценивать эффективность передачи, хранения и обработки информации, а также принимать меры для оптимизации использования ресурсов и обеспечения безопасности данных[8]. В современном информационном обществе, где обработка и передача данных являются неотъемлемой частью нашей жизни, понимание информационной энтропии играет важную роль в эффективном управлении информацией и обеспечении безопасности информационных систем.

В контексте информационной безопасности, информационная энтропия также имеет важное значение. Низкая энтропия в паролях или ключах может сделать их более уязвимыми для атак перебора или взлома. Чтобы повысить безопасность, важно использовать пароли или ключи с высокой информационной энтропией, то есть сочетание разнообразных символов, цифр и букв, а также случайных комбинаций.

Мощным инструментом для оценки стойкости пароля является информационная энтропия по Шеннону[9][*10*]. Она была предложена американским математиком Клодом Шенноном в 1948 году [8] и играет ключевую роль в измерении степени неопределенности и информационной содержательности сообщений.

По определению Шеннона, информационная энтропия  случайной дискретной переменной  с набором возможных значений  и соответствующими вероятностями  вычисляется следующим образом:

 (1.1)

где- вероятность появления значения, аобозначает сумму по всем возможным значениям переменной X.

Формула Шеннона (1.1) показывает, что энтропия измеряется в битах. Это означает, что энтропия представляет собой среднее количество бит информации, необходимых для представления каждого значения переменной X. Чем выше энтропия, тем больше неопределенность и разнообразие в значениях переменной, а, следовательно, больше информации она несет.

Когда все значения переменной X равновероятны (то есть  для всех i), энтропия Шеннона достигает своего максимального значения. В этом случае каждое значение представляет равномерное распределение информации, и энтропия будет равна, где n - количество возможных значений переменной.

Если одно из значений переменной X имеет вероятность 1 (т.е. полностью предсказуемо), то энтропия будет равна 0, так как нет неопределенности и неожиданности в данных.

## **1.5 Машинное обучение и нейронные сети**

С учетом того, что пароли имеют определенные характеристики, которые делают их отличимыми, то вполне вероятно для этой цели можно научить нейронную сеть.

Задача состоит в обучении модели распознавать и классифицировать слова или последовательности символов как потенциальные пароли. Основными проблемами здесь являются разнообразие паролей и их привязка к контексту.

Пароли могут быть выражены в виде стандартных слов или наборов символов, таких как цифры, буквы и специальные символы. Они также могут быть представлены в виде комбинаций стандартных слов, наборов символов или комбинации обоих. Таким образом, задача анализа текста на содержание в нем паролей может быть сложной.

Тем не менее нейронные сети, особенно глубокие обучающие модели могут быть особенно полезны для решения этой проблемы. Поскольку эти модели могут обучаться представлениям данных на разных уровнях абстракции, они могут быть обучены распознавать сложные шаблоны и последовательности символов, которые могут указывать на пароли.

**Recurrent Neural Networks (RNN)** или рекуррентные нейронные сети являются классом нейронных сетей, способных моделировать последовательности данных с учетом контекста и зависимостей между элементами последовательности[*13*]. RNN особенно полезны для работы с текстовыми данными, так как они способны учитывать предыдущие элементы последовательности при обработке текущего элемента.

Однако, стандартная RNN имеет ограниченную способность сохранять долгосрочные зависимости из-за проблемы исчезающего градиента или взрыва градиента. Для решения этой проблемы были разработаны модификации RNN, такие как LSTM и GRU.

**Long Short-Term Memory (LSTM)** является рекуррентной нейронной сетью, которая позволяет моделировать долгосрочные зависимости в последовательностях данных, таких как текст[*13*]. LSTM состоит из специальных блоков памяти, называемых «ячейками» которые имеют возможность сохранять информацию на протяжении длительного времени. Это свойство позволяет LSTM эффективно работать с текстовыми данными разной длины.

**Gated Recurrent Unit (GRU)** же является более простой альтернативой LSTM и обладает похожими свойствами для моделирования последовательностей данных, сохранения долгосрочных зависимостей и избегания проблемы исчезающего градиента[*13*].

Основной идеей GRU является использование шлюзов для контроля информации, передаваемой через скрытое состояние[*13*]. Она состоит из двух гейтов: обновления и сброса. Гейт обновления определяет, какую часть предыдущего скрытого состояния следует сохранить, а какую часть следует обновить с учетом текущего входа. Гейт сброса определяет, какая информация из предыдущего скрытого состояния следует игнорироваться при обработке текущего входа.

**Convolutional Neural Networks (CNN)**, с другой стороны, является архитектурой нейронной сети, основанной на применении сверток для обнаружения локальных шаблонов во входных данных[*13*]. Она обычно используется для анализа изображений, но может быть адаптирована и для работы с текстовыми данными. В контексте поиска паролей, CNN может использоваться для обнаружения характерных признаков, которые могут указывать на наличие пароля в тексте.

Чтобы выбрать какую-то конкретную ИИ, составим таблицу возможностей для каждой (Таблица 1.1)

Таблица 1.1 – Описание положительных и отрицательных черт нейросетей

|  |  |  |
| --- | --- | --- |
| Нейросеть | Положительные черты | Отрицательные черты |
| RNN | 1. Способна учитывать контекстную информацию в паролях, состоящих, например, из имени и года рождения  2. Умеет анализировать последовательности символов и тем самым может запоминать закономерности и шаблоны для паролей, состоящих из нескольких слов и чисел | 1. Из-за ограниченной памяти не будет эффективно работать с паролями сложной структуры и с локальными шаблонами в них  2. Требуется большое кол-во обучающих данных и высокие вычислительные ресурсы |
| LSTM | 1. В отличие от RNN способна сохранять и использовать информацию о долгосрочных зависимостях в паролях  2. Может работать с паролями  переменной длины 3. Способна адаптироваться к различным стилям и паттернам паролей | 1. При увеличении сложности, увеличиваются и необходимые вычислительные мощности для обучения и поиска  2. Наиболее склонна к оверфиттингу – явлению, когда ИИ не может эффективно предсказывать новые варианты, особенно при ограниченных обучающих паролей |
| GRU | 1. Облегченная версия LSTM с меньшим количеством параметров и тем самым наиболее быстрая в обучении и поиске | 1. Имеет ограничения для моделирования более сложных зависимости в отличие от LTMS  2. Хотя GRU способна учитывать долгосрочные зависимости, она может быть менее эффективной в обнаружении более сложного контекста |
| CNN | 1. Хорошо справляется с обнаружением локальных шаблонов в паролях, даже при их изменении или перестановке  2. Может успешно работать с паролями, которые немного искажены или содержат опечатки  3. Может выполнять вычисления параллельно, что делает ее эффективной для обработки больших объемов данных, как при обучении, так и при поиске  4. Наименее подвержена оверфиттингу, так как содержит меньше параметров | 1. Требуется, чтобы входные данные имели фиксированный размер  2. Ограниченный набор паролей для обучения может заметно ухудшить результативность поиска |

## **1.6 Методы получения паролей и ключей**

Проанализировав различные информацию о поиске паролей, было выделено несколько методов, которые должны дать результат.

Один из методов поиска паролей в файлах является **использование энтропии**. Для каждого пароля вычисляется его энтропия. С учетом, что пользователь заранее задает для поиска диапазон значений для энтропии, а также длины пароля, то это дает возможность для поиска паролей с характерными чертами (например, поиск ключей AWS, длина которого ровно 40 символов, а энтропия от 5.7) и для уменьшения ложных срабатываний.

Еще один распространенный метод – **перебор словарем** (Dictionary Attack). Данный способ является одним из основных методов поиска паролей в файлах и основывается на проверке каждого слова на совпадение со словами словаря. В зависимости от целей могут использоваться различные словари. Можно использовать словарь с наиболее популярными паролями, либо сгенерированные комбинации паролей на основе ранее скомпрометированных паролей, персональных данных и различных слов.

Метод перебора словарем является относительно простым и эффективным, особенно когда пароль основан на общеизвестных словах или комбинациях.

Следующий способ - **метод машинного обучения** на основе нейронных сетей для поиска паролей. В данном способе используется обученная модель нейронной сети на большом наборе данных, содержащих как правильные, так и неправильные пароли. Нейронная сеть проходит через процесс обучения, где она настраивает свои веса и параметры для определения характеристик, которые отличают правильные пароли от неправильных.

В процессе обучения нейронная сеть анализирует различные признаки и особенности паролей, такие как длина, наличие чисел, букв верхнего и нижнего регистра, специальных символов и других шаблонов. Она также может использовать контекстуальную информацию, такую как частота использования паролей, словари и известные уязвимые пароли. Такой подход позволяет автоматизировать процесс обнаружения новых вариантов паролей.

## **Выводы по главе 1**

На основе изученной информации было сделано несколько выводов:

* Секретный ключ, который под собой подразумевает пользовательский и сгенерированный пароли, а также криптографический ключ, имеет специфические внешние признаки, из-за чего их можно обнаружить в незашифрованных файлах;
* Так как пользовательские пароли чаще всего имеют небольшую длину, сложность и состоят из распространенных слов или персональной информации, то для таких случаев можно использовать поиск по заготовленному словарю паролей из открытой базы паролей взломанных компаний или сгенерированных на основе собранной персональной информации;
* Для поиска секретных ключах можно использовать информационную энтропию, которая определяет случайность последовательных символов, что очень важно для сгенерированных паролей и криптографических ключ, а также хэшей, которые могут упростить подбор для пользовательских паролей;
* Так как секретные ключи имеют повторяющиеся признаки, то с этой задачей может справиться нейронная сеть, обученная на подготовленной базе секретных ключей.

# **АНАЛИЗ МЕТОДОВ ПОИСКА ПАРОЛЕЙ И КЛЮЧЕЙ В ТЕКСТОВЫХ ФАЙЛАХ**

## **2.1 Применение энтропии и библиотеки Passwordmeter для поиска паролей и ключей**

Для данного метода мы определяем энтропию каждого слова.

Слова с высокой энтропией могут быть сильными паролями, так как они не являются простыми для угадывания или подбора. Однако практика показывает, что высокой энтропией могут обладать различные технические строки из файлов конфигураций, поэтому для увеличения точности будет добавлен инструмент Passwordmeter.

Passwordmeter - это анализатор паролей, который оценивает силу пароля на основе различных критериев, таких как[*14*][*15*][*16*]:

* **Длина пароля**: чем длиннее пароль, тем сложнее его угадать или подобрать. Поэтому алгоритм может присваивать более высокую оценку паролям с более длинными строками символов.
* **Разнообразие символов**: Использование различных типов символов в пароле, таких как буквы верхнего и нижнего регистра, цифры и специальные символы, может увеличить его сложность. Алгоритм будет оценивать наличие разнообразия символов и присваивать более высокую оценку паролям с широким спектром символов.
* **Избегание словарных слов**: Пароли, состоящие из словарных слов или известных фраз, могут быть легче подобрать с помощью словарных атак. Алгоритм может проверять, содержит ли пароль обычные слова или известные фразы и присваивать низкую оценку таким паролям.
* **Отсутствие предсказуемых паттернов**: Пароли, которые следуют предсказуемым паттернам, таким как последовательности цифр или клавиатурные шаблоны, могут быть уязвимыми. Алгоритм может проверять наличие таких паттернов и устанавливать низкую оценку паролям, которые следуют этим шаблонам.
* **Уникальность**: Использование уникальных комбинаций символов в пароле может повысить его безопасность. Алгоритм может проверять наличие повторяющихся символов или простых последовательностей и оценивать пароли с уникальными комбинациями более высоким баллом.

В реализации данной программы пользователь может сам выбрать диапазон обоих значений, однако были выставлены значения по умолчанию. Реализация представлена в Листинге А.4.

## **2.2 Применение словаря и генератора словарей для поиска паролей**

Метод поиска пользовательских паролей по словарю сравнительно прост, однако может быть не менее эффективен. Поиск по словарю — это метод атаки, в котором злоумышленник использует список базу, содержащую реальные или предполагаемые пароли.

Цель заключается в том, чтобы проверить каждое слово в словаре с целью угадать правильный пароль. Такой способ особо хорошо работает против слабых паролей, которые когда-то попали в сеть Интернет из-за кражи базы данных с паролями различных сетевых сервисов. Реализация представлена в Листинге 2.1 и *Листинге А.4*.

Листинг 2.1 – Поиск по словарю

def find\_passwords(directory, directory2, option):

if option.get() == "option2":

if directory2 == " ":

messagebox.showerror("Ошибка", "Словарь не выбран")

return " "

else:

with open(directory2, 'r') as f:

wordbook = f.read()

else:

wordbook = " "

…

if option.get() == "option2":

find\_wordbook(words, file, wordbook)

def find\_wordbook(words, file, wordbook):

words2 = wordbook.split()

for word in words:

for word2 in words2:

entropy = round(has\_password(word), 3)

if word == word2:

passwordmeters, \_ = passwordmeter.test(word)

tokenized\_word = tokenize(word)

word\_prob = np.array([tokenized\_word])

pred = model.predict(word\_prob)

inner\_list = [word, file, entropy, round(passwordmeters\*10, 3), round(float(pred[0]), 3)]

result.append(inner\_list)

Другой подход перебора по словарю заключается в создании своего словаря с помощью генератора – тип программ, которые автоматически создают пароли по некоторым правилам. Они могут использовать различные стратегии, включая комбинирование часто используемых слов, добавление чисел, специальных символов к словам, или использование конкретных шаблонов. Другой подход – это генерация всевозможных комбинаций паролей из подготовленного алфавита символов.

Генераторы словарей особенно часто применяют вкупе с социальной инженерией – социальным методам получения личной информации об атакуемой цели. Злоумышленники могут исследовать публично доступные данные, такие как социальные сети, блоги, форумы, вакансии, новости и другие источники, чтобы получить информацию о целевой аудитории.

На основе собранной информации составляют словарь паролей в виде комбинации имени и даты рождения, имен домашних животных, любимых команд и т. д. Реализация представлена *Листинге А.4*.

## **2.3 Поиск по ключевым словам и регулярному выражению**

В дополнение к поиску по энтропии/Password и словарю необходимо еще добавить поиск по ключевым словам и регулярному выражению.

Принцип первого метода основывается на использовании списка ключевых слов, которые могут указывать на наличие возможных секретных ключей или персональной информации в тексте.

Однако стоит отметить, что он может привести к большому объему ложноположительных результатов, когда ключевые слова встречаются в контексте, не связанном с ценной информацией. Реализация представлена в Листинге 2.2 и *Листинге А.4*.

Листинг 2.2 - Поиск по ключевым словам

if set\_wind.check5 == True:

# Поиск по ключевым словам с частичным совпадением

if set\_wind.check6 == True:

for word in set\_wind.array:

for i in range(len(words)):

if word in words[i]:

inner\_list = [words[i + 1] if i + 1 < len(words) else None, file, None, None, None]

result.append(inner\_list)

# Поиск по ключевым словам с полным совпадением

else:

for regex in set\_wind.array:

for i in range(len(words)):

if regex == words[i]:

inner\_list = [words[i + 1] if i + 1 < len(words) else None, file, None, None, None]

result.append(inner\_list)

Принцип последнего строиться на определенных правилах, которые заранее задаются. Допустим, нам необходимо найти слово, в котором содержится от 7 до 32 символов, хотя бы одна прописная латинская буква, хотя бы одна строчная латинская буква, хотя бы одна цифра и хотя бы один специальный символ. Такое специфическое сочетание чаще всего встречается у секретных ключей, отчего такой поиск также может быть результативен. Реализация представлена в Листинге 2.3 и *Листинге А.4*.

Листинг 2.3 - Поиск по регулярному выражению

PASSWORD\_REGEX = re.compile('^(?=.\*[0-9])(?=.\*[a-z])(?=.\*[A-Z])(?=.\*[~!@#$%^&\*\_\-+=`|\()\{\}[\]:;"\'<>,.?\/])(?=.\*\d).{7,32}$')

…

if set\_wind.check7 == True:

for x, word in enumerate(words):

if PASSWORD\_REGEX.match(word):

inner\_list = [words[x], file, None, None, None]

result.append(inner\_list)

…

## **2.4 Применение нейронных сетей для вероятностного определения паролей**

Для поиска паролей может подойти каждая из представленных в Главе 1 нейросетей, однако каждая имеет и свои особенности и без практической экспериментов сложно сказать какая будет наиболее эффективней. Поэтому исходя из представленного описания из Таблицы 1.1, я бы предпочел LSTM для данной задачи по следующим причинам:

* **Долгосрочные зависимости**: LSTM, в отличие от обычных RNN, умеют эффективно учиться от долгосрочных зависимостей. Это критически важно при обработке паролей, где важно понимать и запоминать структуру и паттерны на протяжении всей последовательности. RNN страдают от проблемы затухающего градиента, что делает их менее эффективными для долгосрочных зависимостей.
* **Работа с секретными ключами переменной длины**: LSTM, в отличие от CNN, могут обрабатывать последовательности переменной длины, что делает их гибкими для работы с паролями разной длины.
* **Моделирование сложных зависимостей**: LSTM имеют большую выразительную способность, чем GRU, из-за наличия дополнительных забывающих и входных вентилей. Это может быть полезно для моделирования более сложных зависимостей в паролях.
* **Универсальность**: LSTM могут адаптироваться к различным стилям и шаблонам паролей. Это делает их более гибкими и применимыми для широкого спектра паролей.

Для программной реализации будем использовать библиотеку TensorFlow2. Так как нейросеть не может работать напрямую со строками, то каждое слово разбивается на отдельные символы, а те в свою очередь преобразовываются в числовой код. Таким образом ИИ обучается на группе чисел и запоминается закономерность их появления. Для данной работы будет использована предобученная модель проекта DeepPass[*17*][*18*] на базе паролей “real human” с CrackStation на 2 000 000 строк длиной от 7 до 32 символов[*18*].

Принцип поиска заключается в том, что на вход обученной модели подается массив слов, который она сама умеет разбивать на символы и преобразовывать в числовой код. На выходе будет массив вероятностей, вычисленных для каждого слова. Далее происходит фильтрация – выводятся только те слова, чья вычисленная вероятность больше заданной пользователем. Реализация представлена в *Листинге А.4*.

## **Выводы по главе 2**

На основе полученных результатов было сделано несколько выводов:

* Поиск с помощью энтропии должен работать для секретных ключей с достаточно большой длиной и состоящий из как можно большего кол-ва случайных и разнообразных символов;
* Поиск на основе оценки Passwordmeter рассчитан для сложных пользовательских паролей, имеющие большую длину, уникальность и отсутствие шаблонных слов в своем составе;
* Поиск с применением словаря рассчитан на слабые и распространённые пользовательские пароли малой длины. Такие пароли может найти в общем доступе сети Интернет;
* Помимо общедоступных словарей можно еще сгенерировать самостоятельно пароли самостоятельно на генераторе словаря. При этом в зависимости от целей и условий можно создать пароли на основе комбинаций различной персональной информации атакуемой цели или на основе всевозможных вариантов паролей заданного алфавита;
* В дополнение к предыдущим методам можно использовать способы обнаружения секретных ключей на основе списка ключевых слов и регулярного выражения. Первый метод рассчитан на то, что перед ценной информации обычно находиться характерное слово, а второй на поиск слов, имеющие определенные символы в своем составе;
* Поиск с применением ИИ кажется очень перспективным, так как он учитывает гораздо больше параметров и тем самым может быть более гибким в обнаружении паролей.

# **ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ДЛЯ ОПРЕДЕЛЕНИЯ ПАРОЛЕПОДОБНЫХ СТРОК В ТЕКСТОВЫХ ФАЙЛАХ**

## **3.1 Алгоритм работы и графическая реализация программы**

Для реализации ПО был выбран язык Python, среда разработки PyCharm и библиотека Tkinter для оконного графического интерфейса.

Сначала был разработан интерфейс пользователя. Он представляет собой несколько окон, в которых были размещены различные элементы взаимодействия. Интерфейс главного окна состоит из:

* Радиокнопок для выбора одного из методов поиска секретных ключей;
* Кнопки для открытия окна настроек параметров поиска;
* Кнопки для открытия окна генератора словаря;
* Кнопки запуска сканирования;
* Кнопки сохранение результатов в файл;
* Таблицы для вывода результатов поиска с колонками: «№», «Пароль/Ключ», «Путь», «Энтропия», «Passwordmeter», «Вероятность по ИИ»;
* Текстовых полей и кнопок для выбора директории сканирования и файла словаря.

Интерфейс окна настроек состоит из:

* Полей ввода для задания диапазона поиска по значению энтропии;
* Полей ввода для задания диапазона поиска по длине секретного ключа;
* Поля ввода для задания поиска по оценке Passwordmeter от заданного значения;
* Поля ввода для задания поиска по вероятности по ИИ от заданного значения;
* Интерактивных элементов, при наведении на которые выводится информативная справка для соответствующих параметров.
* Чекбоксов для выбора форматов файлов, которые нужно анализировать;
* Чекбоксов для поиска ключевых слов с полным или частичным совпадением;
* Чекбокса для поиска по регулярному выражению
* Списка ключевых слов, поля ввода и кнопок для добавления и удаления слов;

Интерфейс окна генератора словаря состоит из:

* Области «комбинатор слов» - генератора, который создает комбинации из слов (пользователей заполняет поля ввода и вводит диапазон кол-ва слов, который можно одновременно комбинировать);
* Интерактивной кнопки, при наведении на которую выводится подсказка для области «комбинатор слов»
* Области «генератор случайных комбинаций» - генератора, который создает все комбинации паролей из алфавита символов (пользователь выбирает символы алфавита и диапазон длин паролей);
* Значения кол-ва паролей для каждой области, которое будет сгенерировано на основе заданных параметров. Значение вычисляется при изменении любых полей или чекбоксов соответствующих областей
* Кнопки сгенерировать словарь

Затем началась разработка основной логики программы. Для каждого из трех методов поиска паролей были написаны отдельные функции - по энтропии и оценке Passwordmeter, по словарю, а также с использованием нейронной сети LSTM. При запуске сканирования ПО ищет все находящиеся в директории файлы и по очереди сохраняет каждый текст, который делится на отдельные слова. Далее в зависимости от выставленных пользователем параметров поиска и выбранного метода происходит анализ слова. Если слово соответствует указанным условиям, то слово классифицируется программой как секретный ключ и сохраняет его в массив.

Метод поиска по энтропии/Passwordmeter основан на вычислении соответствующих значений и если слово входит в диапазон параметров, который заранее указал пользователь, то слово сохраняется в массив.

Метод поиска по словарю работает по простому принципу сравнения слова из текста со словами из словаря. Для работы этой функции пользователь должен заранее указать директорию словаря, который он может сам создать в окне генератора словаря или из других источников.

Метод поиска с помощью нейронной сети LSTM использует модель, которая была заранее обучена на базе паролей реальных пользователей компании RockYou. База попала в общий доступ в 2009 году после кибератаки и теперь используется для обучения нейросетей. Для более эффективной работы программы на вход ИИ поступают все слова из файла и вычисляется значение от 0 до 1, которое можно назвать значением вероятности. Далее все полученные вероятности проверяются на условии того, что оно выше или равно заданному параметру пользователем.

После выполнения сканирования, программа формирует таблицу результатов. В этой таблице для каждого найденного пароля указывается: само слово, директория файла, где был найден пароль, а также значения энтропии, оценка Passwordmeter и вероятность по ИИ.

Пользователь имеет возможность отсортировать эту таблицу по любому из числовых параметров по убыванию или возрастанию значений двойным щелчков левой кнопкой мыши. Также пользователь может скопировать данные из выбранной ячейки, кликнув по ней правой кнопкой мыши. Далее может провести новое сканирование или сохранить список в текстовый файл формата txt. Блок-схема разработанного алгоритма представлена на Рисунок 3.1

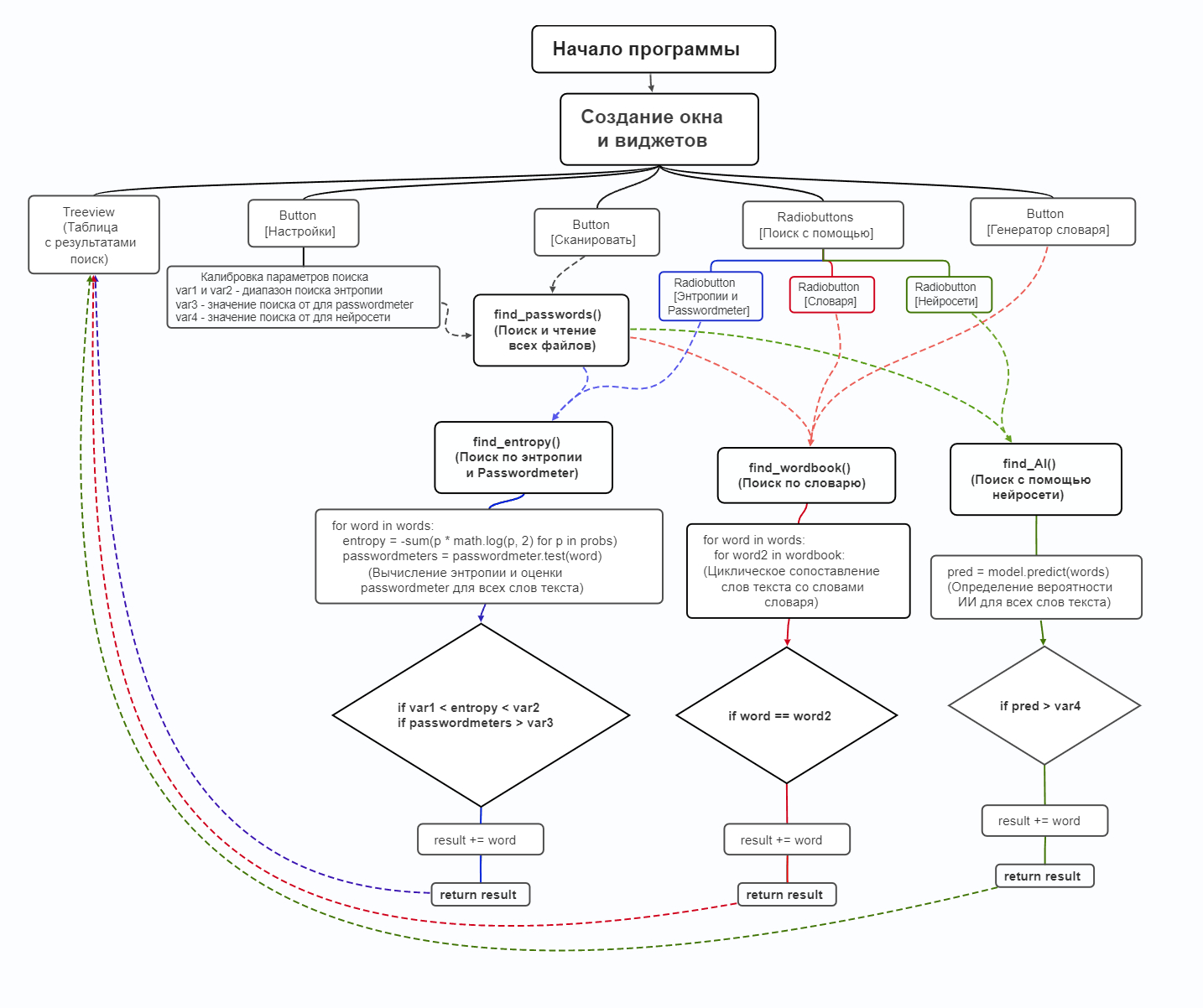


Рисунок 3.1 - Блок-схема работы алгоритма по поиску секретных ключей через оконный интерфейс

Интерфейс оконной программы представлен на Рисунке 3.2 (главное окно), Рисунок 3.3 (окно настроек) и Рисунок 3.4 (окно генератора словаря). Программная реализация представлена в *Листинге А.1* *Листинге А.2* и *Листинге А.5*

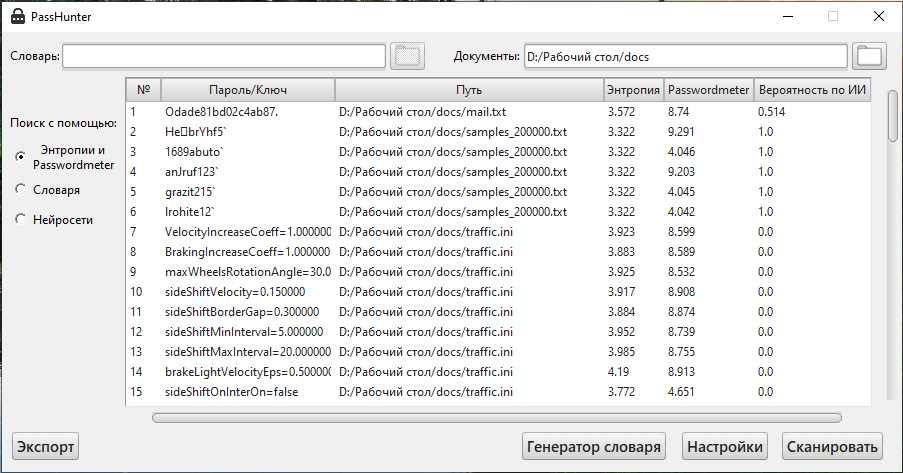


Рисунок 3.2 - Интерфейс главного экрана

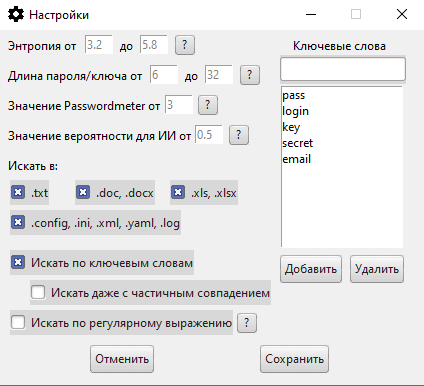


Рисунок 3.3 - Интерфейс экрана настроек

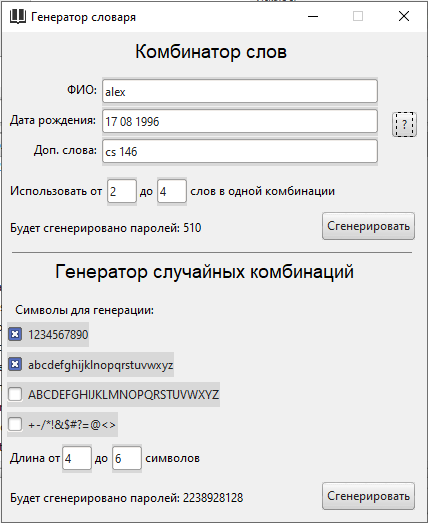


Рисунок 3.4 - Интерфейс генератора словаря

## **3.2 Тестирование на файлах разного содержания**

Для тестирования были созданы 7 файлов разного содержания   
(Таблица 3.1).

Таблица 3.1 - Количественные характеристики файлов

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Кол-во слов | Кол-во паролей | Кол-во ключей |
| Англоязычная статья | 2 845 | 6 | 3 |
| Русскоязычной статья[[1]](#footnote-1) | 1 140 | 4 | 6 |
| Электронное письмо | 71 | 0 | 3 |
| Файл конфигурации | 449 | 0 | 0 |
| Список реальных криптографических ключей | 14 | 0 | 14 |
| Список реальных паролей из базы компании RockYou[*19*] | 59 187 | 59 187 | 0 |
| Список популярных паролей[*20*] | 100 000 | 100 000 | 0 |

Для поиска были заданы такие параметры:

* Энтропия – от 3.7 до 5.8;
* Passwordmeter – от 6 (max 10);
* Длина – от 6 до 512;
* Вероятность по ИИ – от 0.5 (max 1);
* Поиск по ключевым словам и регулярному выражению отключен.

Выбор данных значений обусловлен тем, чтобы сократить число ложных срабатываний, но искать секретные ключи самой разной длины. Результаты пробного тестирования показали, что у английских слов[[2]](#footnote-2) значения энтропии может достигать до 3.5 - 3.7, Passwordmeter до 4.7-8.8, вероятности по ИИ до 4.94. Средняя минимальная длина паролей составляет от 6 знаков.

Тестирование осуществлялось только двумя методами – по энтропии/Passwordmeter и с помощью нейросети. Метод поиска по словарю не применялся, так как результаты напрямую зависят от применяемых словарей. Методы поиска по ключевым словам и регулярному выражению также не имеет смысла тестировать. Формат файлов не влиял на результаты работы, поэтому он не будет указываться.

Результаты работы программы представлены в Таблица 3.2 и Таблица 3.3. В них представлены абсолютные значения и процент от их общего числа из Таблица 3.1

Таблица 3.2 - Результаты для метода поиска по энтропии/Passwordmeter

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Тип | Кол-во ложных срабатываний | % | Кол-во найденных паролей | % | Кол-во найденных ключей | % |
| Англоязычная статья | 1 | **0.03** | 3 | **50** | 1 | **33** |
| Русскоязычная статья | 0 | **0** | 0 | **0** | 4 | **66** |
| Электронное письмо | 0 | **0** | - | **-** | 1 | **33** |
| Файл конфигурации | 246 | **54.8** | - | **-** | - | **-** |
| Список реальных криптографических ключей[[3]](#footnote-3) | - | **-** | - | **-** | 3 / 13 | **21 / 93** |
| Список реальных паролей из базы компании RockYou | - | **-** | 0 | **0** | - | **-** |
| Список популярных паролей | - | **-** | 0 | **0** | - | **-** |

Таблица 3.3 - Результаты поиска для метода поиска с помощью нейросети

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Тип | Кол-во ложных срабатываний | % | Кол-во найденных паролей | % | Кол-во найденных ключей | % |
| Англоязычная статья | 30 | **1** | 0 | **0** | 1 | **33** |
| Русскоязычная статья | 318 | **27.9** | 2 | **50** | 0 | **0** |
| Электронное письмо | 1 | **1.4** | - | **-** | 2 | **66** |
| Файл конфигурации | 2 | **0.5** | - | **-** | - | **-** |
| Список реальных криптографических ключей | - | **-** | - | **-** | 6 | **43** |
| Список реальных паролей из базы компании RockYou | - | **-** | 29 858 | **50** | - | **-** |
| Список популярных паролей | - | **-** | 49 085 | **49** | - | **-** |

## **3.3 Анализ результатов тестирования**

Исходя из вышеприведенных результатов, можно заметить, что результаты доказывают изначальное положение, что метод по энтропии/Passwordmeter справляется в основном находит секретные ключи со случайно сгенерированными символами. Поиск с помощью нейросети заметно лучше справился с паролями.

Однако были замечены некоторые особенности. Для метода по энтропии/Passwordmeter:

* Большинство криптографических ключей имеют значение Passwordmeter не более 5, из-за чего их можно пропустить при высоком параметре Passwordmeter
* Поиск дал высокие ложные срабатывания для файла конфигурации, так как в нем содержались длинные строки длиной 30-50 символов, которые состояли из нескольких слов, знака равно и чисел, что и приводило к высоким значениям энтропии и Passwordmeter;
* Поиск не дал ни одного результата для файлов с простыми пользовательскими паролями. Это связано с тем, что в среднем длина паролей составляет не более 6-12 символов и не очень сложные, в связи с чем энтропия в среднем для них не превышала 3.4, а оценка Passwordmeter не более 2.7;
* В списке были пароли, чья энтропия составляла больше 3.7, а оценка Passwordmeter больше 7, однако эти условия не соблюдались одновременно, в связи с чем они также не были обнаружены;
* Ложные срабатывания при сканировании статьи оказался почти нулевым, при том, что он смог найти некоторое кол-во секретных ключей.

Для метода поиска с помощью нейросети были такие особенности:

* В англоязычной статье нейросеть не нашла ни одного пароля в то время, как первым методом было найдено 3 (50%);
* Нейросеть дала очень много ложных срабатываний для русскоязычной статьи, так как изначально обучалась на паролях из латинских букв;
* При поиске криптографических ключей ИИ могла найти те ключи, которые не были найдены методом энтропии/Passwordmeter. Это связано с тем, что эти ключи были длиной всего 16-24 знака;
* Нейросеть показала достаточно высокие результаты на реальных пользовательских паролях, как на базе паролей компании RockYou, на которой обучалась ИИ, так и на другой базе. Это можно объяснить тем, что и там, и там содержалось много похожих паролей, что тем не менее не умаляет результат.

## **Выводы по главе 3**

На основе проведенного тестирования и экспериментов было сделано несколько выводов:

* Метод энтропии/Passwordmeter эффективен для поиска секретных ключей со случайно сгенерированными символами. Однако оптимальная работа наблюдается при минимальных значениях параметра оценки Passwordmeter, тем самым результат поиск в файле с ключами был увеличен с 1 до 13 ключей из 14;
* Метод энтропии/Passwordmeter может дать высокие ложные срабатывания при анализе файлов конфигурации и ему подобных, содержащих длинные строки специфического формата, что показал результат в 55% ложных результатов;
* Метод энтропии/Passwordmeter не справляется с короткими и несложными паролями, имеющими низкую энтропию и низкий рейтинг Passwordmeter, что показал нулевой результат при поиске в обоих файлах с паролями;
* Высокий результат работы нейросети на пользовательских паролях показал (~50%), что данный метод имеет высокий потенциал и его результаты будут тем лучше, чем больше будет база паролей на входе обучения и чем более разнообразнее она будет, охватывая как можно больше различных вариантов;
* Нейросеть оказалась неэффективной при анализе текстов на языках, отличных от языка, на котором она была обучена, что было видно при 28% ложных срабатываний.
* Эксперимент доказал, что ни один из методов не является универсальным решением, и их совместное использование может повысить эффективность поиска.

# **ЗАКЛЮЧЕНИЕ**

Проблема хранения паролей и криптографических ключей в незашифрованном виде в текстовых файлах остается актуальной и по сей день. Этот вопрос является одним из вопросов проблематики информационной безопасности.

С распространением искусственного интеллекта, проблема становится еще более актуальной. Используя нейронные сети, злоумышленники могут автоматизировать анализа данных, что увеличивает вероятность обнаружения и

секретных ключей.

В ходе работы было проведено изучение и тестирование трех методов анализа уязвимости хранения паролей в файлах: поиск по энтропии и оценке Passwordmeter, по словарю, с использованием нейронной сети архитектуры LSTM (которая способна запоминать долгосрочные и сложные зависимости), а также, по ключевым словам и регулярному выражению. Каждый из этих методов демонстрирует свои уникальные преимущества и недостатки, и их применение определяется поставленными целями и условиями поиска.

Практический эксперимент показал, что поиск с анализом слов по энтропии дал более 90% положительного результата на криптографических ключах при абсолютных значениях 14 строк. А поиск нейронной сетью показал ~50% для двух файлов с разными базами пользовательских паролей при абсолютных значениях 59 тыс. и 100 тыс. строк соответственно.

Таким образом каждый метод может дополнять друг друга и учитывать различные подходы поиска для паролей и криптографических ключей и разные условия поиска.

Стоит подчеркнуть, что тема данной работы относится к узкому и специфическому направлению работ. Она направлена на выявление уязвимых мест, где могут храниться пароли и криптографические ключи в незашифрованном виде.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Exploring 25K AWS S3 buckets // Medium // URL: <https://medium.com/securing/exploring-25k-aws-s3-buckets-f22ec87c3f2a> (дата обращения: 07.05.2023)

2. AWS Access Keys Leak in GitHub Repository and Some Improvements in Amazon Reaction // URL: <https://medium.com/swlh/aws-access-keys-leak-in-github-repository-and-some-improvements-in-amazon-reaction-cc2e20e89003> (дата обращения: 07.05.2023)

3. About secret scanning // GitHub // URL: <https://docs.github.com/en/code-security/secret-scanning/about-secret-scanning> (дата обращения: 07.05.2023)

4. В.В. Белов, В.И. Чистякова. Алгоритмы и структуры данных. Учебник, Инфра-М: 2019, 240 с.

5. Python глазами хакера, BHV: 2022, 176 с.

6. Клевов А.: Шифраторы и радиоразведка. Щит и меч информационного мира. Том 1. Записки криптографа, Центр инновационных технологий: 2019, 960 с.

7. Введение в криптографию и шифрование // Хабр // URL: <https://habr.com/ru/companies/yandex/articles/327636/> (дата обращения: 03.05.2023)

8. Ричард Хэмминг: Глава 13. Теория информации // Хабр // URL: <https://habr.com/ru/articles/422205/> (дата обращения: 03.05.2023)

9. Shannon Entropy, Information Gain, and Picking Balls from Buckets // Medium // URL: <https://medium.com/udacity/shannon-entropy-information-gain-and-picking-balls-from-buckets-5810d35d54b4> *(дата обращения: 05.05.2023)*

10. Как найти все пароли и ключи в большом количестве файлов // HackWare // URL: <https://hackware.ru/?p=10602> (дата обращения: 03.05.2023)

11. Информационные технологии в управлении техническими системами и моделирование, эл. сборник докладов: Белгород: БГТУ им. В.Г. Шухова, 2022., 225 с.

12. Berlin A.: Defensive Security Handbook: Best Practices for Securing Infrastructure, O'reilly, Incorporated: 2019, 284 c.

13. Гагендара Т.: Обработка естественного языка с TensorFlow,   
ДМК Пресс: 2020, 382 с.

14. Passwordmeter // GitHub // URL: <https://github.com/cadithealth/passwordmeter> (дата обращения: 07.05.2023)

15. DumpsterDiver // GitHub // URL: <https://github.com/securing/DumpsterDiver> (дата обращения: 07.05.2023)

16. Hunting for secrets with the DumpsterDiver // Medium // URL: <https://rzepsky.medium.com/hunting-for-secrets-with-the-dumpsterdiver-93d38a9cd4c1> (дата обращения: 07.05.2023)

17. DeepPass // GitHub // URL: <https://github.com/GhostPack/DeepPass> (дата обращения: 13.05.2023)

18. DeepPass — Finding Passwords With Deep Learning // Medium // URL: <https://posts.specterops.io/deeppass-finding-passwords-with-deep-learning-4d31c534cd00> (дата обращения: 13.05.2023)

19. rockyou-75.txt // GitHub // URL: <https://github.com/berandal666/Passwords/blob/master/rockyou-75.txt> (дата обращения: 26.05.2023)

20. 10\_million\_password\_list\_top\_100000.txt // GitHub // URL: <https://github.com/berandal666/Passwords/blob/master/10_million_password_list_top_100000.txt> (дата обращения: 26.05.2023)

# **ПРИЛОЖЕНИЕ А. КОД ПРОГРАММНОЙ РЕАЛИЗАЦИИ ДЛЯ ОПРЕДЕЛЕНИЯ ПАРОЛЕПОДОБНЫХ СТРОК И КЛЮЧЕЙ В ТЕКСТОВЫХ ФАЙЛАХ**

Листинг А.0.1 – Интерфейс главного окна

from tkinter import ttk  
import tkinter as tk  
from tkinter import \*  
from wordbook import open\_wordbok\_window  
from set\_wind import open\_settings\_window  
from func import get\_directory, on\_header\_click, scan, export\_to\_txt, copy\_to\_clipboard, f\_menu, state, get\_directory2  
from ttkthemes import ThemedTk  
from PIL import Image, ImageTk  
  
root = ThemedTk(theme='scidblue') # создание окна  
root.title("PassHunter") # установка заголовка окна  
root.iconbitmap('images/icon.ico') # изменение иконки окна  
# получение размеров экрана  
screen\_width = root.winfo\_screenwidth()  
screen\_height = root.winfo\_screenheight()  
# расчет положения окна по центру экрана  
x\_position = (screen\_width - 900) // 2 # ширина окна  
y\_position = (screen\_height - 440) // 2 # высота окна  
# установка размеров и положения окна  
root.geometry(f"900x440+{x\_position}+{y\_position}")  
# запрет изменения размеров окна  
root.resizable(False, False)  
  
# -------------------------------------------  
# Адрес директории поиска  
label = Label(root, text="Документы:")  
label.place(x=449, y=13)  
directory\_entry = ttk.Entry(root, width=53)  
directory\_entry.place(x=522, y=10)  
# Кнопка выбора директории  
image = ImageTk.PhotoImage(Image.open("images/folder.png"))  
address\_button = ttk.Button(root, image=image, command=lambda: get\_directory(directory\_entry))  
address\_button.place(x=850, y=10)  
  
# Радиокнопка  
label = Label(root, text="Поиск с помощью:")  
label.place(x=5, y=80)  
selected\_option = tk.StringVar(value="option1")  
selected\_option.trace("w", lambda \*args: state(selected\_option, address\_button2))  
option1 = tk.Radiobutton(root, text="Энтропии и\nPasswordmeter", variable=selected\_option, value="option1")  
option1.place(x=8, y=105)  
option2 = tk.Radiobutton(root, text="Словаря", variable=selected\_option, value="option2")  
option2.place(x=8, y=145)  
option3 = tk.Radiobutton(root, text="Нейросети", variable=selected\_option, value="option3")  
option3.place(x=8, y=175)  
  
# Адрес словаря  
label2 = Label(root, text="Словарь:")  
label2.place(x=5, y=13)  
directory\_entry2 = ttk.Entry(root, width=53)  
directory\_entry2.place(x=60, y=10)  
# Кнопка выбора директории  
address\_button2 = ttk.Button(root, image=image, command=lambda: get\_directory2(directory\_entry2))  
address\_button2.place(x=388, y=10)  
state(selected\_option, address\_button2)  
  
# Ползунки  
scrollbar = ttk.Scrollbar(root)  
scrollbar2 = ttk.Scrollbar(root, orient="horizontal")  
# Таблица  
tree = ttk.Treeview(root, show="headings", columns=("num", "passw", "path", "entropy", "Passwordmeter","Probability"),  
 yscrollcommand=scrollbar.set, xscrollcommand=scrollbar2.set)  
scrollbar.config(command=tree.yview)  
scrollbar2.config(command=tree.xview)  
tree.bind('<Double-Button-1>', lambda event: on\_header\_click(event, tree)) # Сортировка двойным кликом ЛКМ  
tree.heading("#1", text="№")  
tree.heading("#2", text="Пароль/Ключ")  
tree.heading("#3", text="Путь")  
tree.heading("#4", text="Энтропия")  
tree.heading("#5", text="Passwordmeter")  
tree.heading("#6", text="Вероятность по ИИ")  
tree.column("#0", width=0, stretch=NO)  
tree.column("#1", width=35, stretch=NO)  
tree.column("#2", width=140, stretch=NO)  
tree.column("#3", width=300, stretch=NO)  
tree.column("#4", width=60, stretch=NO)  
tree.column("#5", width=90, stretch=NO)  
tree.column("#6", width=120)  
tree.place(x=123, y=45, height=330) # Измените высоту таблицы здесь  
scrollbar.place(x=885, y=58, height=310) # Измените высоту полосы прокрутки здесь  
scrollbar2.place(x=150, y=380, width=690) # Измените положение полосы прокрутки горизонтальной прокрутки здесь  
  
# Создание контекстного меню для копирования  
menu = tk.Menu(root, tearoff=0)  
menu.add\_command(label="Скопировать", command=lambda: copy\_to\_clipboard(tree, root))  
 # Вызов контекстного меню при нажатии на правую кнопку мыши  
tree.bind("<Button-3>", lambda event: f\_menu(event, menu))  
  
# Кнопка генератора словаря  
button = ttk.Button(root, text="Генератор словаря", command=lambda: open\_wordbok\_window(root), style='Button2.TButton')  
button.place(x=520, y=400)  
  
# Кнопка настроек  
button = ttk.Button(root, text="Настройки", command=lambda: open\_settings\_window(root), style='Button2.TButton')  
button.place(x=680, y=400)  
  
# Кнопка для запуска сканирования  
style = ttk.Style().configure('Button2.TButton', font=('Segoe UI Semibold', 11)) # Установка размера шрифта  
ttk\_button = ttk.Button(root, text="Сканировать", command=lambda: scan(tree, selected\_option), style='Button2.TButton')  
ttk\_button.place(x=780, y=400)  
  
# Кнопка для экспорта данных  
ttk\_button = ttk.Button(root, text="Экспорт", command=lambda: export\_to\_txt(tree), style='Button2.TButton')  
ttk\_button.place(x=10, y=400)  
  
root.mainloop() # Запуск главного цикла программы

Листинг А.0.2 - Интерфейс окна настроек

import tkinter as tk  
from tkinter import ttk  
from tkinter import \*  
from tkinter import messagebox  
  
# Значения по умолчанию  
input\_text1 = 3.2  
input\_text2 = 5.8  
input\_text3 = 6  
input\_text4 = 32  
input\_text5 = 3  
input\_text6 = 0.5  
array = ["pass", "login", "key", "secret", "email"]  
check1 = True  
check2 = True  
check3 = True  
check4 = True  
check5 = True  
check6 = False  
check7 = False  
  
def open\_settings\_window(root):  
 global input\_text1, input\_text2, input\_text3, input\_text4, input\_text5, input\_text6, array  
 global check1, check2, check3, check4, check5, check6, check7  
 # Настройки окна настроек  
 settings = tk.Toplevel(root)  
 settings.title("Настройки")  
 settings.iconbitmap('images/settings.ico')  
 screen\_width = settings.winfo\_screenwidth()  
 screen\_height = settings.winfo\_screenheight()  
 x = (screen\_width - 425) // 2  
 y = (screen\_height - 355) // 2  
 settings.geometry(f"425x355+{x}+{y}")  
 settings.focus\_set() # Установка фокуса на окно  
 settings.lift() # Перемещение окна поверх других окон  
 settings.grab\_set() # Установка окна как модального  
 settings.resizable(False, False)  
  
 # Фукнции для работы тектовой подсказки в полях ввода  
 def on\_entry\_click(event, input\_entry, placeholder\_text):  
 if input\_entry.get() == placeholder\_text:  
 input\_entry.delete(0, tk.END)  
 input\_entry.config(fg='black')  
 def on\_focus\_out(event, input\_entry, placeholder\_text):  
 if input\_entry.get() == '':  
 input\_entry.insert(0, placeholder\_text)  
 input\_entry.config(fg='gray')  
  
 # Энтропия от и до  
 set\_label1 = Label(settings, text="Энтропия от")  
 set\_label1.place(x=5, y=5)  
 set\_entry1 = tk.Entry(settings, width=4, fg='gray')  
 placeholder1 = str(input\_text1)  
 set\_entry1.insert(0, placeholder1)  
 set\_entry1.bind("<FocusIn>", lambda event: on\_entry\_click(event, set\_entry1, placeholder1))  
 set\_entry1.bind("<FocusOut>", lambda event: on\_focus\_out(event, set\_entry1, placeholder1))  
 set\_entry1.place(x=85, y=5)  
  
 set\_label2 = Label(settings, text="до")  
 set\_label2.place(x=117, y=5)  
 set\_entry2 = tk.Entry(settings, width=4, fg='gray')  
 placeholder2 = str(input\_text2)  
 set\_entry2.insert(0, placeholder2)  
 set\_entry2.bind("<FocusIn>", lambda event: on\_entry\_click(event, set\_entry2, placeholder2))  
 set\_entry2.bind("<FocusOut>", lambda event: on\_focus\_out(event, set\_entry2, placeholder2))  
 set\_entry2.place(x=140, y=5)  
  
 # Подсказка 1  
 help1 = ttk.Button(settings, text="?")  
 help1.place(x=175, y=5, width=20, height=20)  
 help1.bind("<Enter>", lambda event: show\_tooltip(200, 80, event,  
 "По умолчанию - от 3.2 до 5.8\n Для сгенерированных паролей\nи ключей рекомендуется - от 3.9"))  
 help1.bind("<Leave>", lambda event: hide\_tooltip(event))  
  
 # Длина пароля/ключа от и до  
 set\_label3 = Label(settings, text="Длина пароля/ключа от")  
 set\_label3.place(x=5, y=35)  
 set\_entry3 = tk.Entry(settings, width=4, fg='gray')  
 placeholder3 = str(input\_text3)  
 set\_entry3.insert(0, placeholder3)  
 set\_entry3.bind("<FocusIn>", lambda event: on\_entry\_click(event, set\_entry3, placeholder3))  
 set\_entry3.bind("<FocusOut>", lambda event: on\_focus\_out(event, set\_entry3, placeholder3))  
 set\_entry3.place(x=150, y=35)  
  
 set\_label4 = Label(settings, text="до")  
 set\_label4.place(x=182, y=35)  
 set\_entry4 = tk.Entry(settings, width=4, fg='gray')  
 placeholder4 = str(input\_text4)  
 set\_entry4.insert(0, placeholder4)  
 set\_entry4.bind("<FocusIn>", lambda event: on\_entry\_click(event, set\_entry4, placeholder4))  
 set\_entry4.bind("<FocusOut>", lambda event: on\_focus\_out(event, set\_entry4, placeholder4))  
 set\_entry4.place(x=205, y=35)  
  
 # Подсказка 2  
 help2 = ttk.Button(settings, text="?")  
 help2.place(x=240, y=35, width=20, height=20)  
 help2.bind("<Enter>", lambda event: show\_tooltip(185, 80, event,  
 " По умолчанию - от 6 до 32\n Для ключей и хэшей\n рекомендуется - от 16 до 512"))  
 help2.bind("<Leave>", lambda event: hide\_tooltip(event))  
  
 # Значение Passwordmeter от  
 set\_label5 = Label(settings, text="Значение Passwordmeter от")  
 set\_label5.place(x=5, y=65)  
 set\_entry5 = tk.Entry(settings, width=4, fg='gray')  
 placeholder5 = str(input\_text5)  
 set\_entry5.insert(0, placeholder5)  
 set\_entry5.bind("<FocusIn>", lambda event: on\_entry\_click(event, set\_entry5, placeholder5))  
 set\_entry5.bind("<FocusOut>", lambda event: on\_focus\_out(event, set\_entry5, placeholder5))  
 set\_entry5.place(x=165, y=65)  
  
 # Подсказка 3  
 help3 = ttk.Button(settings, text="?")  
 help3.place(x=198, y=65, width=20, height=20)  
 help3.bind("<Enter>", lambda event: show\_tooltip(185, 80, event,  
 " По умолчанию - от 3\n Для сложных паролей\n рекомендуется - от 7 (max 10)"))  
 help3.bind("<Leave>", lambda event: hide\_tooltip(event))  
  
 # Значение вероятности для ИИ  
 set\_label6 = Label(settings, text="Значение вероятности для ИИ от")  
 set\_label6.place(x=5, y=95)  
 set\_entry6 = tk.Entry(settings, width=4, fg='gray')  
 placeholder6 = str(input\_text6)  
 set\_entry6.insert(0, placeholder6)  
 set\_entry6.bind("<FocusIn>", lambda event: on\_entry\_click(event, set\_entry6, placeholder6))  
 set\_entry6.bind("<FocusOut>", lambda event: on\_focus\_out(event, set\_entry6, placeholder6))  
 set\_entry6.place(x=195, y=95)  
  
 # Подсказка 4  
 help4 = ttk.Button(settings, text="?")  
 help4.place(x=229, y=95, width=20, height=20)  
 help4.bind("<Enter>", lambda event: show\_tooltip(200, 120, event,  
 " По умолчанию - от 0.5\n Значение задает вероятность,\nот которой ИИ будет определять\n"  
 " слово как пароль\n\n Рекомендуется не менять"))  
 help4.bind("<Leave>", lambda event: hide\_tooltip(event))  
  
 # Чекбоксы  
 label = Label(settings, text="Искать в:")  
 label.place(x=5, y=125)  
 # Чекбокс "txt"  
 check\_var1 = BooleanVar()  
 check\_var1.set(check1)  
 check\_button = ttk.Checkbutton(settings, text=".txt", variable=check\_var1, style="Custom.TCheckbutton")  
 check\_button.place(x=10, y=150)  
 # Чекбокс "docx, doc"  
 check\_var2 = BooleanVar()  
 check\_var2.set(check2)  
 check\_button = ttk.Checkbutton(settings, text=".doc, .docx", variable=check\_var2, style="Custom.TCheckbutton")  
 check\_button.place(x=75, y=150)  
 # Чекбокс "xlsx', xls"  
 check\_var3 = BooleanVar()  
 check\_var3.set(check3)  
 check\_button = ttk.Checkbutton(settings, text=".xls, .xlsx", variable=check\_var3, style="Custom.TCheckbutton")  
 check\_button.place(x=170, y=150)  
 # Чекбокс "config, ini, xml, yaml, log"  
 check\_var4 = BooleanVar()  
 check\_var4.set(check4)  
 check\_button = ttk.Checkbutton(settings, text=".config, .ini, .xml, .yaml, .log", variable=check\_var4, style="Custom.TCheckbutton")  
 check\_button.place(x=10, y=180)  
 # Чекбокс - искать по ключевым словам  
 check\_var5 = BooleanVar()  
 check\_var5.set(check5)  
 check\_button = ttk.Checkbutton(settings, text="Искать по ключевым словам", variable=check\_var5, style="Custom.TCheckbutton")  
 check\_button.place(x=10, y=220)  
 # Чекбокс - искать по ключевым словам c частичным или полным совпадением  
 check\_var6 = BooleanVar()  
 check\_var6.set(check6)  
 check\_button = ttk.Checkbutton(settings, text="Искать даже с частичным совпадением", variable=check\_var6, style="Custom.TCheckbutton")  
 check\_button.place(x=30, y=250)  
 # Чекбокс - искать регулярному выражению  
 check\_var7 = BooleanVar()  
 check\_var7.set(check7)  
 check\_button = ttk.Checkbutton(settings, text="Искать по регулярному выражению", variable=check\_var7, style="Custom.TCheckbutton")  
 check\_button.place(x=10, y=280)  
 help5 = ttk.Button(settings, text="?")  
 help5.place(x=237, y=283, width=20, height=20)  
 help5.bind("<Enter>", lambda event: show\_tooltip(225, 120, event,  
 "Поиск слов, в которых присутствует\nобязательно хотя бы одна цифра,\nхотя бы одна латинская буква\n(в нижнем или верхнем регистре),\nхотя бы один специальный символ\nи длина строки от 7 до 32 символов"))  
 help5.bind("<Leave>", lambda event: hide\_tooltip(event))  
  
 # Кнопка сохранить и отменить  
 button = ttk.Button(settings, text="Сохранить", command=lambda: save())  
 button.place(x=260, y=315)  
 button = ttk.Button(settings, text="Отменить", command=lambda: settings.destroy())  
 button.place(x=90, y=315)  
  
 # Добавить слово в список ключевых слов  
 def add\_word():  
 words = word\_entry.get().split()  
 if words:  
 for word in words:  
 word\_list.insert(tk.END, word) # добавляем слово в конец списка  
 array.append(word)  
 word\_entry.delete(0, tk.END) # очищаем поле ввода  
  
 # Удалить слово в список ключевых слов  
 def delete\_selected\_word():  
 # получаем индекс выбранного элемента  
 selected\_indices = word\_list.curselection()  
 # удаляем выбранный элемент, если он есть  
 for i in selected\_indices[::-1]:  
 selected\_word = word\_list.get(i)  
 word\_list.delete(i)  
 if selected\_word in array:  
 array.remove(selected\_word)  
 if word\_list.size() > i:  
 word\_list.selection\_set(i)  
 elif word\_list.size() > 0:  
 word\_list.selection\_set(word\_list.size() - 1)  
  
 # Список ключевых слов  
 label = Label(settings, text="Ключевые слова")  
 label.place(x=290, y=5)  
 word\_entry = ttk.Entry(settings)  
 word\_entry.place(x=280, y=25)  
 word\_list = tk.Listbox(settings)  
 word\_list.place(x=280, y=55)  
 for word in array:  
 word\_list.insert(tk.END, word)  
 add\_button = ttk.Button(settings, text="Добавить", command=add\_word)  
 add\_button.place(x=280, y=225)  
 delete\_button = ttk.Button(settings, text="Удалить", command=delete\_selected\_word)  
 delete\_button.place(x=350, y=225)  
  
 def save():  
 global input\_text1, input\_text2, input\_text3, input\_text4, input\_text5, input\_text6  
 global check1, check2, check3, check4, check5, check6, check7  
 end = True  
 if float(set\_entry1.get()) > 0 and float(set\_entry2.get()) > 0 and int(set\_entry3.get()) > 0 \  
 and int(set\_entry4.get()) > 0 and int(set\_entry5.get()) > 0 and float(set\_entry6.get()) > 0:  
 input\_text1 = set\_entry1.get()  
 input\_text2 = set\_entry2.get()  
 input\_text3 = set\_entry3.get()  
 input\_text4 = set\_entry4.get()  
 else:  
 messagebox.showerror("Ошибка", "Значения не могут быть меньше 0")  
 end = False  
 if float(set\_entry5.get()) <= 10:  
 input\_text5 = set\_entry5.get()  
 else:  
 messagebox.showerror("Ошибка", "Значение passwordmeter не может быть\nменьше 0 или больше 10")  
 end = False  
 if float(set\_entry6.get()) <= 1:  
 input\_text6 = float(set\_entry6.get())  
 else:  
 messagebox.showerror("Ошибка", "Значение вероятности не может быть\nменьше 0 или больше 1")  
 end = False  
  
 check1 = check\_var1.get()  
 check2 = check\_var2.get()  
 check3 = check\_var3.get()  
 check4 = check\_var4.get()  
 check5 = check\_var5.get()  
 check6 = check\_var6.get()  
 check7 = check\_var7.get()  
 if end:  
 settings.destroy()  
  
 def show\_tooltip(c\_x, c\_y, event, text):  
 global tooltip  
 tooltip = tk.Toplevel(root)  
 ttk.Style().configure("Tooltip.TLabel", background="white")  
 tooltip.wm\_overrideredirect(True) # Убираем рамку окна  
 tooltip.geometry(f"{c\_x}x{c\_y}+{x}+{y}")  
 tooltip.wm\_geometry(f"+{event.x\_root+10}+{event.y\_root-40}") # Позиционируем окно подсказки  
 tooltip.configure(bg='white')  
  
 label = ttk.Label(tooltip, text="Справка", style="Tooltip.TLabel")  
 label.place(x=72, y=5)  
 label = ttk.Label(tooltip, text=text, style="Tooltip.TLabel")  
 label.place(x=10, y=25)  
  
 def hide\_tooltip(event):  
 tooltip.destroy()

Листинг А.0.3 - Реализация различных функций

from tkinter import \*  
from scan import find\_passwords  
from tkinter import filedialog  
from tkinter import messagebox  
import tkinter as tk  
import os  
import openpyxl  
  
# Выбор дикректории сканирования  
directory = "C:/"  
def get\_directory(directory\_entry):  
 global directory  
 directory = filedialog.askdirectory()  
 directory\_entry.delete(0, END) # очищаем строку ввода  
 directory\_entry.insert(0, directory) # вставляем адрес выбранной директории  
  
directory2 = " "  
def get\_directory2(directory\_entry2):  
 global directory2  
 directory2 = filedialog.askopenfilename(filetypes=[("Text Files", "\*.txt")])  
 directory\_entry2.delete(0, END) # очищаем строку ввода  
 directory\_entry2.insert(0, directory2) # вставляем адрес выбранной директории  
  
# Запуск сканирования  
def scan(tree, option):  
 if directory == "C:/":  
 messagebox.showerror("Ошибка", "Директория не выбрана")  
 else:  
 tree.delete(\*tree.get\_children())  
 result = find\_passwords(directory, directory2, option)  
 if result != " ":  
 for i, (password, path, entropy, passwordmeter, probability) in enumerate(result, 1):  
 tree.insert("", "end", text=str(i), values=(i, password, path, entropy, passwordmeter, probability))  
  
# Сортировка столбцов по возрастанию и убыванию  
reverse=False  
def on\_header\_click(event, tree):  
 column = tree.identify\_column(event.x) # Определение столбца, по которому было произведено нажатие  
 global reverse  
 reverse = not reverse  
 data = [(tree.set(child, column), child) for child in tree.get\_children('')]  
 data.sort(key=lambda x: num\_key(x[0]), reverse=reverse) # Сортировка данных по указанному столбцу  
 for index, (\_, child) in enumerate(data):  
 tree.move(child, '', index) # Перемещение элементов в таблице в новый порядок  
  
# Перевод строки в число или 0 при ошибке  
def num\_key(num\_str):  
 try:  
 return float(num\_str)  
 except ValueError:  
 return 0  
  
# Экспорт данных в файл  
def export\_to\_txt(tree):  
 filename = filedialog.asksaveasfilename(defaultextension=".txt", filetypes=  
 [("Text Files", "\*.txt"),("Excel Files", "\*.xlsx")])  
 if not filename:  
 return # Если пользователь не указал имя файла, прерываем операцию  
 else:  
 \_, extension = os.path.splitext(filename)  
 if extension == ".txt":  
 with open(filename, "w") as file:  
 for child in tree.get\_children():  
 values = tree.item(child)["values"]  
 line = "\t".join(str(value) for value in values) # Соединяем значения через табуляцию  
 file.write(line + "\n") # Записываем строку в файл  
 else:  
 # Создание нового Excel-файла  
 workbook = openpyxl.Workbook()  
 sheet = workbook.active  
 # Получение заголовков столбцов из Treeview  
 columns = tree['columns']  
 for i, column in enumerate(columns):  
 column\_heading = tree.heading(column)['text']  
 sheet.cell(row=1, column=i + 1, value=column\_heading)  
 # Запись данных из Treeview в Excel  
 for row\_id in tree.get\_children():  
 row = tree.item(row\_id)['values']  
 for i, value in enumerate(row):  
 sheet.cell(row=str(row\_id+2), column=str(i + 1), value=value)  
 # Сохранение файла Excel  
 workbook.save(filename)  
  
 messagebox.showinfo("Экспорт завершен", "Данные успешно сохранены.")  
  
# Копирование значения выделенной ячейки  
def copy\_to\_clipboard(tree, root):  
 try:  
 row\_id = tree.selection()[0] # получаем ID выделенной строки  
 col\_index = tree.identify\_column(coord\_x) # Определение столбца, по которому было произведено нажатие  
 col\_id = int(col\_index.replace('#', ''))  
 value = tree.item(row\_id)['values'][col\_id-1] # получаем значение ячейки  
 root.clipboard\_clear() # очищаем буфер обмена  
 root.clipboard\_append(value) # добавляем значение в буфер обмена  
 except IndexError:  
 print("Ячейка не выбрана")  
  
# Создание контекстного меню  
def f\_menu(event, menu):  
 global coord\_x  
 coord\_x = event.x  
 menu.post(event.x\_root, event.y\_root)  
  
def state(selected\_option, option2\_button):  
 if selected\_option.get() != "option2":  
 option2\_button.config(state="disabled")  
 else:  
 option2\_button.config(state="normal")

Листинг А.0.4 - Функции, реализующие все методы сканирования

import os  
import math  
from collections import Counter  
import set\_wind  
from docx import Document  
from openpyxl import load\_workbook  
from tkinter import messagebox  
import passwordmeter  
import tensorflow as tf  
import numpy as np  
import re  
model = tf.keras.models.load\_model('./model-lstm')  
  
# извлечено из токенизатора fit Keras, поэтому нам не нужен Keras / Tensorflow в качестве требования  
CHAR\_DICT = {'<UNK>': 1, 'e': 2, 'i': 3, 'a': 4, 'n': 5, 't': 6, 'r': 7, 'o': 8, 's': 9, 'c': 10, 'l': 11, 'A': 12,  
 'E': 13, 'd': 14, 'u': 15, 'm': 16, 'p': 17, 'I': 18, 'S': 19, 'R': 20, 'O': 21, 'N': 22, 'g': 23, 'T': 24,  
 '-': 25, 'L': 26, 'h': 27, 'y': 28, 'C': 29, 'b': 30, 'f': 31, 'M': 32, 'v': 33, 'D': 34, '1': 35, 'U': 36,  
 'H': 37, 'P': 38, 'k': 39, '2': 40, '0': 41, 'B': 42, 'G': 43, 'w': 44, 'Y': 45, 'K': 46, '3': 47, '9': 48,  
 'F': 49, '.': 50, ',': 51, '4': 52, '8': 53, 'V': 54, '5': 55, '7': 56, '6': 57, 'W': 58, 'j': 59, 'x': 60,  
 'z': 61, 'J': 62, 'q': 63, 'Z': 64, '\_': 65, "'": 66, ':': 67, 'X': 68, 'Q': 69, '/': 70, ')': 71, '(': 72,  
 '"': 73, '!': 74, ';': 75, '\*': 76, '@': 77, '\\': 78, ']': 79, '?': 80, '[': 81, '<': 82, '>': 83,  
 '=': 84, '#': 85, '&': 86, '$': 87, '+': 88, '%': 89, '`': 90, '~': 91, '^': 92, '{': 93, '}': 94, '|': 95}  
  
# регулярное выражение для слов, в которых присутствует хотя бы одна цифра,  
# хотя бы одна латинская буква (в нижнем или верхнем регистре)  
# хотя бы один специальный символ и что длина строки от 7 до 32 символов  
PASSWORD\_REGEX = re.compile('^(?=.\*[0-9])(?=.\*[a-z])(?=.\*[A-Z])(?=.\*[~!@#$%^&\*\_\-+=`|\()\{\}[\]:;"\'<>,.?\/])(?=.\*\d).{7,32}$')  
  
# Поиск всех файлов в директории и во всех подпапках  
def get\_files\_recursive(directory):  
 file\_list = []  
 for root, dirs, files in os.walk(directory):  
 for file in files:  
 file\_path = os.path.join(root, file)  
 file\_path = file\_path.replace("\\", "/") # Замена обратных слэшей на прямые слэши  
 file\_list.append(file\_path)  
 return file\_list  
def find\_passwords(directory, directory2, option):  
 global result  
  
 files = get\_files\_recursive(directory)  
 result = []  
 if option.get() == "option2":  
 if directory2 == " ":  
 messagebox.showerror("Ошибка", "Словарь не выбран")  
 return " "  
 else:  
 with open(directory2, 'r') as f:  
 wordbook = f.read()  
 else:  
 wordbook = " "  
  
 # Проход по каждому файлу и проверка его содержимого на наличие пароля  
 for file in files:  
 content = ""  
 if file.endswith(('.txt')):  
 if set\_wind.check1:  
 with open(os.path.join(directory, file), 'r') as f:  
 content = f.read()  
  
 elif file.endswith(('.docx', '.doc')):  
 if set\_wind.check2:  
 doc = Document(os.path.join(directory, file))  
 paragraphs = [p.text for p in doc.paragraphs]  
 content = "\n".join(paragraphs)  
  
 elif file.endswith(('.xlsx', '.xls')):  
 if set\_wind.check3:  
 wb = load\_workbook(os.path.join(directory, file))  
 sheet = wb.active  
 rows = sheet.iter\_rows(values\_only=True)  
 for row in rows:  
 content += " ".join(str(cell) for cell in row) + "\n"  
  
 elif file.endswith(('.config', '.ini', '.xml', '.yaml', '.log')):  
 if set\_wind.check4:  
 with open(os.path.join(directory, file), 'r') as f:  
 content = f.read()  
  
 words = content.split() # Разделение текста на слова  
  
 if option.get() == "option1":  
 find\_entropy(words, file)  
 elif option.get() == "option2":  
 find\_wordbook(words, file, wordbook)  
 elif option.get() == "option3":  
 find\_AI(words, file)  
 # print(len(result))  
 return result  
  
# Вычисление энтропии текста  
def has\_password(text):  
 freq = Counter(text)  
 probs = [count / len(text) for count in freq.values()]  
 entropy = - sum(p \* math.log(p, 2) for p in probs)  
 return entropy  
def find\_entropy(words, file):  
 # Поиск по регулярному выражению  
 if set\_wind.check7 == True:  
 for x, word in enumerate(words):  
 if PASSWORD\_REGEX.match(word):  
 inner\_list = [words[x], file, None, None, None]  
 result.append(inner\_list)  
  
 if set\_wind.check5 == True:  
 # Поиск по ключевым словам с частичным совпадением  
 if set\_wind.check6 == True:  
 for word in set\_wind.array:  
 for i in range(len(words)):  
 if word in words[i]:  
 inner\_list = [words[i + 1] if i + 1 < len(words) else None, file, None, None, None]  
 result.append(inner\_list)  
 # Поиск по ключевым словам с полным совпадением  
 else:  
 for regex in set\_wind.array:  
 for i in range(len(words)):  
 if regex == words[i]:  
 inner\_list = [words[i + 1] if i + 1 < len(words) else None, file, None, None, None]  
 result.append(inner\_list)  
  
 for word in words:  
 if float(set\_wind.input\_text3) <= len(word) <= float(set\_wind.input\_text4):  
 passwordmeters, \_ = passwordmeter.test(word)  
 if passwordmeters\*10 >= float(set\_wind.input\_text5):  
 entropy = round(has\_password(word), 3)  
 # Проверка наличия пароля по заданному порогу энтропии  
 if float(set\_wind.input\_text1) < entropy < float(set\_wind.input\_text2):  
 tokenized\_word = tokenize(word)  
 word\_prob = np.array([tokenized\_word])  
 pred = model.predict(word\_prob)  
 inner\_list = [word, file, entropy, round(passwordmeters\*10, 3), round(float(pred[0]), 3)]  
 result.append(inner\_list)  
  
def find\_wordbook(words, file, wordbook):  
 if set\_wind.check7 == True:  
 for x, word in enumerate(words):  
 if PASSWORD\_REGEX.match(word):  
 inner\_list = [words[x], file, None, None, None]  
 result.append(inner\_list)  
  
 if set\_wind.check5 == True:  
 # Ищем по ключевым словам с частичным совпадением  
 if set\_wind.check6 == True:  
 for word in set\_wind.array:  
 for i in range(len(words)):  
 if word in words[i]:  
 inner\_list = [words[i + 1] if i + 1 < len(words) else None, file, None, None, None]  
 result.append(inner\_list)  
 # Ищем по ключевым словам с полным совпадением  
 else:  
 for word in set\_wind.array:  
 for i in range(len(words)):  
 if word == words[i]:  
 inner\_list = [words[i + 1] if i + 1 < len(words) else None, file, None, None, None]  
 result.append(inner\_list)  
  
 words2 = wordbook.split()  
 for word in words:  
 for word2 in words2:  
 entropy = round(has\_password(word), 3)  
 if word == word2:  
 passwordmeters, \_ = passwordmeter.test(word)  
 tokenized\_word = tokenize(word)  
 word\_prob = np.array([tokenized\_word])  
 pred = model.predict(word\_prob)  
 inner\_list = [word, file, entropy, round(passwordmeters\*10, 3), round(float(pred[0]), 3)]  
 result.append(inner\_list)  
  
# Поиск через нейросеть  
def find\_AI(text, file):  
 # Преобразуем вводимые слова в последовательности и увеличиваем их до 32  
 tokenized\_words = [tokenize(word) for word in text]  
 words = np.array(tokenized\_words)  
 # Размещаем маркированные слова в обслуживаемой модели  
 pred = model.predict(words)  
  
 # Определение вероятности, что слово является паролем  
 pred\_bool = (pred > set\_wind.input\_text6)  
 # Используйте предсказания в качестве индексов для возвращаемых слов  
 positive\_indicies = np.where(pred\_bool == True)  
 for x in positive\_indicies[0]:  
 password = text[x]  
 entropy = round(has\_password(password), 3)  
 passwordmeters, \_ = passwordmeter.test(password)  
 inner\_list = [password, file, entropy, round(passwordmeters \* 10, 3), round(float(pred[x]), 3)]  
 result.append(inner\_list)  
  
  
def tokenize(word): # Выделяет и дополняет введенное слово до нужной длины.  
 global CHAR\_DICT  
  
 if len(word) < 7 or len(word) > 32:  
 return [0] \* 32  
 else:  
 seq = [CHAR\_DICT[char] if char in CHAR\_DICT else 1 for char in word]  
 seq.extend([0] \* (32 - len(seq)))  
 return seq

Листинг А.0.5 - Окно генератора словаря

import itertools  
import tkinter as tk  
from tkinter import ttk  
from tkinter import \*  
  
check1 = True  
check2 = True  
check3 = False  
check4 = False  
  
def open\_wordbok\_window(root):  
 global check1, check2, check3, check4  
 wordbok = tk.Toplevel(root)  
 wordbok.title("Генератор словаря")  
 wordbok.iconbitmap('images/wordbook.ico')  
 screen\_width = wordbok.winfo\_screenwidth()  
 screen\_height = wordbok.winfo\_screenheight()  
 x = (screen\_width - 425) // 2  
 y = (screen\_height - 550) // 2  
 wordbok.geometry(f"425x490+{x}+{y}")  
 wordbok.focus\_set() # Установка фокуса на окно  
 wordbok.lift() # Перемещение окна поверх других окон  
 wordbok.grab\_set() # Установка окна как модального  
 wordbok.resizable(False, False)  
  
 # Комбинатор слов  
 label = Label(wordbok, text="Комбинатор слов", font=("Arial", 14))  
 label.place(x=130, y=5)  
 label1 = Label(wordbok, text="ФИО:")  
 label1.place(x=62, y=47)  
 ent1 = tk.StringVar()  
 entry1 = ttk.Entry(wordbok, width=45, textvariable=ent1)  
 entry1.place(x=100, y=45)  
 label2 = Label(wordbok, text="Дата рождения:")  
 label2.place(x=5, y=77)  
 ent2 = tk.StringVar()  
 entry2 = ttk.Entry(wordbok, width=45, textvariable=ent2)  
 entry2.place(x=100, y=75)  
 label3 = Label(wordbok, text="Доп. слова:")  
 label3.place(x=29, y=107)  
 ent3 = tk.StringVar()  
 entry3 = ttk.Entry(wordbok, width=45, textvariable=ent3)  
 entry3.place(x=100, y=105)  
 label4 = Label(wordbok, text="Использовать от")  
 label4.place(x=5, y=148)  
 ent4 = tk.StringVar()  
 entry4 = ttk.Entry(wordbok, width=4, textvariable=ent4)  
 entry4.place(x=105, y=145)  
 label5 = Label(wordbok, text="до")  
 label5.place(x=135, y=148)  
 ent5 = tk.StringVar()  
 entry5 = ttk.Entry(wordbok, width=4, textvariable=ent5)  
 entry5.place(x=155, y=145)  
  
 label6 = Label(wordbok, text="слов в одной комбинации")  
 label6.place(x=185, y=148)  
 # Подсказка 1  
 help1 = ttk.Button(wordbok, text="?")  
 help1.place(x=390, y=80, width=25, height=25)  
 help1.bind("<Enter>", lambda event: show\_tooltip(205, 70, event, "Слова разделяются через пробел\n "  
 "и не должны повторяться"))  
 help1.bind("<Leave>", lambda event: hide\_tooltip(event))  
 # Кнопка сгенирировать  
 button = ttk.Button(wordbok, text="Сгенерировать", command=lambda: combinator())  
 button.place(x=320, y=180)  
 # Разграничительная линия  
 h\_line = tk.Frame(wordbok, height=1, bg="gray", width=400)  
 h\_line.place(x=10, y=220)  
  
  
 # Генератор случайных комбинаций  
 label7 = Label(wordbok, text="Генератор случайных комбинаций", font=("Arial", 14))  
 label7.place(x=50, y=225)  
 # Чекбоксы  
 label8 = Label(wordbok, text="Символы для генерации:")  
 label8.place(x=10, y=267)  
 # Чекбокс цифр  
 check\_var1 = BooleanVar()  
 check\_var1.set(check1)  
 check\_button = ttk.Checkbutton(wordbok, text="1234567890", variable=check\_var1, style="Custom.TCheckbutton")  
 check\_button.place(x=5, y=290)  
 # Чекбокс строчных латинских букв  
 check\_var2 = BooleanVar()  
 check\_var2.set(check2)  
 check\_button = ttk.Checkbutton(wordbok, text="abcdefghijklnopqrstuvwxyz", variable=check\_var2, style="Custom.TCheckbutton")  
 check\_button.place(x=5, y=320)  
 # Чекбокс прописных латинских букв  
 check\_var3 = BooleanVar()  
 check\_var3.set(check3)  
 check\_button = ttk.Checkbutton(wordbok, text="ABCDEFGHIJKLMNOPQRSTUVWXYZ", variable=check\_var3, style="Custom.TCheckbutton")  
 check\_button.place(x=5, y=350)  
 # Чекбокс спец. символов  
 check\_var4 = BooleanVar()  
 check\_var4.set(check4)  
 check\_button = ttk.Checkbutton(wordbok, text="+-/\*!&$#?=@<>", variable=check\_var4, style="Custom.TCheckbutton")  
 check\_button.place(x=5, y=380)  
  
 # Длина пароля  
 label9 = Label(wordbok, text="Длина от")  
 label9.place(x=5, y=415)  
 ent6 = tk.StringVar()  
 entry6 = ttk.Entry(wordbok, width=4, textvariable=ent6)  
 entry6.place(x=60, y=412)  
 label10 = Label(wordbok, text="до")  
 label10.place(x=90, y=415)  
 ent7 = tk.StringVar()  
 entry7 = ttk.Entry(wordbok, width=4, textvariable=ent7)  
 entry7.place(x=110, y=412)  
 label11 = Label(wordbok, text="символов")  
 label11.place(x=140, y=415)  
  
 # Кол-во паролей, которое будет сгенерировано  
 def update\_label1(\*args):  
 label\_var1.set("Будет сгенерировано паролей: " + str(m()))  
 def update\_label2(\*args):  
 label\_var2.set("Будет сгенерировано паролей: " + str(n()))  
  
 check\_var1.trace("w", update\_label2)  
 check\_var2.trace("w", update\_label2)  
 check\_var3.trace("w", update\_label2)  
 check\_var4.trace("w", update\_label2)  
 ent1.trace("w", update\_label1)  
 ent2.trace("w", update\_label1)  
 ent3.trace("w", update\_label1)  
 ent4.trace("w", update\_label1)  
 ent5.trace("w", update\_label1)  
 ent6.trace("w", update\_label2)  
 ent7.trace("w", update\_label2)  
  
 label\_var1 = tk.StringVar()  
 label12 = Label(wordbok, textvariable=label\_var1)  
 label12.place(x=5, y=185)  
 label\_var2 = tk.StringVar()  
 label13 = Label(wordbok, textvariable=label\_var2)  
 label13.place(x=5, y=455)  
  
 # Кнопка сгенирировать  
 button = ttk.Button(wordbok, text="Сгенерировать", command=lambda: generator())  
 button.place(x=320, y=450)  
  
 # Комбинатор  
 def combinator():  
 words = []  
 words.extend(entry1.get().split())  
 words.extend(entry2.get().split())  
 words.extend(entry3.get().split())  
  
 result\_dict = {}  
  
 for i in range(int(ent4.get()), int(ent5.get())+1):  
 for subset in itertools.combinations(words, i):  
 for perm in itertools.permutations(subset):  
 result\_dict[''.join(perm)] = True  
  
  
 with open("wordbook.txt", "w") as f:  
 for result in result\_dict.keys():  
 f.write(result + '\n')  
 # Генератор  
 def generator():

len\_alf = ""

len\_alf += "1234567890" if check\_var1.get() else ""

len\_alf += "abcdefghijklmnopqrstuvwxyz" if check\_var2.get() else ""

len\_alf += "ABCDEFGHIJKLMNOPQRSTUVWXYZ" if check\_var3.get() else ""

len\_alf += "!@#$%^&\*+=-?~" if check\_var4.get() else ""

result\_dict = {}

if ent6.get() != '' and ent7.get() != '' and int(ent6.get()) <= int(ent7.get()):

for i in range(int(ent6.get()), int(ent7.get()) + 1):

for subset in itertools.combinations(len\_alf, i):

for perm in itertools.permutations(subset):

result\_dict[''.join(perm)] = True

with open("wordbook.txt", "w") as f:

for result in result\_dict.keys():

f.write(result + '\n')  
  
  
 def show\_tooltip(c\_x, c\_y, event, text):  
 global tooltip  
 tooltip = tk.Toplevel(root)  
 ttk.Style().configure("Tooltip.TLabel", background="white")  
 tooltip.wm\_overrideredirect(True) # Убираем рамку окна  
 tooltip.geometry(f"{c\_x}x{c\_y}+{x}+{y}")  
 tooltip.wm\_geometry(f"+{event.x\_root + 10}+{event.y\_root - 40}") # Позиционируем окно подсказки  
 tooltip.configure(bg='white')  
  
 label = ttk.Label(tooltip, text="Справка", style="Tooltip.TLabel")  
 label.place(x=72, y=5)  
 label = ttk.Label(tooltip, text=text, style="Tooltip.TLabel")  
 label.place(x=10, y=25)  
  
  
 def hide\_tooltip(event):  
 tooltip.destroy()  
  
 def n():  
 sum\_pass = 0  
 len\_alf = 0  
 len\_alf += 10 if check\_var1.get() else 0  
 len\_alf += 26 if check\_var2.get() else 0  
 len\_alf += 26 if check\_var3.get() else 0  
 len\_alf += 13 if check\_var4.get() else 0  
 if ent6.get() != '' and ent7.get() != '' and int(ent6.get()) <= int(ent7.get()):  
 for i in range(int(ent6.get()), int(ent7.get())+1):  
 sum\_pass += len\_alf \*\* i  
 return sum\_pass  
 return 0  
  
 def m():  
 words = []  
 words.extend(entry1.get().split())  
 words.extend(entry2.get().split())  
 words.extend(entry3.get().split())  
 sum\_pass = 0  
 if ent4.get() != '' and ent5.get() != '':  
 if ent1.get() != '' or ent2.get() != '' or ent3.get() != '':  
 if len(words) >= int(ent5.get()) and int(ent4.get()) <= int(ent5.get()):  
 for i in range(int(ent4.get()), int(ent5.get()) + 1):  
 n = len(words)  
 r = i  
 Fn = 1  
 Fnr = 1  
  
 for j in range(1, n + 1):  
 Fn \*= j  
 for j in range(1, n - r + 1):  
 Fnr \*= j  
  
 sum\_pass += Fn / Fnr  
  
 return int(sum\_pass)

1. Пароли были в том числе на кириллице [↑](#footnote-ref-1)
2. Так как программа разделяет текст на слова по пробелам, то они анализируются в том числе со спец. знаками и числами. Так как слова могут иметь запятые, кавычки, дефис с цифрами и т.к., то это увеличивает значения энтропии и Passwordmeter, что дает больше ложных срабатываний [↑](#footnote-ref-2)
3. При первоначально выставленных настройках результат вышел слишком низкий и было решено изменить параметр оценки Passwordmeter от 1 для данного случая [↑](#footnote-ref-3)