МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

СОЗДАНИЕ СРЕДЫ ДЛЯ ИЗОЛИРОВАННОГО ВЫПОЛНЕНИЯ НЕДОВЕРЕННЫХ ПРОГРАММ В ОС WINDOWS

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Кузнецова Александра Вадимовича

Научный руководитель
Зав.каф., к. ф.-м. н., доцент _____ С. В. Миронов

Заведующий кафедрой
к. ф.-м. н., доцент _____ С. В. Миронов

Саратов 2021

ВВЕДЕНИЕ

В настоящее время наблюдается активный рост количества вредоносных компьютерных программ: вирусов, троянов, червей, вымогателей и других видов. Традиционный способ борьбы с ними — использование сигнатурного анализа — зачастую оказывается неэффективным, поскольку огромное количество ежедневно появляющихся образцов компьютерных вирусов препятствует их своевременному добавлению в базы данных антивирусного ПО. Так, каждый день выявляется порядка 560000 вредоносных программ. Выявлять, анализировать и включать в базы сигнатур такое количество образцов вручную практически невозможно.

Для противодействия сложившейся ситуации используются следующие подходы:

1. **Эвристический анализ**. Вредоносные программы выявляются путём анализа их исполняемого кода и детектирования «аномалий»: участков, не свойственных добросовестным исполняемым файлам, таких как самомодифицирующийся код, использование редких системных вызовов, необычный формат структур исполняемого файла (PE-заголовок, таблица импортов) и прочее.
2. **Поведенческий анализ**. Специальный монитор постоянно анализирует поведение запущенных на компьютере программ и выявляет действия потенциально вредоносного ПО: сканирование портов, изменение системных файлов, перехват нажатий клавиш, модификация памяти других процессов.
3. **Машинное обучение**. Это относительно новый и активно набирающий популярность способ выявления вредоносных программ. Его суть состоит в создании модели-классификатора и обучении его задаче разделения класса программ на опасные и безопасные.
4. **Изолированные среды (песочницы)**. Программы-песочницы позволяют выполнять недоверенный код изолированно: так, чтобы он не имел доступа к реальному окружению пользователя. В результате, даже если недоверенный исполняемый файл окажется вредоносным, он не сможет нанести вред.
5. **Виртуальные машины**. Позволяют запускать приложения в отдельной *гостевой* (виртуальной) операционной системе, исключая большинство

рисков для основной. В контексте защиты от вредоносных программ используются либо для их исследования, либо в ситуациях, когда необходима максимально высокая степень изоляции недоверенного кода от основной ОС. В отличие от песочниц, виртуальные машины не предоставляют выполняемым программам никакого доступа к основной ОС, за исключением малочисленных и строго контролируемых методов: например, виртуальная машина `VirtualBox` позволяет создавать так называемые «общие папки» для обмена файлами между основной и гостевой системами.

6. **Гибридные методы**, сочетающие несколько предыдущих. Так, антивирус `Avast` использует комбинацию песочницы, эвристического и поведенческого анализов для оценки безопасности «редких» — ранее не встречавшихся — исполняемых файлов.

Серьёзную угрозу безопасности создают так называемые криптовымогатели. Эти вредоносные программы шифруют файлы пользователя и требуют от него выкуп за их расшифровку. Криптовымогатели представляют угрозу как обычным пользователям, так и организациям. Выкуп, требуемый злоумышленниками, может варьироваться в широких пределах: от нескольких тысяч рублей до десятков миллионов долларов. У криптовымогателей есть ряд особенностей, значительно усложняющих защиту от подобного вида вредоносного ПО:

— Для работы им не требуется высоких прав. Запуска исполняемого файла с обычными, не административными, правами достаточно, чтобы вымогатель зашифровал все файлы, которые может модифицировать текущий пользователь: его документы, фотографии, данные браузера и прочее.
— Вымогателю не нужно закрепляться в операционной системе. В отличие от многих других видов вредоносного ПО (троянов, кейлоггеров), ему достаточно быть запущенным только один раз.
— Зачастую вымогателям не требуется доступ в интернет. Применение асимметричной криптографии позволяет им не передавать на сервера злоумышленника симметричные ключи, используемые для шифрования файлов.
— Поведение вымогателя во время шифрования обычно довольно простое и сложно отличимо от действий обычных программ.

— Доступность и развитие надёжных криптографических примитивов позволяют злоумышленникам реализовывать устойчивую криптографию, которую фактически невозможно взломать на текущем уровне развития технологий. Единственный реалистичный способ восстановить файлы после атаки качественного шифровальщика — это получение приватных ключей, которые могут храниться на серверах или компьютерах злоумышленников.
— Некоторые виды криптовымогателей производят атаку при помощи доверенных приложений, обладающих цифровой подписью (например, GPG), и предназначенных для легального шифрования файлов.

Эти особенности приводят к тому, что распознать деятельность криптовымогателей при помощи эвристического и поведенческого анализов становится крайне сложно, а сигнатурный анализ слишком сильно «запаздывает»: к моменту, когда сигнатура очередного криптовымогателя попадёт в базу данных антивирусного ПО, он успеет нанести существенный ущерб.

Количество атак криптовымогателей и ущерб от них постоянно растут. Актуальной становится проблема защиты файлов от несанкционированного шифрования. Наиболее эффективной мерой превентивной борьбы с подобным классом вредоносных программ являются песочницы и виртуальные машины: криптовымогатель, запущенный в изолированной среде, запрещающей доступ к файлам пользователя, не сможет нанести существенного ущерба.

Целью дипломной работы является разработка легковесной изолированной среды (песочницы) для защиты пользовательских файлов от несанкционированной модификации недоверенными программами.

Для достижения поставленной цели были выполнены следующие задачи:

— Обзор и анализ существующих решений.
— Обзор способов создания изолированных сред.
— Постановка требований к создаваемому приложению.
— Исследование предметной области.
— Разработка приложения в соответствии с поставленными требованиями и проведёнными исследованиями.

Актуальность данной работы определяется возросшим количеством вредоносных приложений, повреждающих пользовательские файлы, отсутствием надёжных средств их предварительного обнаружения, сложностью и неэффек-

тивностью существующих решений.

Работа состоит из введения, трёх глав, заключения, списка использованных источников и четырёх приложений.

1 КРАТКОЕ СОДЕРЖАНИЕ

В первой главе работы произведена постановка задачи, выведены функциональные и нефункциональные требования, проанализированы существующие решения, выбраны средства разработки.

В ходе работы были выведены следующие функциональные требования к разрабатываемому приложению:

— **Изолирование доступа к файлам**. Недоверенный исполняемый файл (*НИФ*), исполняющийся внутри песочницы, не должен иметь никакого доступа (на чтение, изменение, выполнение или перечисление) к файлам, не требующимся для работы этого исполняемого файла. Это необходимо для обеспечения защиты пользовательских файлов от несанкционированных модификации или чтения.

— **Прозрачность файловой системы**. НИФ должен иметь возможность читать, модифицировать, создавать или исполнять файлы в строго заданных директориях, расположенных на реальной файловой системе пользователя. Это обеспечивает возможность бесшовного переключения на выполнение НИФ без песочницы: например, когда доброкачественность НИФ была подтверждена.

— **Гибкость**. Для каждого НИФ необходимо иметь возможность задать гибкие разрешения в виде списков путей, к которым у НИФ есть доступ. Должны поддерживаться как белые, так и чёрные списки. Разрешения должны задаваться четвёркой «чтение, запись, исполнение, перечисление». Первые три разрешения относятся к файлам, последнее — к директориям. Поскольку большинство программ хранят свои данные в различных директориях, необходимо организовать настраиваемую возможность доступа к ним.

— **Рекурсивность**. Все новые процессы, порождаемые во время работы НИФ, также должны выполняться внутри песочницы. Так, если НИФ во время работы запускает интерпретатор какого-либо скриптового языка, этот интерпретатор тоже не должен иметь доступа к пользовательским файлам.

Дополнительно был выведен ряд нефункциональных требований:

— **Поддержка ОС Windows**.

— **Легковесность**. Песочница не должна ощутимо влиять на скорость за-

пуска, производительность и отзывчивость исполняемых в ней НИФ.

— **Минимальность прав**. Процесс песочницы должен выполняться с минимально необходимыми правами: такими же, какие требуются для запуска конкретного НИФ. Это упрощает её использование и улучшает безопасность.
— **Исполнение в режиме пользователя**. Исходя из предыдущих пунктов, песочница должна выполняться в режиме пользователя, не ниже третьего кольца защиты.

В ходе анализа существующих решений были выделены и проанализированы приложения *Sandboxie*, *Avast Sandbox*, *Windows Sandbox*. Основными общими недостатками этих решений в контексте работы являются необходимость установки ядерных модулей (драйверов) и требование высокого уровня привилегий.

В качестве средств разработки были выбраны:

— Язык **C++** для разработки основной функциональности.
— Язык **C#** для разработки графического интерфейса пользователя.
— Язык ассемблера **FASM** для разработки кода внедрения (шеллкода).
— Интегрированная среда разработки (IDE) **Microsoft Visual Studio 2019 Community**.

Также для разработки проекта был использован ряд сторонних библиотек:

— Библиотека логгирования **plog**, распространяющаяся по открытой лицензии Mozilla Public License 2.0.
— Библиотека для работы с JSON **rapidjson**, распространяющаяся по открытой лицензии MIT.
— Библиотека создания хуков **minhook**, распространяющаяся по открытой лицензии BSD 2-Clause.
— Встроенная СУБД **SQLite**.

Во второй главе проанализированы и классифицированы способы изоляции файловой системы, исследованы строение и возможности динамически загружаемых библиотек (DLL) в операционной системе Windows, выделены и проанализированы наиболее удобные способы внедрения библиотек, изучен механизм работы 32-битных приложений в 64-битных версиях ОС Windows, проанализирована техника «Heaven’s Gate», позволяющая выполнять код с раз-

рядностью, отличающейся от разрядности внедряющего процесса, проанализированы принципы работы системных вызовов Windows, разработана общая архитектура создаваемого приложения.

Разделяемые библиотеки — это бинарные файлы, состоящие из кода часто используемых процедур. Разделяемые библиотеки могут быть *загружены* произвольным количеством независимых программ и использоваться независимо. В ОС Windows разделяемые библиотеки распространяются в формате «DLL» — «Dynamic Link Library». Эти библиотеки могут быть загружены *динамически*, прямо во время работы программы.

Внедрением библиотеки называется её загрузка в адресное пространство чужого процесса с последующим выполнением кода из внедрённой библиотеки. Существует множество способов внедрения библиотек в ОС Windows. В работе был выбран механизм внедрения при помощи системного вызова `NtQueueUserAPC()`, обеспечивающий универсальность и возможность внедрения библиотеки в процесс с отличающейся битностью.

В современных версиях ОС Windows могут выполняться как 64-битные процессы, так и 32-битные. В защищённом и расширенном защищённом режимах процессор определяет битность выполняемого кода при помощи параметров дескриптора сегмента, селектор которого находится в сегментном регистре `CS`. Исследованная в работе техника, позволяющая изменять разрядность исполняющегося в процессе кода, называется «Heaven's Gate».

Для разработки приложения-песочницы, реализующего заданные требования, в ходе работы были реализованы следующие компоненты:

- `HookDLL`: внедряемая в целевой процесс библиотека, обеспечивающая изоляцию файловой системы путём ограничения системных вызовов.
- `Sandy.dll`: библиотека, предоставляющая интерфейс для запуска изолированного процесса.
- `SandyElevate`: приложение, обеспечивающее запуск изолированного процесса с повышенными правами.
- `Sandy-GUI`: графический интерфейс пользователя, предоставляющий возможности управления правами доступа, создания, редактирования и удаления конфигураций (наборов прав) и запуска заданного процесса в песочнице.

Для обеспечения модульности и лучшего переиспользования кода были

реализованы дополнительные разделяемые компоненты:

— `InjectionLib`: компонент, обеспечивающий внедрение библиотек при помощи системного вызова `NtQueueUserAPC`.

— `PermissionsLib`: компонент, осуществляющий проверку прав и ограничение доступа путём предоставления ограниченных обёрток над системными вызовами.

— `UtilityLib`: компонент, содержащий различные вспомогательные функции и типы данных: работа с низкоуровневыми путями, определения недокументированных структур и прочее.

В третьей главе описана разработка необходимых компонентов приложения с использованием описанного в предыдущих главах теоретического материала.

HookDLL — внедряемая библиотека: основной компонент, назначением которого является получение конфигурации, инициализация, низкоуровневая работа с хуками (установка, активация, выгрузка при отключении), открытие консоли для отладочного вывода, загрузка и активация плагинов. `HookDLL` компилируется в разделяемую библиотеку, внедрение которой обеспечивает изоляцию файловой системы целевого процесса. Точка входа этого компонента — экспортируемая функция `Execute()`, принимающая указатель на динамическую структуру `HookDllConfig`, содержащую конфигурацию изоляции.

InjectionLib — разделяемый компонент, предоставляющий функциональность универсального внедрения библиотек. Публичный интерфейс компонента состоит из заголовочного файла `Injection.h` и двух функций: `injectDll()` для внедрения произвольной библиотеки и `injectSelf()` для внедрения текущего модуля. Обе функции принимают параметр конфигурации, в котором целиком записана сформированная структура `HookDllConfig`.

Внутренняя реализация компонента содержит файлы `WOW64.cpp` (вспомогательная библиотека для выполнения 64-битного кода в контексте 32-битного приложения), `Injection.cpp` (реализация публичного интерфейса), а также, в заранее скомпилированном виде, написанный на ассемблере шеллкод внедрения, непосредственно выполняющийся в адресном пространстве целевого процесса и обеспечивающий загрузку библиотеки и вызов точки входа `Execute()`.

PermissionsLib — компонент, реализующий всю функциональность, свя-

занную с правами доступа и их применению. Базовые типы данных, используемые в компоненте:

— `Rule`: **Правило**, пара из пути и разрешений для этого объекта.

— `AppConfig`: конфигурация изоляции, содержащая правила, разрешения по умолчанию и другие поля. В отличие от структуры `HookDllConfig`, которой этот тип соответствует, хранит поля в виде обычных `C++`-объектов (строк, векторов). Для передачи объекта этого типа в более низкоуровневые компоненты его следует сериализовать.

— `StartConfig`: конфигурация запуска. Содержит путь к запускаемому приложению, аргументы командной строки и конфигурацию изоляции. Требуется для рекурсивного внедрения в процессы, создаваемые семейством функций `ShellExecute`.

У всех описанных типов данных определены функции сериализации и десериализации в формат `JSON`.

Для реализации проверки прав доступа компоненту `PermissionsLib` требуется эффективный метод выбора подходящих под заданный путь правил из их предопределённого набора. Наивный подход решения этой задачи состоит в последовательной проверке каждого правила из набора, однако это требует неоправданно больших затрат на сравнение уже обработанных и отброшенных префиксов. Для решения этой проблемы был разработан метод выбора правил при помощи *префиксного дерева*.

При рассмотрении наивного решения можно заметить, что существенная часть лишних сравнений происходит во время проверки правил, начальные директории — префиксы — которых уже были проверены и отброшены. Логичным шагом будет рекурсивно сгруппировать правила по компонентам, и рассматривать только те правила, префиксы которых соответствуют проверяемому пути. Образец такой группировки представлен на рисунке 1.

Данное дерево было реализовано при помощи классов `PathTree` (непосредственно структура данных с низкоуровневыми операциями поиска), `PathList` (абстрактный «список правил», содержащий дерево правил для каждого присутствующего в наборе диска) и `PathList::Adapter` — обёртки над конкретным узлом дерева, необходимой для проверки прав во время перечисления файлов в конкретной директории.

SandyElevateProxy — компонент, предназначенный для изолирования про-

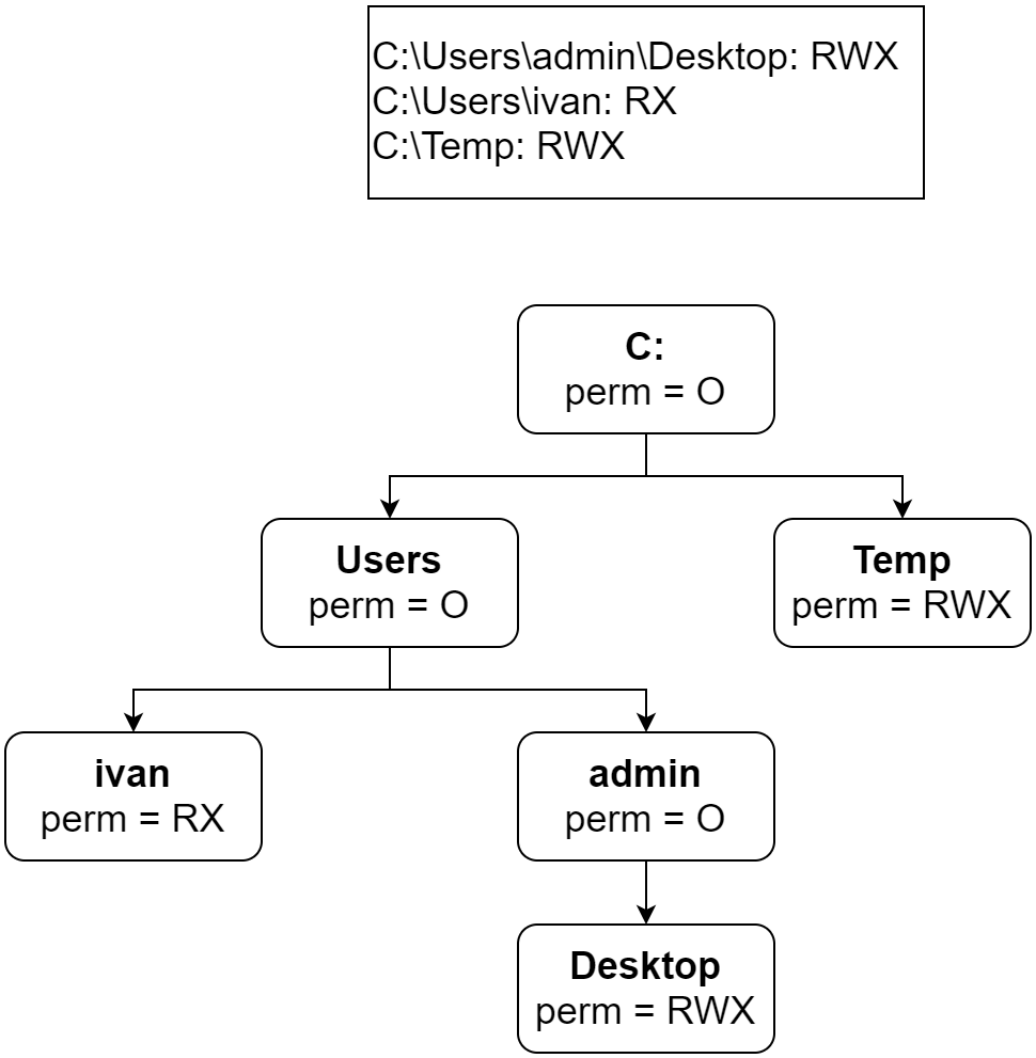

Рисунок 1 – Пример префиксного дерева, составленного из трёх правил

цесса, запущенного с повышением уровня целостности (*Integrity Level*). В ОС Windows пользовательские процессы по умолчанию имеют средний (*medium*) уровень целостности (УЦ). Для совершения административных действий — например, изменение системных файлов и настроек, запись в большинство веток реестра — процесс должен обладать высоким (*high*) УЦ. Переход от низкого УЦ к высокому называется *повышением уровня целостности* (*elevating*) и выполняется при помощи вызова функции из семейства `ShellExecute` с заданным *глаголом* (*verb*) `runas`. Этот вызов запускает переданный ему исполняемый файл в режиме с повышенными правами, предварительно запросив пользователя о допустимости такого действия при помощи механизма UAC.

От разрабатываемой песочницы требуется рекурсивно внедрять себя во все процессы, создаваемые изолированным приложением. Однако механизм обязательного контроля целостности (*Mandatory Integrity Control*) делает

невозможной модификацию памяти процесса с более высоким УЦ. Для обхода этого ограничения и был создан компонент SandyElevateProxy. Он представляет из себя утилиту, принимающую в качестве аргумента название непостоянного отображённого в память файла, в котором находится сериализованная в формат JSON конфигурация запуска. Эта утилита запускается с повышенными правами вместо оригинального исполняемого файла, после чего, используя компонент Sandy.dll, запускает оригинальный исполняемый файл и внедряет в него библиотеку HookDll с переданной конфигураций запуска, обеспечивая рекурсивную изоляцию файловой системы.

Компонент состоит из единственного исходного файла Main.cpp с функцией wmain, для обеспечения передачи конфигурации запуска через непостоянный отображённый в памяти файл использующей функции из модуля ShellExecuteIPC компонента PermissionsLib.

Sandy.dll — библиотека, предоставляющая функцию startSandyApp, запускающую приложение в песочнице Sandy. Компонент состоит из следующих файлов:

- dllmain.cpp: содержит точку входа DLL. Точка входа не используется в компоненте, однако должна быть определена для корректного использования библиотеки.
- Exports.h и Exports.cpp: интерфейс и реализация библиотеки. В нём определено две экспортируемые функции: startSandyApp и disableSRP (вспомогательная функция, отключающая механизм политики ограничения приложений Windows для текущего процесса).
- Starter.h и Starter.cpp: объявление и реализация внутренней функции startApp, принимающей JSON-документ с конфигурацией запуска, и запускающей указанное в конфигурации приложение внутри песочницы. Запуск происходит путём создания нового приостановленного процесса вызовом CreateProcessW и внедрения в созданный процесс библиотеки HookDll при помощи компонента InjectionLib.

Sandy-GUI — графический интерфейс пользователя, разработанный на языке C#.

Главное окно приложения разработано в визуальном редакторе среды Visual Studio и соответствует классу RulesForm. В окне расположены следующие функциональные компоненты:

— dataGridRules: компонент DataGridView, отображающий список правил. Каждое правило состоит из строки с путём и четырёх переключателей, устанавливающих разрешения для указанного пути: на открытие папки (O), на чтение (R), на запись (W) и на выполнение (X).
— lstConfigurations: список конфигураций (наборов правил). Конфигурации загружаются из базы данных конфигураций.
— Группа panelDefaultPerms: содержит четыре переключателя для разрешений по умолчанию.
— Группа panelDefaultComponentsPerms: содержит четыре переключателя для разрешений компонентов путей правил.
— btnDisableSRP: кнопка, отключающая политики ограничений приложений (SRP) Windows для песочницы. Используется в ситуациях, когда необходимо запустить недоверенное приложение, обычное исполнение которого запрещено политиками ограничений приложений по соображениям безопасности, однако допустимо исполнение в песочнице.
— checkBoxDisableSRP: переключатель, отключающий SRP в запускаемом процессе.
— btnNewConfig: кнопка создания новой конфигурации.
— btnCopyConfig: кнопка копирования выбранной конфигурации.
— btnRenameConfig: кнопка переименования выбранной конфигурации.
— btnSaveConfig: кнопка сохранения выбранной конфигурации в базу данных.
— btnDelConfig: кнопка удаления выбранной конфигурации.
— txtAppPath: текстовое поле, задающее путь к исполняемому файлу, который необходимо запустить в песочнице.
— txtCmdLine: текстовое поле, задающее аргументы для передаче запускаемому приложению.
— btnStart: кнопка, запускающая заданное приложение внутри песочницы с выбранной конфигурацией.

Изображение главного окна приложения приведён на рисунке 2.

Помимо главного, в приложении имеется дополнительное модальное окно установки параметров конфигурации. Это окно соответствует классу ChooseConfigParamsForm, отображается при создании, изменении или копировании конфигурации и содержит следующие функциональные компоненты:

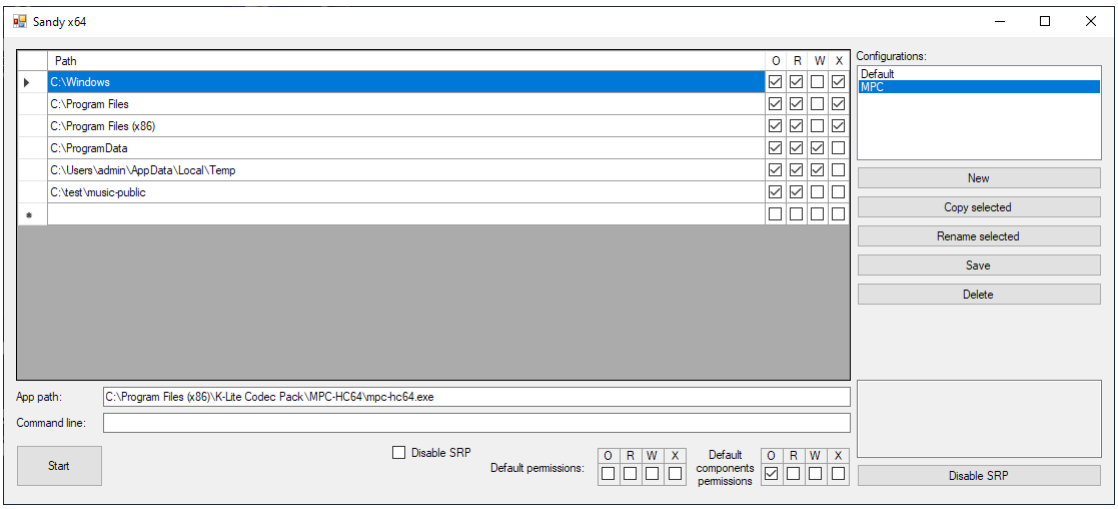

Рисунок 2 – Главное окно приложения Sandy

— `txtName`: текстовое поле с названием конфигурации.
— `checkBoxGlobal`: переключатель глобальности конфигурации.
— `btnCancel`: кнопка отмены.
— `btnSave`: кнопка сохранения параметров конфигурации.

Для хранения конфигураций используется СУБД `SQLite`. Данная встраиваемая СУБД хранит базу данных в едином файле, располагающемся по пути *%LocalAppData%/Sandy/SandyDB.sqlite* (при отсутствии на компьютере пользователя путь создаётся автоматически в методе инициализации `SandyDirs.InitSandy()`), где %LocalAppData% — стандартная директория Windows для хранения данных приложений, специфичных для текущего пользователя.

База данных состоит из следующих таблиц:

— `Configurations`: таблица с информацией о конфигурациях. Содержит столбцы *id* (идентификатор), *name* (имя конфигурации), *defaultPerms* (разрешения по умолчанию), *defaultCompPerms* (разрешения для компонентов путей правил), *isGlobal* (является ли конфигураций глобальной, для любых приложений) и *isPermanent* (является ли конфигурация неудаляемой).
— `Rules`: таблица с информацией о правилах. Содержит столбцы *id* (идентификатор), *fspath* (путь к объекту), *permissions* (разрешения для указанного пути) и *confId* (идентификатор конфигурации, которой принадлежит текущее правило; связь N:1).
— `Apps`: таблица приложений для разработки поддержки специфичных для

конкретных исполняемых файлов конфигураций. В текущей версии не используется.

— `ConfigList`: таблица N:M связи приложений с конфигурациями. В текущей версии не используется.

ЗАКЛЮЧЕНИЕ

Целью данной работы являлась разработка легковесной изолированной среды (песочницы) для защиты пользовательских файлов от несанкционированной модификации недоверенными программами.

Для достижения поставленной цели решались следующие задачи:

- Обзор и анализ существующих решений.
- Обзор способов создания изолированных сред.
- Постановка требований к создаваемому приложению.
- Исследование предметной области.
- Разработка приложения в соответствии с поставленными требованиями и проведёнными исследованиями.

Актуальность работы связана с возросшим количеством вредоносных программ, несанкционированно изменяющих (уничтожающих) пользовательские файлы, и трудностью борьбы с подобным классом вредоносного программного обеспечения классическими способами: сигнатурным, поведенческим и эвристическим анализами.

Новизна работы заключается в разработке гибкой легковесной песочницы, работающей в пространстве пользователя и не требующей установки ядерного модуля. Основные отличия от проанализированных существующих решений заключаются в отсутствии снижения производительности нерелевантных системных вызовов, работе в пространстве пользователя и гибкой настройке разрешений.

Таким образом, поставленные в работе цели и задачи выполнены.