Разработка программного модуля защиты целостности скрипта ключом — символьным паролем

**Введение**

В условиях современной цифровой трансформации вопрос обеспечения информационной безопасности становится особенно актуальным. Одной из ключевых задач является защита целостности программного кода, особенно скриптов, которые могут быть подвержены изменениям как со стороны злоумышленников, так и по ошибке.

**Глава 1. Теоретические основы защиты целостности**

1.1 Понятие целостности данных  
 1.2 Угрозы целостности скриптов  
 1.3 Методы проверки целостности  
 1.4 Роль криптографии в обеспечении целостности

**Глава 2. Криптографические основы**

2.1 Хеш-функции и их свойства  
 2.2 Символьные пароли и генерация ключей  
 2.3 Алгоритмы HMAC и цифровой подписи

**Глава 3. Анализ существующих решений**

3.1 Обзор библиотек и инструментов  
 3.2 Сравнение по критериям безопасности, производительности, удобству  
 3.3 Выводы и выбор подходящего метода

**Глава 4. Проектирование программного модуля**

4.1 Постановка задачи и требования  
 4.2 Архитектура модуля  
 4.3 Структура хранения контрольных сумм  
 4.4 Выбор алгоритма: HMAC-SHA256 с символьным паролем

**Глава 5. Реализация**

5.1 Язык программирования и инструменты разработки  
 5.2 Генерация ключа из пароля  
 5.3 Вычисление и сохранение хеша скрипта  
 5.4 Сравнение хеша при выполнении  
 5.5 Обработка ошибок и отклонений

**Глава 6. Тестирование и результаты**

6.1 Методика тестирования  
 6.2 Тестовые данные и случаи  
 6.3 Результаты: производительность, устойчивость к изменениям

**Глава 7. Применение и перспективы**

7.1 Использование в реальных проектах  
 7.2 Ограничения метода  
 7.3 Возможности для дальнейшего развития: биометрические ключи, аппаратные токены

**Заключение**

Разработка программного модуля защиты целостности скрипта с использованием символьного пароля представляет собой эффективный способ предотвращения несанкционированных изменений в коде.

**Список литературы**

[1] Stallings, W. Cryptography and Network Security  
 [2] Schneier, B. Applied Cryptography  
 [3] RFC 2104 – HMAC: Keyed-Hashing for Message Authentication  
 [4] NIST FIPS PUB 180-4 – Secure Hash Standard  
 [5] Официальная документация Python hashlib и hmac

**Приложения**

A. Пример кода на Python  
 B. Конфигурационные файлы  
 C. Скрипты для тестирования

**Глава 1. Теоретические основы защиты целостности**

### **1.1 Понятие целостности данных**

Целостность данных — это одно из фундаментальных свойств информационной безопасности, означающее сохранность и неизменность информации в процессе её хранения, обработки и передачи. Суть этого свойства заключается в том, чтобы данные не были случайно или намеренно изменены, удалены или подменены без соответствующего разрешения.

Целостность информации является критически важной для обеспечения достоверности функционирования программных систем. Любое, даже незначительное изменение данных или кода может привести к сбоям в работе программ, утечке конфиденциальной информации или уязвимости перед атаками злоумышленников.

Целостность данных можно обеспечить как на уровне хранения (например, при сохранении файлов), так и на уровне передачи (например, при обмене данными между клиентом и сервером). В обоих случаях используются различные методы контроля целостности, такие как контрольные суммы, хеш-функции и цифровые подписи.

Поддержка целостности часто реализуется через:

* Вычисление и проверку хеш-сумм;
* Использование криптографических хешей (например, SHA-256);
* Применение HMAC (ключевых хешей) с секретными ключами;
* Контроль версий и журналирование изменений.

Если целостность нарушена, это может свидетельствовать об ошибке в системе, попытке взлома, вирусной активности или случайной порче данных. Поэтому контроль целостности часто используется как индикатор возможной атаки.

### **1.2 Угрозы целостности скриптов**

Скрипты, как легковесные исполняемые файлы (например, на языках Python, JavaScript, Bash), широко используются в автоматизации задач, веб-разработке, системном администрировании и других областях. Однако их легкость и доступность одновременно делают их уязвимыми для различных угроз, связанных с нарушением целостности.

**Основные угрозы целостности скриптов:**

#### **1. Внедрение вредоносного кода**

Злоумышленник может получить доступ к скрипту и внедрить вредоносный фрагмент — например, функцию отправки данных на внешний сервер или шифровальщик. Такие атаки часто осуществляются через уязвимости в правах доступа, заражённые внешние библиотеки или через подмену файлов в репозиториях.

#### **2. Случайное повреждение**

Изменения в скрипте могут произойти непреднамеренно, например, при ошибочной правке другим разработчиком, сбоем системы или повреждением файловой системы. Даже такие изменения могут привести к сбоям в логике работы, утрате функциональности или созданию лазеек для атак.

#### **3. Атаки типа Man-in-the-Middle (MITM)**

При передаче скриптов по незащищённым каналам, например, при клонировании из ненадёжного источника или скачивании через HTTP, возможна подмена файла во время передачи. Таким образом, атакующий может внедрить изменённую версию скрипта.

#### **4. Инсайдерские угрозы**

Сотрудник с доступом к исходному коду может преднамеренно изменить скрипт, добавив вредоносную функциональность. Это особенно опасно в корпоративной среде и требует строгого контроля целостности.

#### **5. Атаки через зависимости**

Скрипты часто используют внешние библиотеки и зависимости. Если одна из зависимостей скомпрометирована (например, опубликован вредоносный пакет в публичный репозиторий), это может привести к нарушению целостности конечного скрипта.

**Последствия нарушений целостности скриптов:**

* Потеря доверия к ПО;
* Утечка конфиденциальной информации;
* Введение бэкдоров;
* Нарушение работоспособности систем;
* Финансовые и репутационные потери.

### **1.3 Методы проверки целостности**

Для обеспечения и контроля целостности данных и скриптов используется ряд методов, основанных на криптографических и не криптографических подходах. Цель этих методов — своевременно обнаружить любые несанкционированные изменения содержимого скрипта.

#### **1. Контрольные суммы (Checksum)**

Контрольная сумма — это простейший способ проверки целостности. Алгоритм (например, CRC32) вычисляет числовое значение на основе содержимого файла. При передаче или хранении файла контрольная сумма сравнивается с первоначально сохранённой. Недостаток этого метода — низкая криптостойкость: злоумышленник может изменить файл и пересчитать контрольную сумму.

#### **2. Криптографические хеш-функции**

Криптографические хеш-функции (SHA-256, SHA-3 и др.) создают уникальный отпечаток (хеш) файла. При любом изменении даже одного байта, результат хеширования изменяется непредсказуемо. Такие хеши сложно подделать без знания содержимого файла. Это один из самых надёжных и широко используемых методов контроля целостности.

#### **3. HMAC (Hash-based Message Authentication Code)**

HMAC — это комбинация хеш-функции и секретного ключа. Она используется для проверки целостности и подлинности данных. Даже если хеш-функция известна, без ключа злоумышленник не сможет сгенерировать корректный HMAC. Это особенно эффективно для защиты скриптов от подмены при передаче по сети.

#### **4. Цифровые подписи**

Цифровая подпись использует пару ключей (приватный и публичный). Скрипт подписывается приватным ключом, а проверяется — публичным. Это позволяет не только подтвердить целостность, но и проверить авторство файла. Используются в системах обновлений, подписанных скриптах и приложениях с повышенными требованиями безопасности.

#### **5. Контроль версий и мониторинг изменений**

Использование систем контроля версий (Git, Mercurial и др.) позволяет отслеживать изменения в скриптах, сравнивать текущую версию с предыдущими, откатываться при необходимости. В сочетании с автоматизированным аудитом (например, с использованием CI/CD) это значительно повышает надёжность.

#### **6. Файловый мониторинг**

Инструменты мониторинга целостности файлов (например, Tripwire, AIDE) позволяют отслеживать любые изменения в исполняемых скриптах в реальном времени. Они создают базу хеш-сумм для защищаемых файлов и периодически сравнивают текущие значения с базой.

## **2. Проектирование программного модуля защиты целостности скрипта ключом-паролем**

Современные программные модули обеспечения информационной безопасности требуют не только надёжности, но и гибкости в применении. Одним из универсальных способов защиты целостности скриптов является использование символьного пароля (ключа), с которым осуществляется генерация и верификация хеш-суммы файла. В данном проекте рассматривается модуль, реализующий такую защиту с помощью HMAC и криптографических хеш-функций.

### **2.1 Выбор метода защиты**

Для обеспечения высокого уровня безопасности необходимо использовать криптографически стойкие методы. В качестве базовой технологии выбран метод HMAC (Hash-based Message Authentication Code), который использует хеш-функцию (например, SHA-256) в сочетании с символьным паролем (секретным ключом).

**Преимущества HMAC:**

* Защита от подделки хеша без знания ключа;
* Возможность проверки подлинности и целостности одновременно;
* Простота интеграции в любые скрипты.

**Принцип работы HMAC-модуля:**

1. Пользователь указывает пароль (ключ);
2. Модуль вычисляет HMAC от содержимого скрипта;
3. Результат сохраняется либо в отдельный файл, либо в конце скрипта в виде комментария;
4. При повторной проверке содержимое скрипта вновь хэшируется, и результат сравнивается с сохранённым значением.

**Выбор языка программирования:**Для реализации модуля выбран язык Python, как кроссплатформенный, удобный для работы со скриптами, и обладающий широким выбором криптографических библиотек (например, hashlib, hmac).

### **2.2 Архитектура программного модуля**

Архитектура программного модуля защиты целостности скрипта ориентирована на простоту интеграции и возможность автономного использования как в командной строке, так и в составе других систем. Модуль реализован с применением принципов модульности, расширяемости и безопасности.

#### **Основные компоненты модуля:**

1. **Компонент чтения скрипта**
   * Отвечает за загрузку содержимого указанного файла;
   * Выполняет фильтрацию встроенных хешей (если они присутствуют);
   * Учитывает кодировку и защищает от ошибок при чтении.
2. **Компонент генерации HMAC**
   * Получает символьный пароль от пользователя;
   * С помощью алгоритма HMAC-SHA256 создаёт хеш-значение;
   * Использует библиотеку hmac из Python или аналог на другом языке.
3. **Компонент верификации**
   * Сравнивает текущий HMAC с ранее сохранённым;
   * Показывает результат: «Целостность подтверждена» или «Файл изменён»;
   * Поддерживает вывод в лог или файл отчёта.
4. **Компонент сохранения/встраивания подписи**
   * Записывает вычисленную HMAC-строку в отдельный .sig файл, либо;
   * Добавляет её в конец скрипта как комментарий (например, # HMAC:<значение>).
5. **Компонент командного интерфейса (CLI)**
   * Предоставляет интерфейс с командами:
     + generate — для генерации подписи;
     + verify — для проверки целостности;
   * Поддерживает аргументы: имя файла, путь, пароль, режим работы.

#### **Структура проекта:**

hmac\_guard/

├── main.py # CLI-интерфейс

├── hasher.py # Логика HMAC и хеширования

├── file\_handler.py # Работа с файлами

├── verifier.py # Проверка подписи

└── utils.py # Вспомогательные функции

#### **Технологии:**

* Язык: Python 3.x;
* Библиотеки: hashlib, hmac, argparse, os.