Задание по информационной безопасности

Дурнов Алексей Николаевич

Московский физико-технический институт
Физтех-школа радиотехники и компьютерных технологий

Москва, 2024 г.

Настоящее руководство предназначено для разработчиков программного обеспечения (далее, ПО), написанного на языке программирования C++, и охватывает ПО разного назначения: встраиваемые системы, высоконагруженные системы и т.д. Настоящее руководство распространяется на весь жизненный цикл разработки, начиная с проектирования системы и заканчивая поддержкой готового продукта.

Цели организации в области создания безопасного ПО

Основными целями организации в области создания безопасного ПО являются:

1. Минимизация рисков, связанных с различными уязвимостями ПО,
2. Соответствие стандартам и требованиям безопасности,
3. Повышение качества кода,
4. Оптимизация затрат на поддержку и эксплуатацию продукта,
5. Улучшение репутации компании.

Перечень и описание мер по разработке безопасного ПО, подлежащих реализации в среде разработки ПО

- Проектирование и архитектура
  1. Разделяйте код на независимые модули, каждый из которых отвечает за свою конкретную задачу. Это сокращает время поиска и устранения уязвимостей.
  2. Проектируйте приложение таким образом, чтобы оно работало с минимально возможными правами доступа. Это уменьшает возможные последствия успешных атак.
  3. Определите и документируйте архитектуру безопасности вашего приложения, включая модели угроз и методы защиты.

Перечень и описание мер по разработке безопасного ПО, подлежащих реализации в среде разработки ПО

- Написание кода
  1. Работайте с последними стандартами языка, которые предоставляют множество возможностей для безопасного программирования.
  2. Используйте по возможности умные указатели для автоматического управление памятью, чтобы избегать утечек памяти и двойного освобождения.
  3. Применяйте идиому RAII (Resource Acquisition Is Initialization) для автоматического управления ресурсами, включая файлы, сокеты и другие объекты. Не работайте с примитивами синхронизации (std::mutex и другие) без RAII оберток.
  4. Используйте std::string вместо сырых строк типа char*. Это предотвращает многие проблемы, связанные с переполнением буфера.
  5. Всегда экранируйте и валидируйте пользовательские данные, особенно если они используются в SQL-запросах, HTML-коде и т.д.

Перечень и описание мер по разработке безопасного ПО, подлежащих реализации в среде разработки ПО

- Тестирование и анализ кода
  1. Проводите юнит-тестирование и функциональное тестирования для проверки корректности работы отдельных функций и модулей.
  2. Используйте инструменты статического анализа (например, cppcheck, svace, pvs-studio) для выявления потенциальных уязвимостей и ошибок.
  3. Используйте инструменты динамического анализа (например, valgrind, clang code sanitizers) для обнаружения уязвимостей и ошибок.
  4. Проводите фаззинг-тестирования для поиска сбоев программы.
  5. Проводите тестирование на проникновение для выявления уязвимостей в реальных условиях эксплуатации.
- Управление уязвимостями
  1. Быстро реагируйте на обнаруженные уязвимости, выпуская патчи и обновления.
  2. Следите за новыми уязвимостями в используемых библиотеках и компонентах. Регулярно обновляйте зависимости.
  3. Ведите журнал всех обнаруженных уязвимостей и их статуса (исправлено, отложено и т.д.).

Перечень и описание мер по разработке безопасного ПО, подлежащих реализации в среде разработки ПО

- Документирование и обучение
  1. Создавайте и поддерживайте документацию по безопасности, включая описания политик, процедур и рекомендуемых практик.
  2. Регулярно обучайте разработчиков основам безопасного программирования и новым технологиям.
  3. Периодически проводите внешние аудиты безопасности для независимой оценки качества безопасности вашего ПО.
- Инфраструктура
  1. Внедряйте логирование и мониторинг для отслеживания аномального поведения и попыток взлома.
  2. Регулярно делайте резервные копии данных и тестируйте процедуры восстановления.
- Сопровождение и поддержка
  1. Регулярно обновляйте операционную систему, библиотеки и другие компоненты для устранения известных уязвимостей.
  2. Обучайте пользователей правилам безопасного использования вашего ПО и помогайте им решать проблемы, связанные с безопасностью.

Перечень документации разработчика ПО, связанной с реализацией мер по разработке безопасного ПО

1. Описания технических требований к проекту.
2. Кодекс безопасности: внутренний документ, содержащий лучшие практики и стандарты по безопасной разработке ПО.
3. Отчеты по тестированию: результаты статических и динамических анализов кода, отчеты по фаззинг-тестированию, отчеты по тестированию на проникновение.
4. Журналы аудита: записи о проведенных аудитах безопасности и результаты проверок.
5. Политика информационной безопасности: документ описывает общие принципы и правила работы с информацией внутри организации, включая меры по защите данных, доступу к ресурсам и управлению инцидентами.

Описание действий, направленных на улучшение процессов, связанных с разработкой безопасного ПО

1. **Интеграция принципов безопасной разработки в культуру компании**.
   Проводите регулярные тренинги и семинары для разработчиков по вопросам информационной безопасности. Внедрите систему мотивации для разработчиков, которые активно участвуют в улучшении безопасности кода.
2. **Автоматизация процессов**.
   Интегрируйте инструменты статического анализа кода и тестирования безопасности в систему непрерывной интеграции. Настройте автоматическую проверку и обновление используемых библиотек и компонентов на наличие последних версий, содержащих исправления безопасности. Внедрите системы мониторинга и оповещения, которые будут уведомлять разработчиков и администраторов безопасности о возникновении новых уязвимостей или инцидентов.
3. **Применение лучших практик программирования**.
   Переходите на использование современных стандартов C++. Современные версии языка содержат встроенные механизмы защиты от уязвимостей.

- Библиотека «nlohmann/json» (также известная как «JSON for Modern C++») представляет собой высокоэффективную библиотеку для работы с JSON-форматом данных в языке программирования C++. Она позволяет легко сериализовать и десериализовать объекты в формат JSON и обратно.
- Основные особенности проекта:
  1. **Интуитивно понятный синтаксис**: с помощью перегрузки операторов удалось добиться удобного и интуитивно понятного интерфейса на ровне с другими высокоуровневыми языками.
  2. **Простота интеграции**: весь код состоит из одного заголовочного файла «json.hpp». У проекта нет ни внешних зависимостей, ни сложной системы сборки. Библиотека написана на C++11.
  3. **Высокая производительность**: библиотека оптимизирована для высокой производительности при работе с большими объемами данных, несмотря на простоту интерфейса.
  4. **Открытый исходный код**: проект имеет лицензию MIT, что делает его доступным для свободного использования в коммерческих и некоммерческих проектах.

- Проверка входных данных: неправильный формат данных и внедрение вредоносного кода.
- Защита от атак типа "отказ в обслуживании"(DoS): необходимо контролировать объем и сложность входных данных, устанавливая лимиты на количество уровней вложенности, длину строк и общее число элементов.
- Процесс преобразования объектов C++ в JSON и обратно должен быть защищен от возможных уязвимостей. **Десериализованные** объекты могут содержать конфиденциальную информацию, поэтому важно убедиться, что доступ к ним ограничен только авторизованным пользователям.
- Изоляция среды выполнения: среда выполнения не подвержена утечкам информации или несанкционированному доступу к ресурсам.
- Аудит кода и тестирование.

- Для моделировании угроз безопасности информации будет использоваться формальная модель STRIDE.
- Модель будет охватывать саму библиотеку и её взаимодействие с данными, которые она обрабатывает.
- Защищаемыми активами являются исходный код библиотеки, данные, обрабатываемые библиотекой, инфраструктура разработки и среда исполнения.

Классификация угроз по модели STRIDE:

- Подмена личности (Spoofing)
  1. Утечка учетных данных разработчиков: если учетные данные разработчиков будут украдены, злоумышленники смогут выдавать себя за легитимных участников проекта и вносить изменения в исходный код.
- Несанкционированное изменение (Tampering)
  1. Изменение исходного кода: злоумышленники могут попытаться внести изменения в исходный код библиотеки, добавив вредоносный код или удалив важные элементы безопасности.

- Отказ от ответственности (Repudiation)
  1. Отсутствие аудита: отсутствие должного аудита и журналирования действий может затруднить расследование инцидентов и установление виновных.
  2. Невозможность доказать подлинность изменений: без надлежащей системы контроля версий и управления изменениями невозможно точно установить, кто и когда вносил изменения в код.
- Раскрытие информации (Information Disclosure)
  1. Некорректная обработка данных: Некорректная обработка JSON-данных может привести к утечке конфиденциальной информации.
  2. SQL-инъекции и XSS (Cross-Site Scripting): обработки запросов к базам данных или отображение данных в веб-интерфейсе может привести к раскрытию информации.
- Отказ в обслуживании (Denial of Service)
  1. Переполнение буфера и утечки памяти: может привести к исчерпанию ресурсов и отказу в обслуживании.
- Повышение привилегий (Elevation of Privilege)
  1. Неправильное управление правами доступа: недостаточное разграничение прав доступа может позволить злоумышленникам получить больше полномочий, чем им положено.

Архитектурно проект организован следующим образом:

1. **Центральным классом** данной библиотеки является класс json: основной контейнер для хранения и обработки JSON-данных. Данный класс обеспечивает работу с различными типами данных: строками, числами, массивами, булевскими значениями и другими JSON-объектами.
2. Библиотека предлагает удобные механизмы для автоматической **сериализации** и **десериализации** стандартных типов данных C++. Также поддерживается возможность расширения функциональности для пользовательских типов через перегрузку специальных функций.
3. Поставляется в виде **одного заголовочного файла**. Данный вид поставки библиотеки упрощает интеграцию в проекты за счёт отсутствия отдельной компиляции и линковки библиотеки.
4. В репозитории содержится **обширная база тестов**. Документация также доступна в репозитории и включает в себя базовые примеры кода, руководство по использованию и справочную информацию по API библиотеки.

Используемые инструментальные средства

Ниже представлен список основных используемых инструментальные средств в проекте:

1. CMake – основной инструмент сборки проектов. Используется для создания кросс-платформенной среды разработки и сборки библиотеки. С помощью CMake можно генерировать разные системы сборки.
2. Mkdocs – инструмент для генерации документации из комментариев в исходном коде. Позволяет создавать HTML, LaTeX и PDF-документы с описанием классов, методов и функций.
3. GitHub Actions – система непрерывной интеграции, которая автоматически запускает тесты и проверки кода после изменения в проекте. Это помогает быстро выявлять ошибки и поддерживать высокое качество кода.
4. Doctest – один из основных фреймворков для тестирования кода на C++, используемый для написания юнит-тестов.
5. Clang-format – инструмент для автоматического форматирования кода.

Информация о прослеживаемости исходного кода программы к проекту архитектуры программы

В данной библиотеке простая структура проекта. Исходный код разбит на нескольких ключевых компонентов:

1. json.hpp – главный заголовочный файл, содержащий всю функциональность библиотеки. Этот файл включает определения основных классов и функций, таких как класс json, методы для сериализации и десериализации, а также вспомогательные классы и структуры.
2. detail – директория, содержащая реализацию внутренней логики библиотеки. Здесь находятся специализированные алгоритмы, утилиты и вспомогательные функции, которые не предназначены для прямого использования пользователями библиотеки.
3. tests – директория, содержащая тесты, написанными с использованием фреймворка doctest.
4. examples – директория, содержащая примеры использования библиотеки, демонстрирующие базовые сценарии работы с JSON-данными. Эти примеры помогают новым пользователям быстрее освоиться с функционалом библиотеки.

Порядок оформления исходного кода программы

- Порядок оформления исходного кода программы определен и закреплен с помощью конфигурационного файла clang-format, находящегося в корне репозитория.

Сведения о результатах проведения экспертизы исходного кода программы

- Сведений о результатах проведения внешних экспертных аудитов не содержится в репозитории, но можно отметить, что репозиторий имеет более 250 участников разработки (contributer).
- Библиотека придерживается рекомендациям безопасной разработки от Core Infrastructure Initiative (CII).

Сведения о результатах проведения функционального тестирования программы

- Библиотека имеет обширный набор тестов.
- Написаны они с использованием фреймворка doctest.
- Код библиотеки прошел тщательное модульное тестирование, имеет 100% покрытие кода, включая все исключительные ситуации.

Сведения о результатах проведения тестирования на проникновение

- Открытые сведения о результатах проведения тестирования на проникновения у библиотеки отсутствуют.
- Однако известно, что данная библиотека прошла проверку в Лаборатории Касперского и Apple.

Сведения о результатах проведения динамического анализа кода программы

- Для динамического анализа кода программы используются следующие инструменты: valgrind и clang code sanitizers.
- Данные инструменты помогают находить такого рода ошибки как утечки памяти, неопределенное поведение, использование неинициализированной памяти, состояние гонок, выходы за граница памяти и другого рода ошибок. В результате применений данных инструментов ошибок подобного рода не было выявлено.
- Помимо динамического анализа кода библиотеки используются также и статический анализ: cppcheck и прочие утилиты.
- Оба вида анализа дополняют друг друга для обеспечения безопасности библиотеки.

Сведения о результатах проведения фаззинг-тестирования программы

- Для фаззинг-тестирования в проекте используется Google OSS-Fuzz.
- Он дополнительно запускает фаззинг-тесты со всеми анализаторами в режиме реального времени.
- На данный момент корпус тестов насчитывает более миллиарда тестов.

Описание процедур отслеживания и исправления обнаруженных ошибок ПО и уязвимостей программы

- Данная библиотека является с открытым исходным кодом.
- Пользователи и разработчики могут сообщать об ошибках и уязвимостях несколькими способами:
  1. **Issue** - основной способ уведомления разработчиков об ошибках и проблемах.
  2. Если проблема **связана с безопасностью**, то рекомендуется использовать соответствующую функцию Github, чтобы сообщить о проблеме **приватно**. Это позволит команде разработчиков исправить уязвимость до публичного раскрытия информации.
- После получения отчёта команда разработчиков анализирует серьезность ошибки или уязвимости, оценивает последствия возможных атак для пользователей, устанавливает приоритет задачи в зависимости от ее влияния на стабильность и безопасность библиотеки.
- Создается патч, который проходит строгое код-ревью, чтобы гарантировать его корректность и отсутствие новых ошибок, а также проводится полное тестирование, включая регрессионное тестирование, чтобы убедиться, что исправление не вызывает новых проблем.

Описание процедуры поиска разработчиком ПО уязвимостей программы

- Процедура поиска уязвимостей в программном обеспечении требует систематического подхода и применения различных инструментов и методик.
- Основные шаги, которые разработчик может предпринять, для поиска уязвимостей следующие:
  1. Статический анализ кода
     Инструменты: cppcheck, clang-tidy, svace, pvs-studio.
  2. Динамический анализ
     Инструменты: valgrind, clang code sanitizers.
  3. Фаззинг
     Инструменты: AFL++, Google OSS-Fuzz.
  4. Тестирование на проникновение
     Инструменты: Burp Suite, OWASP ZAP.

Реализация и использование процедуры уникальной маркировки каждой версии ПО

- Используется **семантическое версионирование** и **система контроля версий Git**. Версия ПО согласно этому стандарту имеет вид major.minor.patch, где:
  1. major: увеличивается при добавлении несовместимых изменений;
  2. minor: увеличивается при добавлении новых функций, не нарушающих совместимость;
  3. patch: увеличивается при исправлении багов, не влияя на совместимость.
- Каждая новая версия библиотеки помечается **уникальным тегом в системе контроля версий Git**. Также каждой новой версии создается файл со **списком изменений**, внесенные в данную версию.
- Выпуск новой версии состоит из следующих шагов: завершается разработка и тестирования всех изменений, определяется номер новой версии продукта, генерируется пакет для распространения по необходимости, оформляется список изменений, и затем эти изменения публикуются на Github с тегом новой версии.

Использование системы управления конфигурацией ПО

- Система управления конфигурацией охватывает **систему контроля версий, построения и распространения**.
- Для данной библиотеки используется **система контроля версий Git**. Git позволяет разработчикам отслеживать изменения в кодовой базе, создавать ветки для параллельной разработки, объединять изменения и фиксировать состояние кода на определенных этапах.
- Для автоматизации процесса сборки и тестирования используется **система непрерывной интеграции, основанная на GitHub Actions**. Эта система позволяет автоматически запускать сборку и тестирование кода при каждом изменении в репозитории.

В данном проекте используются следующие меры:

1. **Управление доступом** на Github происходит с помощью ограничений прав членов команды. Разработчики имеют разные уровни доступа в зависимости от своей роли (читатель, участник, администратор). Это помогает предотвращать случайные или злонамеренные действия со стороны отдельных участников.
2. **Two-Factor Authentication (2FA)**: рекомендуется использовать двухфакторную аутентификацию для повышения безопасности учетных записей разработчиков.
3. **Шифрование данных** используется для SSH-ключей и для HTTPS. Это помогает обеспечить защиту передаваемых данных от прослушивания и несанкционированного доступа даже в случае компрометации инфраструктуры.
4. **Аудит действий** в GitHub: GitHub предоставляет подробные журналы аудита, которые регистрируют все действия, выполненные участниками проекта.