Проект ‹‹Программное средство автоматизированного аудита и харденинга операционной системы на основе стандарта SCAP››

Ученик ФИО

Московская область - 2026

Оглавление

ИСПОЛЬЗУЕМЫЕ ТЕРМИНЫ И СОКРАЩЕНИЯ....3
ВВЕДЕНИЕ....6
1. ИССЛЕДОВАНИЕ ПРОБЛЕМЫ....7
1.1 Проблема и её актуальность....7
1.2 ПО для автоматического анализа уязвимостей....7
1.3 SCAP протокол....8
1.4 Анализ аналогичных решений....11
1.5 Аудитория проекта....12
2. РЕШЕНИЕ ПРОБЛЕМЫ....14
2.1 Формализация проблемы....14
2.2 Описание алгоритма решения проблемы....15
2.3 Описание существующих алгоритмов....16
2.4 Модель угрозы, MITRE ATT&CK....17
3. ОПИСАНИЕ РЕАЛИЗАЦИИ....20
3.1 Техническое задание.....20
3.2 Описание идеи проекта....21
3.3 Используемые технологии.....21
3.4 Принцип работы программы....24
3.5 Доказательство работоспособности....25
3.6 Функциональное тестирование....25
3.7 Перспективы развития проекта....26
ЗАКЛЮЧЕНИЕ....28
СПИСОК ЛИТЕРАТУРЫ....30
ПРИЛОЖЕНИЕ....31

ИСПОЛЬЗУЕМЫЕ ТЕРМИНЫ И СОКРАЩЕНИЯ

SCAP - Security Content Automation Protocol: набор открытых стандартов для представления/обмена данными ИБ и автоматизации проверок.

CVE - идентификаторы известных уязвимостей.

CWE - классификация типов слабостей (уязвимых паттернов).

CCE - идентификаторы параметров/настроек конфигурации.

CVSS - методика оценки критичности уязвимостей.

OVAL - Open Vulnerability and Assessment Language: язык (стандарт) описания и выполнения технических проверок состояния системы.

XCCDF - Extensible Configuration Checklist Description Format: формат описания чек-листов/политик проверок, профилей и результатов; поддерживает remediation.

CIS - набор бенчмарков/рекомендаций по безопасной конфигурации.

ОС - операционная система.

ПО - программное обеспечение.

XML - формат данных.

JSON - формат данных.

PDF - формат данных.

HTML - формат данных.

JS - JavaScript.

Qt - фреймворк для GUI (графического интерфейса).

GTK - фреймворк GUI.

GPL - лицензия GNU General Public License.

LGPL - лицензия GNU Lesser General Public License.

PyQt - Python-библиотека (биндинги) для Qt.

PySide - Python-библиотека (биндинги) для Qt.

C++ - язык программирования.

Dart - язык программирования.

openSCAP - инструментарий/движок для обработки SCAP-контента.

Linux / Windows / macOS - платформы.

Open Source - открытый исходный код.

SCC - SCAP-инструмент.

Ovaldi – движок для использования SCAP контента.

RedCheck, XSpider, ScanOVAL, MaxPatrol 8, MaxPatrol VM - названия продуктов/аналогов.

Аудит (информационной безопасности) - регулярные проверки, отчётность, анализ состояния защищённости.

Харденинг - укрепление конфигурации ОС/системы (снижение поверхности атаки).

Киберугрозы - угрозы атак на ИС.

Угроза – мнимая или реальная опасность

Уязвимость - это слабость в системе, которая может привести к реализации угрозы.

Мисконфигурация - небезопасная/ошибочная настройка.

Стандартизация проверок/данных - унификация форматов правил и результатов.

SCAP-контент - набор машиночитаемых правил/описаний проверок (OVAL/XCCDF).

OVAL-дефиниция - отдельное правило/описание проверки в формате OVAL.

XCCDF-правило - отдельное требование/проверка в XCCDF.

XCCDF-профиль - набор (выборка) правил XCCDF для конкретной политики.

Tailoring - адаптация (кастомизация) SCAP/XCCDF-контента под целевую платформу при несовпадении идентификаторов.

Remediation - действия по устранению несоответствий (в т.ч. автоматическое исправление).

Проприетарный - закрытый (непрозрачный) механизм/формат/подход вендора.

Вендор - поставщик продукта/решения.

SCAP-интерпретатор / движок - компонент, который исполняет проверки по SCAP-контенту.

Агрегация результатов - это сбор данных из файлов результатов.

Vulnerability management - управление уязвимостями.

Сигнатурное/базовое сопоставление - подход: сбор версий ПО и сравнение с базой уязвимых версий.

Оценка соответствия конфигурации - проверка системы на соответствие политике/профилю требований.

ВВЕДЕНИЕ

В наше время информационные технологии используются во всех областях, начиная от бизнеса и заканчивая государственными организациями. Цифровая структура всех этих областей растёт крайне стремительно, что также повышает требования к обеспечению защищённости систем. При этом киберугрозы развиваются также быстро, как и технологии: злоумышленники используют широкий спектр методов - от известный уязвимостей до ошибок конфигураций. В таких условиях требуется непрерывный мониторинг информационной безопасности систем, обычно это достигается за счёт регулярных проверок, формирования отчётностей и последующего устранения выявленных уязвимостей.

Важной составляющей таких процессов является стандартизация данных и проверок. Использование стандартов позволяет унифицировать проверки и их результаты, а также снизить зависимость от конкретных инструментов. В области автоматизации контроля безопасности существует единственный набор открытых стандартов SCAP, предназначенный для описания и обмена данными по безопасности, а также для автоматизирования проверок.

Цель работы: Создание программного средства автоматизированного аудита и харденинга операционной системы на основе стандарта SCAP

Задачи:

1. Реализовать визуальный интерфейс программы для запуска проверок по SCAP-контенту и просмотра результатов в удобном для чтения виде.
2. Разработать модуль выполнения проверок по OVAL-дефинициям
3. Разработать модуль выполнения проверок по XCCDF-контенту
4. Реализовать модуль для автоматического tailoring
5. Добавить поддержку remediation для XCCDF
6. Реализовать модуль подгрузки SCAP-контента для выбранной операционной системы
7. Реализовать модуль агрегации результатов проверки из XML файлов, с последующим выводом в табличном виде

1. ИССЛЕДОВАНИЕ ПРОБЛЕМЫ

1.1 Проблема и её актуальность

С каждым днём цифровая инфраструктура компаний растёт, число угроз увеличивается, а злоумышленники активно используют публично известные уязвимости и ошибки конфигураций. На этом фоне заметна общая тенденция - темпы появления уязвимостей увеличиваются. По графику с Рис. П1.1 видно, что лишь за период с 2015 по 2019 год количество опубликованных уязвимостей выросло на 200% (с 6000 до 1800), а с 2019 по 2023 - ещё на 55% (с 18000 до 28000). Это означает, что объём информации, которую необходимо анализировать и обрабатывать специалистам из области информационной безопасности, растёт непрерывно уже более 10 лет, что делает задачу автоматизации процессов выявления и устранения уязвимостей и мисконфигураций особенно актуальной.

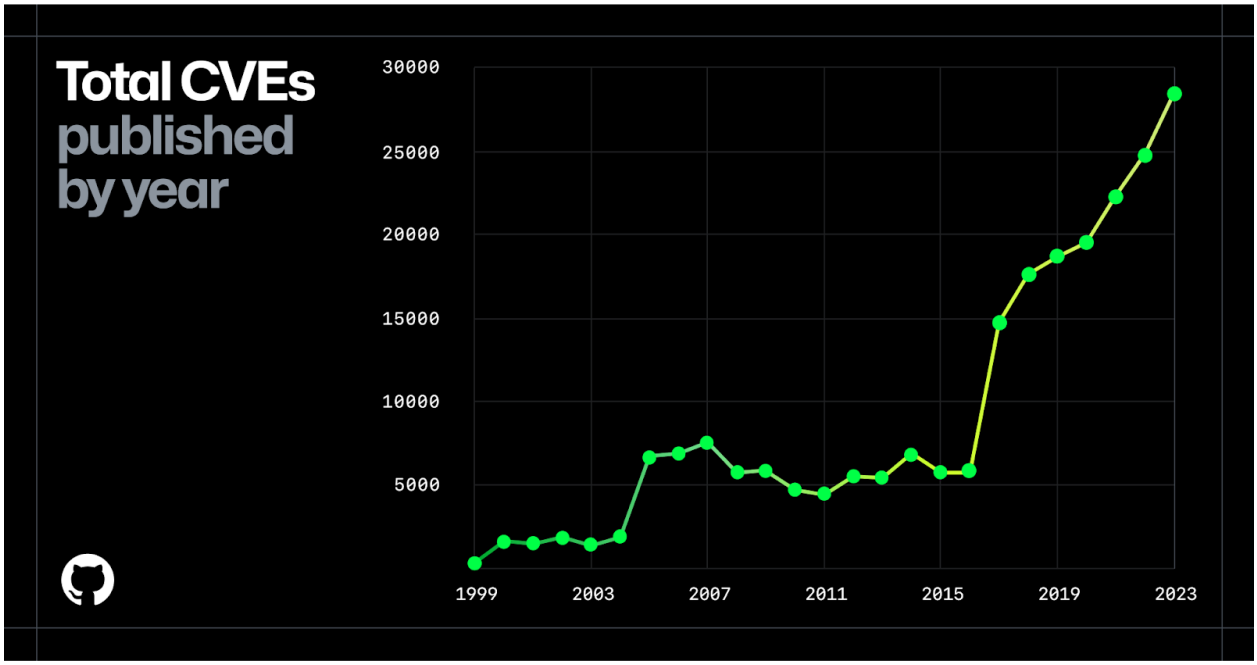

Рис. П1.1 График количества найденных уязвимостей по годам

1.2 ПО для автоматического анализа уязвимостей

На данный момент существует крайне большое количество инструментов для поиска и управления уязвимостями в информационных системах. Среди распространённых решений можно выделить Redcheck, MaxPatrol 8, XSpider, MaxPatrol VM. Эти продукты помогают проводить аудит уязвимостей и формировать отчётность, однако у большинства из них есть ограничения, которые становятся критичными при построении процессов информационной безопасности.

Основные недостатки подобных решений:

- **Проприетарные методы анализа.** Внутренние механизмы проверки и логика обнаружения зачастую закрыты. Это затрудняет независимую верификацию результатов, а также ограничивает расширение или изменение функционала программы без вмешательства вендора.

- **Собственные базы данных и форматы представления контента для проверки.** Многие вендоры агрегируют сведения об уязвимостях, например CVE, в собственные базы/JSON/XML-инструкции. Данный подход приводит к зависимости от конкретного вендора и затруднению разработки инструмента, работающего с программой или её контентом.

Именно для избегания данных проблем разработанная программа на каждом этапе будет опираться на открытые стандарты SCAP.

1.3 SCAP протокол

SCAP (Security Content Automation Protocol, протокол автоматизации управления данными безопасности) - это набор открытых стандартов, определяющих технические спецификации для представления, описания и обмена данными в области безопасности. Его задача - обеспечить единый подход к автоматизации проверок: от описания уязвимостей и конфигураций до форматов тестов и результатов оценки. В состав SCAP входят различные компоненты, среди которых:

- **CVE** - идентификаторы известных уязвимостей;

- **CWE** - классификация типов слабостей;

- **CCE** - идентификаторы параметров конфигураций;

- **CVSS** - методика оценки критичности;

- **XCCDF** - формат описания контрольных листов и политик проверок;

- **OVAL** - язык описания и выполнения технических проверок состояния системы.

При создании программы ключевыми компонентами будут являться CVE, OVAL и XCCDF, потому что именно благодаря им программа будет проверять систему на наличие уязвимостей и мисконфигураций по машиночитаемому описанию проверок. Роль этих трёх

компонентов стоит разобрать как в рамках разбора работы протокола SCAP, так и в рамках реализации программы:

- **CVE** предоставляет унифицированные идентификаторы уязвимостей и базовую справочную информацию. Большинство продуктов на рынке ограничиваются только агрегацией информации из CVE и последующим формированием собственных правил. Но в протоколе SCAP ролью CVE является только базовое описание уязвимости и предоставление ей идентификатора

- **OVAL (Open Vulnerability and Assessment Language)** - это открытый стандартизированный язык, который в машиночитаемом виде предоставляет информацию как проверять систему на наличие уязвимостей. OVAL имеет одинаковую структуру вне зависимости от вендора.

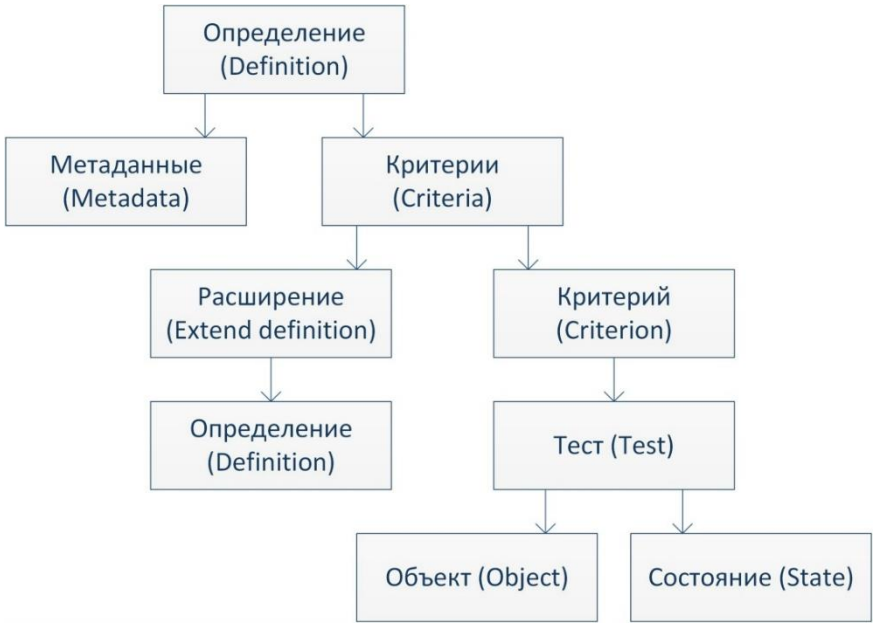

Рис. П1.2 Структура OVAL

- **XCCDF (Extensible Configuration Checklist Description Format)** - это формат, который позволяет описывать политики и структуру проверок: какие требования проверять, как их группировать, какие профили использовать, как формировать результат оценки. Также благодаря XCCDF можно проводить remediation (устранение уязвимостей).

Рис. П1.3 Структура XCCDF

Для разработки программы по автоматизации обнаружения уязвимостей необходимо изучить общий процесс работы SCAP (Рис. П2.1). Также в рамках целей проекта необходимо разобрать техническую реализацию взаимодействия OVAL-дефиниций и XCCDF-контента (Рис. П2.2).

Рис. П1.4 Процесс работы протокола SCAP

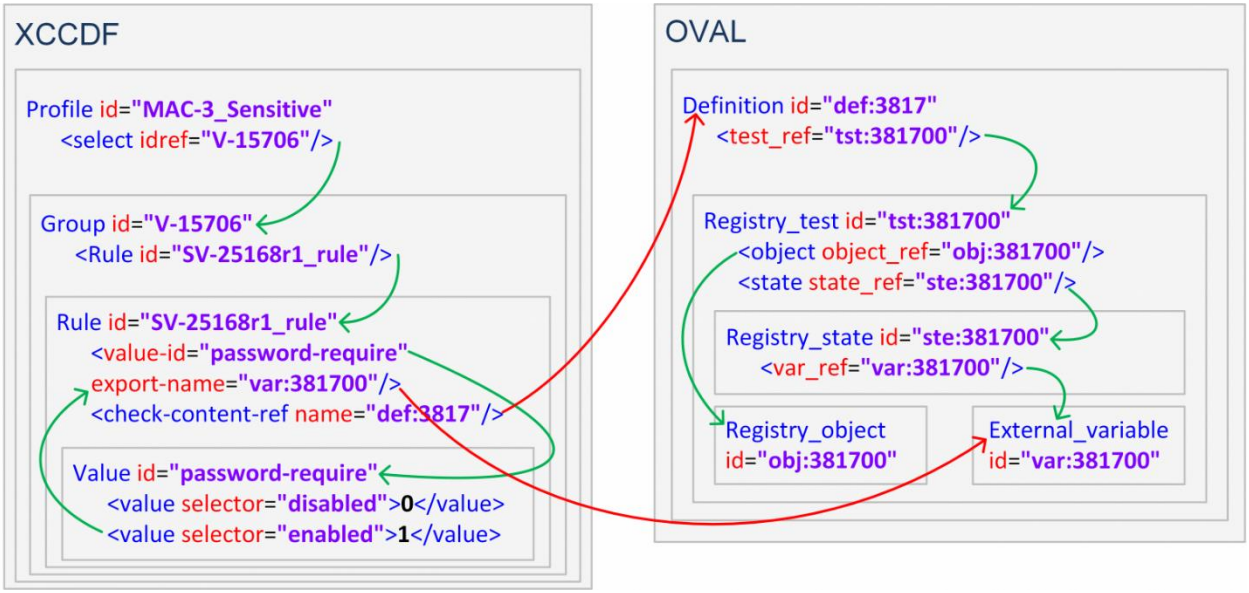

Рис. П1.5 Структура взаимодействия XCCDF и OVAL на примере CIS Benchmark

1.4 Анализ аналогичных решений

Для определения новизны и уникальности создаваемой программы был проведён анализ существующих решений для поиска и управления уязвимостями. В качестве аналогов были рассмотрены: RedCheck, XSpider, ScanOVAL, MaxPatrol VM.

Инструмент	Поддержка SCAP	OpenSource	Цена	Remediation	Tailoring
Xspider	Нет	Нет	Платно	Нет	Нет
RedCheck	Да	Нет	Платно	Да	Нет
ScanOVAL	Частичная (только OVAL-контент от ФСТЭК)	Нет	Бесплатно	Нет	Нет
MaxPatrol VM	Нет	Нет	Платно	Да	Нет
Проектная программа	Да	Да	Бесплатно	Да	Да

Табл. 1. Сравнение с аналогами

Сравнение выполнено по следующим критериям: **поддержка SCAP**, **открытость исходного кода (Open Source)**, **стоимость**, наличие **remediation** (механизмов устранения

или рекомендаций по устранению), а также **tailoring** (возможности адаптации и кастомизации проверочного контента).

XSpider является примером классического сканера уязвимостей. Инструмент ориентирован на собственные механизмы обнаружения и не использует SCAP-контент, поэтому в рамках данного подхода в нём отсутствуют поддержка tailoring и remediation. Решение не является Open Source и распространяется на коммерческой основе.

RedCheck относится к решениям, поддерживающим SCAP-контент. Продукт предоставляет remediation функционал, однако не реализует механизмы tailoring для адаптации проверочного контента под целевую систему. Кроме того, решение является закрытым и коммерческим.

ScanOVAL это бесплатный инструмент, ориентированный на проверку системы по OVAL-дефинициям, что означает только частичную поддержку SCAP. Поскольку программа не поддерживает XCCDF-контент, в ней отсутствуют возможности tailoring и remediation в контексте XCCDF-проверок.

MaxPatrol VM представляет собой коммерческую платформу управления уязвимостями, ориентированную на полный цикл vulnerability management. За счёт этого в продукте реализованы механизмы remediation. Вместе с тем, MaxPatrol VM не предполагает использования SCAP-контента.

Таким образом, рассмотренные решения либо совсем не поддерживают SCAP-контент, либо являются закрытыми коммерческими продуктами. Кроме того, большинство программ, поддерживающих SCAP-контент, не обладают возможностью автоматической адаптации (tailoring) SCAP-контента под конкретную операционную систему.

1.5 Аудитория проекта

Поскольку разрабатываемая программа предназначена для автоматизированного аудита системы и поддержки харденига на основе SCAP-контента (OVAL/XCCDF), её целевая аудитория напрямую связана с задачами контроля защищённости, соответствия требованиям и регулярной проверки конфигураций и уязвимостей. Соответственно, основными группами пользователей являются:

- **Аудиторы информационной безопасности**. Для них актуальна задача получения воспроизводимых результатов проверок, формирования отчётности и подтверждения выполнения требований (политик безопасности, внутренних регламентов и внешних стандартов). Использование стандартизированного контента облегчает сравнение результатов между системами и аудитами.

- **Системные администраторы**. Для них программа полезна как средство быстрой проверки состояния системы, выявления уязвимостей и мисконфигураций, а также устранения уязвимостей, что упрощает поддержание безопасности информационных систем.

Аудитория проекта является профессиональной и относительно ограниченной, поскольку ориентирована на специализированные задачи, однако программа имеет широкое применение в организациях, где нужен постоянный аудит и стандартизированная проверка безопасности.

2. РЕШЕНИЕ ПРОБЛЕМЫ

2.1 Формализация проблемы

В рамках данной программы решается задача автоматизированной проверки информационной системы на наличие уязвимостей и нарушений конфигураций в системе на основе стандартизированного SCAP-контента. Для исключения неоднозначности и возможности воспроизведения алгоритма задача формализуется в виде входных данных, выходных данных, ограничений и алгоритма обработки:

Входные данные:

1. Целевая система S - проверяемая операционная система

2. SCAP-контент C, включает набор OVAL-дефиниций D = {d1,d2,d3...,dn} и XCCDF-правил R = {r1,r2,r3..rn}, в случае XCCDF также содержат данные remediation RD = {rd_r1, rd_2, rd_r3...rd_rn}

3. Параметры запуска P: ОС, профиль XCCDF, режим выполнения проверок.

Выходные Данные:

- Результаты при проведении проверки по OVAL: для каждой дефиниции di вычисляется статус Result_OVAL ∈ {true,false,unknown,error,not_applicable}, где true - уязвимость/несоответствие присутствует, false - уязвимость/несоответствие отсутствует, unknown/error/not_applicable - невозможность или неприменимость проверки.

- Результаты при проведении проверки по XCCDF: для каждого правила ri вычисляется статус Result_XCCDF ∈ {pass,fail,not_checked,not_applicable,error}, где pass - правило успешно выполнилось, fail - правило выполнилось не успешно, not_checked,not_applicable,error - невозможность или неприменимость проверки. При включённом remediation для каждого fail ri выполняется rdi

3. Отчёт Report в формате таблицы, содержащей для каждой проверки:

- При проверке OVAL-дефиниций Report содержит: cve_id, di, описание уязвимости

- При проверке XCCDF-правил Report содержит: XCCDF_rule, di, описание правила, уровень важности правила.

2.2 Описание алгоритма решения проблемы

Формализованный алгоритм работы программы:

1. Выбор контента и подготовка среды: определяется платформа S, загружается SCAP-контент C, задаются параметры запуска P

2. Tailoring (при необходимости): если контент содержит идентификаторы платформы, не совпадающие с целевой системой, формируется tailoring файл для успешной работы SCAP-контента C на S.

3. Выполнение OVAL-проверок: для каждой дефиниции di вычисляются требуемые свойства системы (версии, параметры, состояния объектов), после чего определяется Result_OVAL(di)

4. Выполнение XCCDF-проверок: на основе выбранного профиля вычисляется результат для каждого правила ri, после чего определяется Result_XCCDF(ri). Если включен remediation - уязвимость исправляется при наличии remediation контента в XCCDF.

5. Формирование отчёта: результаты Result_OVAL и Result_XCCDF агрегируются и преобразуются в удобный для анализа вид (таблица).

6. Вывод пользователю: отчёт предоставляется пользователю в человекочитаемом виде внутри интерфейса программы.

Описание алгоритма решения проблемы:

Режим 1: Проверка по запросу (OVAL)

Данный режим используется для разовой оценки системы и запускается пользователем вручную.

1. Выбор ОС и контента. Пользователь выбирает целевую платформу (ОС/дистрибутив), далее программа скачивает SCAP-контент.

2. Выполнение проверок. Программа последовательно выполняет OVAL-проверки (получая факты о системе: версии ПО, параметры, состояния объектов) и записывает их в список с результатами проверки

3. Формирование результатов. Пользователь получает сводный результат в виде таблицы: идентификаторы CVE, статусы (true/false и т.д.), описания CVE.

Режим 2: Проверка по запросу (XCCDF + remediation)

Данный режим используется для разовой оценки системы и запускается пользователем вручную.

1. Выбор ОС и контента. Пользователь выбирает целевую платформу (ОС/дистрибутив), далее программа скачивает SCAP-контент, после чего пользователь выбирает XCCDF профиль.

2. Адаптация (tailoring). Если выбранный XCCDF-контент предназначен для близкой платформы, но не применяется из-за различающегося идентификатора системы, программа выполняет автоматическую адаптацию контента (tailoring), обеспечивая возможность корректного запуска проверок на текущей системе.

3. Выполнение проверок. Программа последовательно выполняет OVAL-проверки (получая факты о системе: версии ПО, параметры, состояния объектов), затем вычисляет результаты XCCDF-правил на основе выбранного профиля и записывает их в список с результатами проверки.

4. Формирование результатов. Пользователь получает сводный результат в виде таблицы: идентификаторы проверок, описания, статусы (pass/fail и т.д.), важность проверок.

2.3 Описание существующих алгоритмов

Помимо алгоритмов, взятых из стандартов SCAP, таких как OVAL evaluation и XCCDF evaluation также стоит рассмотреть несколько других методов анализа системы на небезопасные конфигурации и уязвимости:

Сигнатурное/базовое сопоставление - этот алгоритм заключается в сборе информации о ПО в системе (инвентаризация пакетов, версий) и сопоставления их с базой уязвимых версий программ. Данный алгоритм используется только для нахождения CVE, позволяет быстро масштабировать базу данных, но обладает минусами в виде большого числа ложных срабатываний, сложностью написания сложных условий и плохой возможностью описания конфигурационных требований к системе.

Алгоритмы оценки соответствия конфигурации - этот алгоритм заключается в наличии политики/профиля требований, для каждого требования запускается проверка состояния, затем агрегируются результаты (pass/fail/not applicable) и строится отчёт по профилю.

SCAP алгоритмы же, как мы можем видеть, объединяют самые лучшие практики из множества уже существующих алгоритмов нахождения уязвимостей и небезопасных конфигураций

2.4 Модель угрозы, MITRE ATT&CK

Проект предназначен для повышения защищённости Linux-систем за счёт регулярного автоматизированного аудита и харденинга на основе стандартизированного SCAP-контента. Для формального описания того, какие этапы атаки усложняет харденинг, используется структура фреймворка MITRE ATT&CK.

Целевой класс угроз: Угрозы, использующие публично известные уязвимости и мисконфигурации.

Общая цель злоумышленника: использовать известные уязвимости и/или ошибки конфигурации для получения доступа к системе, расширения прав, закрепления и дальнейшего выполнения действий в инфраструктуре.

1. Соответствие тактикам и техникам MITRE ATT&CK

Ниже приведены примеры техник MITRE ATT&CK, для которых результаты аудита и remediation по SCAP-контенту выступают превентивной контрмерой (через харденинг и устранение условий атаки):

1. **T1190: Exploit Public-Facing Application**

Что соответствует: злоумышленники пытаются эксплуатировать слабости в публично доступных сервисах, включая уязвимости ПО и ошибки конфигурации.

Как противодействует проект: проверяет состояние системы и конфигураций по OVAL/XCCDF (версии, параметры, настройки сервисов) и применяет remediation, уменьшая вероятность наличия уязвимых/небезопасно настроенных компонентов на хосте.

2. **T1068: Exploitation for Privilege Escalation**

Что соответствует: эксплуатация уязвимостей ПО/ОС для повышения привилегий.

Как противодействует проект: за счёт проверок уязвимых состояний и их последующего исправления снижает вероятность наличия известных уязвимых конфигураций, используемых для эскалации.

3. **T1562.004 — Impair Defenses: Disable or Modify System Firewall**

Что соответствует: злоумышленник отключает или изменяет правила системного файрвола, чтобы обойти ограничения сетевого трафика.

Как противодействует проект: через SCAP-контент выполняет аудит состояния файрвола и правил (включён ли сервис, разрешённые сервисы/порты и т.д.) и применяет remediation, возвращая конфигурации к заданным стандартам.

4. **T1548.003 — Abuse Elevation Control Mechanism: Sudo and Sudo Caching**

Что соответствует: злоумышленник использует особенности sudo и настройки sudoers, чтобы выполнить команды от имени другого пользователя или запустить процессы с более высокими привилегиями.

Как противодействует проект: через SCAP-контент проверяет и исправляет конфигурацию sudo.

2. Вывод по модели угроз

Разрабатываемая программа является превентивной мерой: за счет автоматизированного аудита и харденинга программа устраняет уязвимости и мисконфигурации, минимизируя поверхность атаки и устраняя возможность эксплуатации типовых техник, тем самым защищая от сразу нескольких тактик злоумышленника по MITRE ATT&CK:

- **TA0001-Initial Access** (T1190-Exploit Public-Facing Application)

- **TA0004-Privilege Escalation** (T1068-Exploitation for Privilege Escalation, T1548.003-Abuse Elevation Control Mechanism: Sudo and Sudo Caching)

- **TA0005-Defense Evasion** (T1548.003-Abuse Elevation Control Mechanism: Sudo and Sudo Caching, T1562.004-Impair Defenses: Disable or Modify System Firewall)

3. ОПИСАНИЕ РЕАЛИЗАЦИИ

3.1 Техническое задание.

Назначение и цель проекта:

Разработать программное средство для автоматизированной проверки информационной системы на наличие уязвимостей и нарушений конфигурации на основе стандартизированного SCAP-контента (OVAL/XCCDF), с формированием табличного и обычного отчёта и возможностью автоматического устранения несоответствий (remediation).

Задачи разработки:

1. Обеспечить загрузку и подготовку SCAP-контента для выбранных операционных систем.
2. Реализовать выполнение проверок по OVAL-дефинициям с получением статусов результатов.
3. Реализовать выполнение проверок по XCCDF-контенту (профили/правила) с получением статусов результатов.
4. Реализовать механизм автоматического tailoring для адаптации XCCDF-контента под целевую систему при несовпадении идентификаторов платформы.
5. Реализовать применение remediation для правил XCCDF со статусом fail при наличии соответствующего контента.
6. Реализовать формирование отчёта в табличном виде и отображение результатов пользователю через графический интерфейс.

Требования к функциональности:

1. Загрузка SCAP-контента: Программа должна предоставлять возможность выбора операционной системы/платформы и загрузки соответствующего SCAP-контента.
2. Выполнение OVAL-проверок: Программа должна интерпретировать OVAL-дефиниции и выполнять проверки состояния системы.
3. Выполнение XCCDF-проверок: Программа должна поддерживать выбор профиля XCCDF (если профили присутствуют). Программа должна интерпретировать XCCDF-правила и выполнять проверки состояния системы.

4. Tailoring: при несовпадении идентификаторов платформы в контенте и на целевой системе программа должна выполнять автоматическую адаптацию (tailoring), обеспечивая применимость выбранного XCCDF-контента на целевой ОС.
5. Remediation: для правил со статусом fail программа должна применять remediation-действия при наличии соответствующего remediation-контента.
6. Отчёт и вывод результатов: Программа должна формировать отчёт в табличном виде, включающий: идентификатор проверки (OVAL/XCCDF), описание, статус, уровень важности (если присутствует в XCCDF).

3.2 Описание идеи проекта

Идея проекта заключается в создании программного средства, которое позволяет автоматизировать аудит и харденинг операционной системы с использованием открытых стандартов SCAP. Это необходимо для того, чтобы упростить и ускорить процесс выявления уязвимостей и мисконфигураций, а также сделать результаты проверок воспроизводимыми и переносимыми между разными системами и инструментами.

В отличие от подходов, основанных на проприетарных базах и закрытых форматах проверок, проект опирается на SCAP-контент (OVAL/XCCDF) как на унифицированный способ описания правил аудита. Программа загружает SCAP-контент для выбранной операционной системы, выполняет проверки по OVAL-дефинициям и XCCDF-правилам, после чего формирует результаты в удобном для анализа и чтения виде. Дополнительно реализованы механизм tailoring, позволяющий адаптировать контент под целевую платформу при несовпадении идентификаторов, и remediation, обеспечивающий автоматическое устранение выявленных несоответствий при наличии соответствующего контента.

Таким образом, проект является программным решением, которое снижает трудозатраты на регулярные проверки безопасности, повышает степень автоматизации и позволяет применять единый стандартизированный подход к аудиту и харденингу операционных систем.

3.3 Используемые технологии.

Этот этап можно считать по настоящему важным, так как от выбранных инструментов зависят многие характеристики конечного продукта. Мы разрабатываем десктопное приложение, вследствие чего выбор фреймворка графического интерфейса

является довольно важным этапом. Для удобного просмотра основных параметров фреймворка была сделана таблица.

Наименование фреймворка, иллюстрация	Поддерживаемые языки	Лицензия	Производительность	размер конечного приложения	Поддерживаемые платформы	Потребление оперативной памяти
 Electron js	JavaScript	MIT	Низкая	Большой	Windows, macOS, Linux	Высокое
 Flutter	Dart	BSD	Высокая	Средний-большой	Windows, macOS, Linux, Android, iOS, Web	Среднее
 GTK	20+ языков, включая c++ и python	LPGL	Средняя (зависит от ОС)	Низкий-средний	Linux (нативно), Windows и macOS.	Низкое-среднее (зависит от ОС)
 Qt	5 языков, включая c++ и python	LPGL, GPL	Высокая	Средний	Windows, macOS, Linux, Android, iOS	Низкое

Табл. 2. Сравнение фреймворков

Самыми главными параметрами при выборе можно по праву считать поддерживаемые платформы и лицензию. Для начала следует разобрать вопрос лицензий. Все виды лицензий (кроме GPL) в таблице почти не накладывают ограничений на использование в коммерческих продуктах. Также, как можно видеть в таблице, GTK имеет слабую поддержку ОС помимо линукса, а Electron js имеет высокое потребление ресурсов из-за использования Chromium. В итоге нам стоит рассмотреть только 2 фреймворка – Flutter и Qt. По своим параметрам они очень похожи, но отличия возникают при их ближайшем рассмотрении: Flutter поддерживает только язык программирования Dart, в то время как на Qt можно писать почти на всех наиболее распространённых языках, что уменьшит время разработки. Также Flutter проигрывает Qt в аспектах веса конечной программы и потребления оперативной памяти. После выбора фреймворка было решено писать на Python, так как по сравнению с c++ с ним будет проще и быстрее работать.

После выбора языка перед нами появляется проблема в виде того, что библиотека PyQt для работы с Qt распространяется по лицензии GPL, что не позволит использовать её в коммерческих проектах. Но также существует и библиотека PySide, которая почти полностью идентична PyQt, но при этом распространяется по лицензии LGPL, соответственно использовать мы будем её.

В качестве инструмента для разработки графического интерфейса программы будет использоваться Qt Designer, а для реализации функционала программы будут использоваться следующие модули: subprocess, xml.etree.ElementTree, sys.

Первая версия программы будет разрабатываться под ОС Linux, так как на Windows SCAP не возымел должной популярности, вследствие чего поддержка данного протокола была прекращена как со стороны как Microsoft, так и со стороны разработчиков инструментария SCAP. Например, OpenSCAP прекратил поддержку ОС Windows в 2022 году. Второй же важной причиной является то, что основной аудиторией программы являются Системные Администраторы, а по официальной статистике около 77% серверов работают под управлением ОС Linux.

Далее идёт выбор инструмента для обработки SCAP контента. Этот этап можно считать по настоящему важным, так как от выбранного инструментария будет зависеть большинство характеристик программы. Для обоснования выбора приведена Табл. 2. По ней мы можем видеть, что SCC не подходит нам, потому что работает только на windows, Ovaldi не подходит из-за отсутствия обновлений и поддержки с 2015, вследствие чего он не будет работать без дописывания его функционала вручную. А самодельный SCAP-интерпретатор не имеет смысла писать по многим причинам, начиная от технической сложности интерпретаторов, заканчивая тем, что он будет работать медленней других аналогичных разработок. В итоге для разработки программы в качестве движка для обработки SCAP контента выбран openSCAP.

Движок	Скорость проверки	Актуальность функционала	Возможность интеграции в программу	Поддерживаемые ОС
OpenSCAP	Высокая	Высокая	Средняя	Linux
SCC (SCAP Compliance Checker)	Низкая	Высокая	Низкая	Windows
Самодельный	Низкая	Низкая	Высокая	Linux/Windows
Ovaldi	Высокая	Низкая	Средняя	Linux, Windows

Табл. 3. Сравнение SCAP-интерпретаторов

3.4 Принцип работы программы

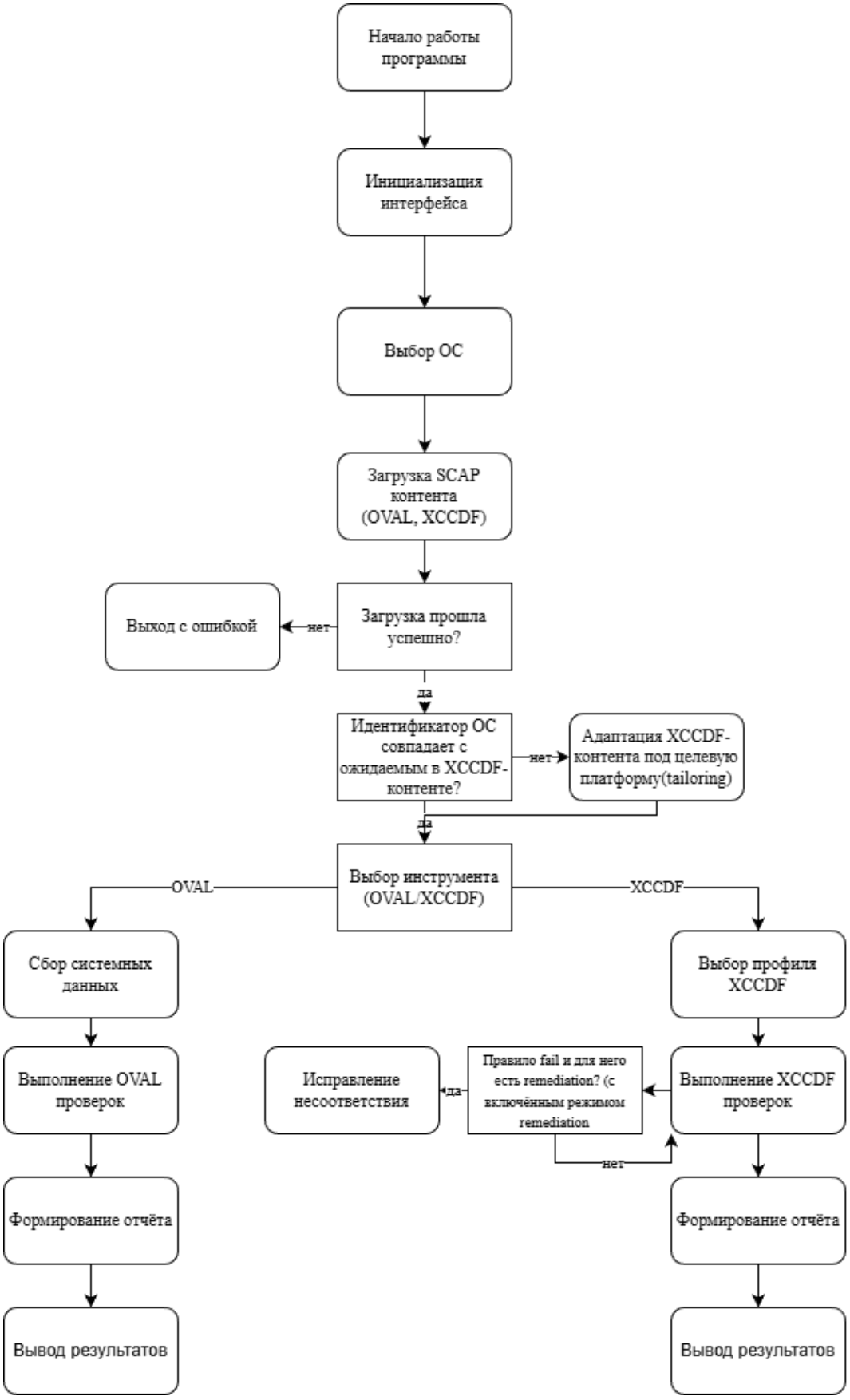

Рис. П2.1 Принцип работы программы

3.5 Доказательство работоспособности

Работоспособность разработанной программы подтверждается тем, что она реализует решение задачи в том виде, в каком она была формально поставлена в разделах 2.1–2.2: определены входные данные S, C, P, выходные данные Result_OVAL, Result_XCCDF, Report и допустимые статусы результатов. Программа выполняет полный цикл: загрузка SCAP-контента, tailoring при необходимости, выполнение проверок, применение remediation и формирование итогового отчёта.

Формальная верификация в рамках данной работы заключается в проверке соответствия реализации указанной формальной модели: для каждого элемента входа должен быть получен корректный элемент выхода, а значения результатов должны принадлежать заранее заданным множествам статусов. Это проверяется следующим образом:

1. Проверка корректности результатов OVAL. Для каждой OVAL-дефиниции di программа вычисляет Result_OVAL(di) и присваивает ему один из формально заданных статусов {true,false,unknown,error,not_applicable}. Полученные значения сохраняются и отображаются в отчёте, что обеспечивает проверяемость и повторяемость результата.
2. Проверка корректности результатов XCCDF. Для каждого XCCDF-правила ri в рамках выбранного профиля вычисляется Result_XCCDF(ri) из множества {pass,fail,not_checked,not_applicable,error}. Полученные значения сохраняются и отображаются в отчёте, что также обеспечивает проверяемость и повторяемость результата.
3. Проверка tailoring. Если исходный контент неприменим к целевой платформе из-за отличающихся идентификаторов, выполняется tailoring, после чего контент становится применимым и корректно проходит этап вычисления результатов.
4. Проверка remediation. Для правил со статусом fail, при наличии remediation-контента, программа выполняет действие исправления.

Таким образом, программа демонстрирует работоспособность не только на уровне запуска и получения отчёта, но и на уровне соответствия формальной постановке задачи: входные данные преобразуются в определённые формально выходные результаты, а корректирующие действия (remediation) приводят систему к состоянию, которое подтверждается повторной проверкой

3.6 Функциональное тестирование

Также было проведено функциональное тестирование на наборе тестов, в качестве операционных систем были взяты: Debian 12, Red Hat Linux Enterprise 9.6, Rocky Linux 9.7,

а в качестве основных функций для тестирования – проверка по OVAL-дефинициям, проверка по XCCDF-правилам и автоматическое генерирование tailoring.

Суть теста	Ожидаемый результат	Получаемый результат
Выполнение проверки OVAL-дефиниций на Debian 12	Все статусы проверок дефиниций вернут true или false	Все статусы проверок дефиниций содержали true или false. Проверка проверила 1428 дефиниц, 1279 из которых вернули false и 149 вернули true. (одна дефиниция проверяла 1 или сразу несколько CVE)
Выполнение проверки XCCDF-правил с профилем xccdf_org.ssgproject.content_profile_cis на Red Hat Linux Enterprise 9.6	Все статусы проверок XCCDF-правил вернут pass или fail	Все статусы проверок XCCDF-правил вернули pass или fail. Всего прошло тестирование 375 правил, 170 из которых вернули pass и 205 вернули fail
Выполнение проверки XCCDF-правил с профилем xccdf_org.ssgproject.content_profile_cis на Rocky Linux 9.7 без автоматического tailoring	Неизвестный результат	Все статусы проверок XCCDF-правил вернули not applicable, хоть Rocky Linux и основан на RHEL. Произошло это из-за несовпадения ID ОС.
Выполнение проверки XCCDF-правил с профилем xccdf_org.ssgproject.content_profile_cis на Rocky Linux 9.7 с автоматическим tailoring	Все статусы проверок XCCDF-правил вернут pass или fail	Из 375 статусов проверок XCCDF-правил 5 вернули not applicable, 167 вернули pass, 203 вернули fail.

Табл. 4. Функциональное тестирование

3.7 Перспективы развития проекта.

Разработанная программа ввиду своей модульной архитектуры как на стороне визуального интерфейса, так и на стороне функционала - позволяет быстро изменять и расширять

функционал продукта. Из основных направлений дальнейшего развития проекта можно выделить:

1. Расширение поддерживаемых дистрибутивов и источников SCAP-контента. Возможна поддержка большего числа дистрибутивов, а также возможно увеличение количества источников SCAP-контента.
2. Улучшение отчётности и экспорт результатов. Можно добавить экспорт отчётов в форматы PDF/HTML/JSON, фильтрацию и поиск по правилам.

ЗАКЛЮЧЕНИЕ

В ходе выполнения проекта была достигнута поставленная цель - создано программное средство автоматизированного аудита и харденига операционной системы на основе SCAP-контента (OVAL/XCCDF).

Для достижения цели были решены следующие задачи:

1. Реализован визуальный интерфейс программы, обеспечивающий запуск проверок по SCAP-контенту и просмотр результатов в удобочитаемом виде.
2. Разработан модуль выполнения проверок по OVAL-дефинициям, обеспечивающий интерпретацию условий и получение статусов результатов.
3. Разработан модуль выполнения проверок по XCCDF-контенту, позволяющий проводить оценку правил по выбранному профилю и формировать результаты проверки.
4. Реализован механизм автоматического tailoring, обеспечивающий адаптацию XCCDF-контента к целевой системе при несовпадении идентификаторов платформы.
5. Добавлена поддержка remediation для XCCDF, позволяющая применять корректирующие действия при наличии соответствующего контента.
6. Реализован модуль подгрузки SCAP-контента для выбранной операционной системы и подготовки контента к выполнению проверок.
7. Реализована агрегация результатов проверки из XML-файлов с последующим выводом в табличном виде.

Проект является перспективным для практического применения, поскольку опирается на открытые стандарты SCAP, обеспечивает воспроизводимость проверок и снижает зависимость от проприетарных решений. Модульная архитектура программы позволяет расширять функциональность: добавлять поддержку новых платформ и источников SCAP-контента, улучшать отчётность, а также развивать механизмы автоматизации харденига.

Вместе с тем текущая версия имеет ряд ограничений. В частности, после применения remediation не выделяются исправленные несоответствия, что снижает удобство подтверждения устранения несоответствий и может быть улучшено в дальнейшем. Также требуют развития возможности отчётности (экспорт в дополнительные форматы, расширенная фильтрация) и расширение набора поддерживаемых дистрибутивов/контента.

Таким образом, разработанное решение демонстрирует работоспособность и практическую применимость для задач регулярного аудита и харденига операционных систем, а обозначенные направления развития позволяют повысить уровень автоматизации и удобство эксплуатации в дальнейшем.

СПИСОК ЛИТЕРАТУРЫ

1. OpenSCAP User Manual [Электронный ресурс] // static.open-scap.org. — 2025. — URL: https://static.open-scap.org/openscap-1.3/oscap_user_manual.html
2. Аудит информационной безопасности. SCAP ликбез [Электронный ресурс] // habr.com. — 2021. — URL: https://habr.com/ru/articles/537974/
3. Введение в OVAL: Open vulnerability and Assessment Language [Электронный ресурс] // habr.com. — 2012. — URL: https://habr.com/ru/articles/136046/
4. Securing the open source supply chain: The essential role of CVEs [Электронный ресурс] // github.blog. — 2024. — URL: https://github.blog/security/supply-chain-security/securing-the-open-source-supply-chain-the-essential-role-of-cves/
5. Usage share of operating systems [Электронный ресурс] // en.wikipedia.org. — 2024. — URL: https://en.wikipedia.org/wiki/Usage_share_of_operating_systems
6. Российские сканеры уязвимостей: обзор технологий и решений [Электронный ресурс] // habr.com. — 2024. — URL: https://www.securitylab.ru/blog/personal/paragraph/354220.php
7. Obligations of the GPL and LGPL [Электронный ресурс] // habr.com. — 2021. — URL: https://www.qt.io/licensing/open-source-lgpl-obligations
8. Аудит информационной безопасности. XCCDF и OVAL [Электронный ресурс] // habr.com. — 2021. — URL: https://habr.com/ru/articles/538764/

ПРИЛОЖЕНИЕ

.

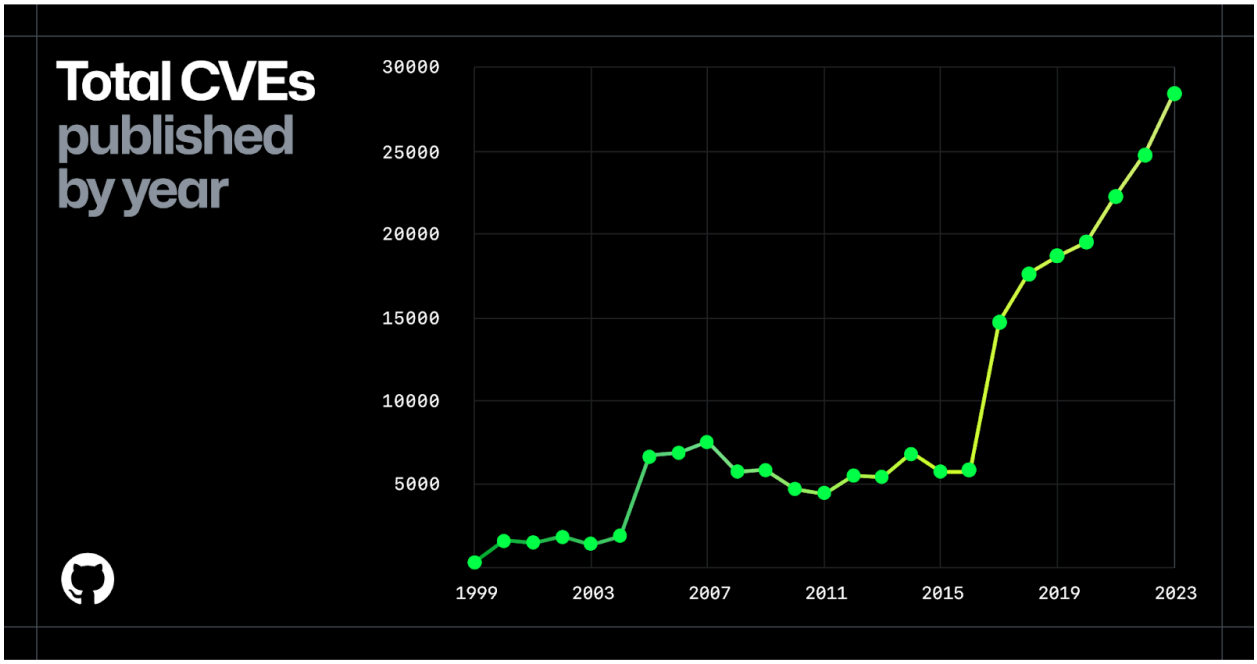

Рис. П1.1 График количества найденных уязвимостей по годам

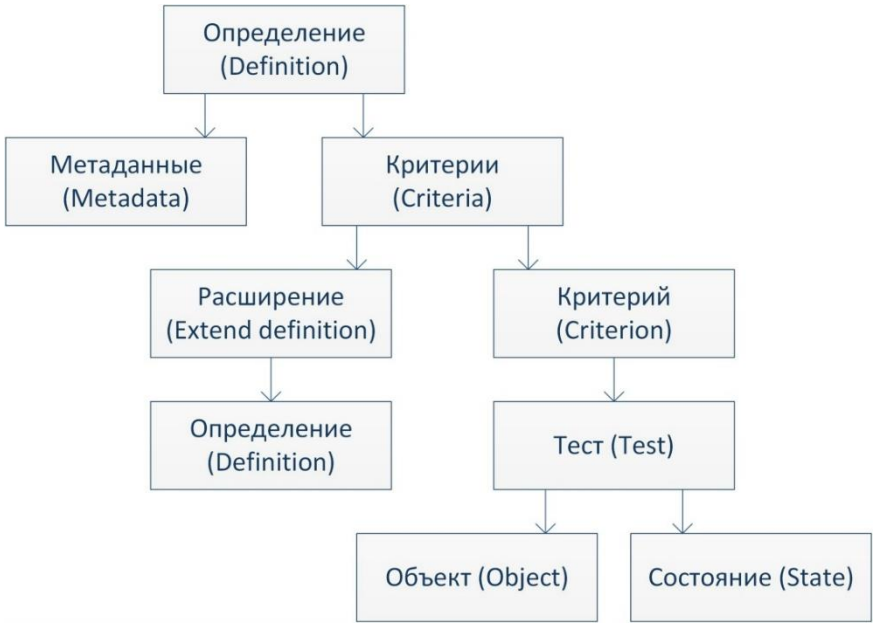

Рис. П1.2 Структура OVAL

Рис. П1.3 Структура XCCDF

Рис. П1.4 Процесс работы протокола SCAP

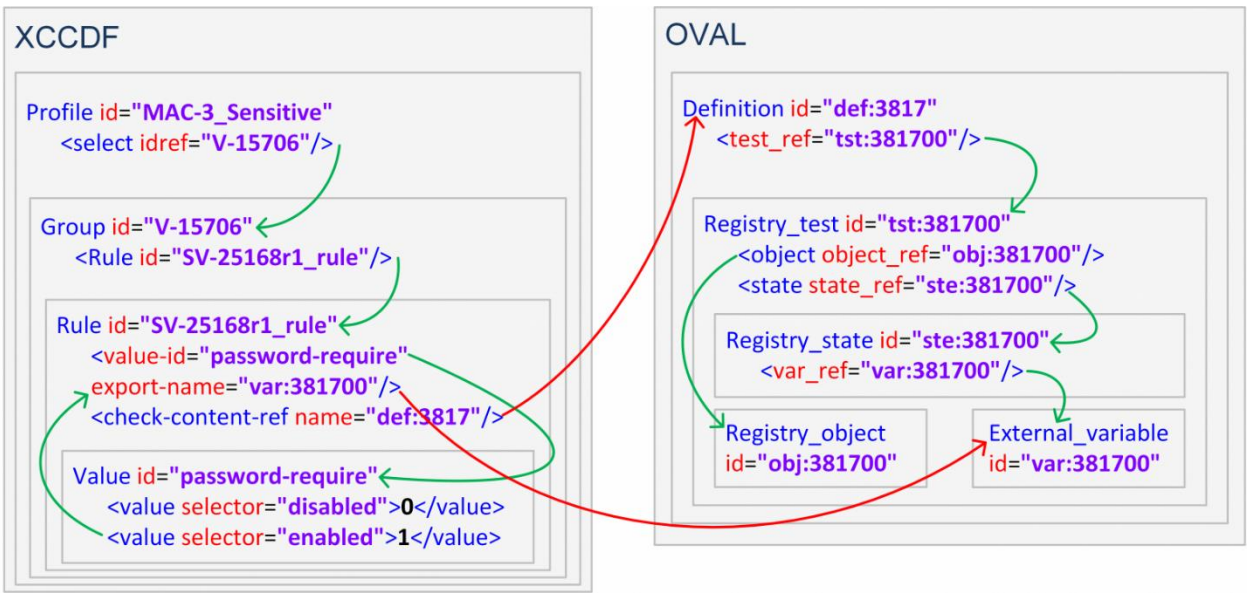

Рис. П1.5 Структура взаимодействия XCCDF и OVAL на примере CIS Benchmark

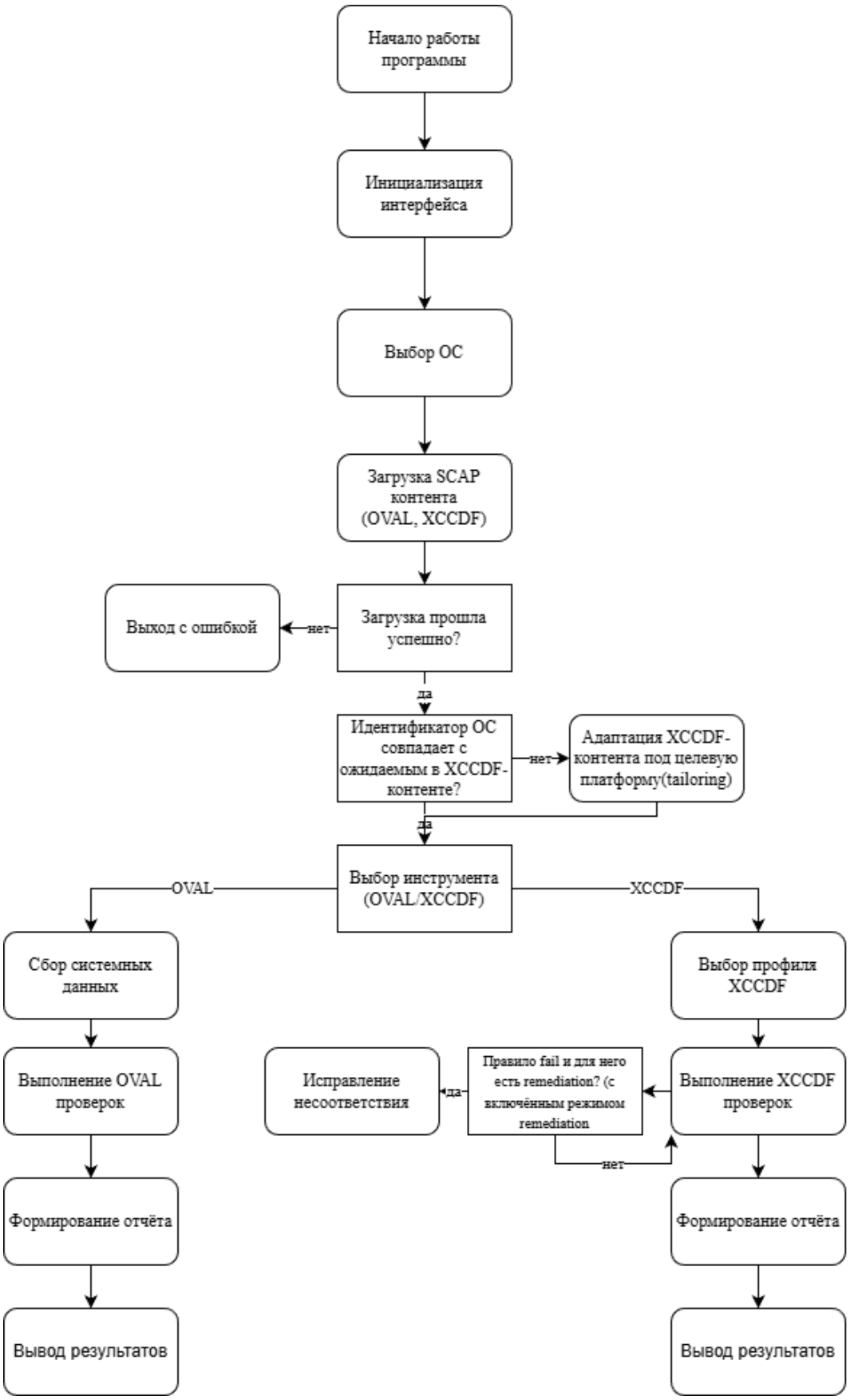

Рис. П2.1 Принцип работы программы

Инструмент	Поддержка SCAP	OpenSource	Цена	Remediation	Tailoring
Xspider	Нет	Нет	Платно	Нет	Нет
RedCheck	Да	Нет	Платно	Да	Нет
ScanOVAL	Частичная (только OVAL-контент от ФСТЭК)	Нет	Бесплатно	Нет	Нет
MaxPatrol VM	Нет	Нет	Платно	Да	Нет
Проектная программа	Да	Да	Бесплатно	Да	Да

Табл. 1. Сравнение с аналогами

Наименование фреймворка, иллюстрация	Поддерживаемые языки	Лицензия	Производительность	размер конечного приложения	Поддерживаемые платформы	Потребление оперативной памяти
 Electron js	JavaScript	MIT	Низкая	Большой	Windows, macOS, Linux	Высокое
 Flutter	Dart	BSD	Высокая	Средний-большой	Windows, macOS, Linux, Android, iOS, Web	Среднее
 GTK	20+ языков, включая c++ и python	LPGL	Средняя (зависит от ОС)	Низкий-средний	Linux (нативно), Windows и macOS.	Низкое-среднее (зависит от ОС)
 Qt	5 языков, включая c++ и python	LPGL, GPL	Высокая	Средний	Windows, macOS, Linux, Android, iOS	Низкое

Табл. 2. Сравнение фреймворков

Движок	Скорость проверки	Актуальность функционала	Возможность интеграции в программу	Поддерживаемые ОС
OpenSCAP	Высокая	Высокая	Средняя	Linux
SCC (SCAP Compliance Checker)	Низкая	Высокая	Низкая	Windows
Самодельный	Низкая	Низкая	Высокая	Linux/Windows
Ovaldi	Высокая	Низкая	Средняя	Linux, Windows

Табл. 3. Сравнение SCAP-интерпретаторов

Суть теста	Ожидаемый результат	Получаемый результат
Выполнение проверки OVAL-дефиниций на Debian 12	Все статусы проверок дефиниций вернут true или false	Все статусы проверок дефиниций содержали true или false. Проверка проверила 1428 дефиниц, 1279 из которых вернули false и 149 вернули true. (одна дефиниция проверяла 1 или сразу несколько CVE)
Выполнение проверки XCCDF-правил с профилем xccdf_org.ssgproject.content_profile_cis на Red Hat Linux Enterprise 9.6	Все статусы проверок XCCDF-правил вернут pass или fail	Все статусы проверок XCCDF-правил вернули pass или fail. Всего прошло тестирование 375 правил, 170 из которых вернули pass и 205 вернули fail
Выполнение проверки XCCDF-правил с профилем xccdf_org.ssgproject.content_profile_cis на Rocky Linux 9.7 без автоматического tailoring	Неизвестный результат	Все статусы проверок XCCDF-правил вернули not applicable, хоть Rocky Linux и основан на RHEL. Произошло это из-за несовпадения ID ОС.
Выполнение проверки XCCDF-правил с профилем xccdf_org.ssgproject.content_profile_cis на Rocky Linux 9.7 с автоматическим tailoring	Все статусы проверок XCCDF-правил вернут pass или fail	Из 375 статусов проверок XCCDF-правил 5 вернули not applicable, 167 вернули pass, 203 вернули fail.

Табл. 4. Функциональное тестирование