УДК 004.056.5 URL: https://ptsj.bmstu.ru/catalog/icec/insec/1072.html

АНАЛИЗ УЯЗВИМОСТЕЙ В ПРОГРАММИРОВАНИИ НА РАЗНЫХ ЯЗЫКАХ

А.М. Андреев andreevam1@student.bmstu.ru
М.А. Кузнецов kuznetsovma@student.bmstu.ru
В.Н. Савинчуков savinchukovvn@student.bmstu.ru
Ф.М. Садков sadkovfm@student.bmstu.ru
Е.В. Глинская glinskaya@bmstu.ru
SPIN-код: 5430-3023

МГТУ им. Н.Э. Баумана, Москва, Российская Федерация

В статье проанализированы уязвимости, характерные для языков программирования Python, C, C++ и Java, а также выявлены общие и специфические проблемы, связанные с их использованием. В рамках исследования рассмотрены типичные ошибки разработчиков, особенности реализации языков, которые способствуют возникновению уязвимостей, и методы их предотвращения. Актуальность темы обусловлена необходимостью обеспечить безопасность программных систем в условиях растущего числа кибератак и ужесточения требований к защите данных. Понимание природы уязвимостей и их источников в различных языках программирования позволяет не только минимизировать риски, но и разрабатывать более устойчивые и надежные приложения, что является важным шагом на пути к созданию безопасной цифровой среды.

Ключевые слова: уязвимости, языки программирования, программное обеспечение, кибербезопасность, кибератаки, безопасная цифровая среда

Введение. Сегодня программирование является неотъемлемой частью общества. Каждый день пользователям становятся доступны улучшения программного обеспечения для любой отрасли. Однако вместе с ростом сложности и функциональности программных систем увеличивается и количество уязвимостей, которые могут быть использованы преступниками для нарушения работы систем, кражи данных или выполнения несанкционированных действий. В связи с этим *анализ уязвимостей* в языках программирования становится одной из ключевых задач в области кибербезопасности и разработки программного обеспечения. Языки программирования Python, C, C++ и Java наиболее популярны для создания разнообразных приложений — от вебсервисов и мобильных приложений до системного программного обеспечения и высокопроизводительных вычислений. Каждый из этих языков обладает своими особенностями, которые влияют на возникновение уязвимостей.

 ISSN 2541-8009

Python. Язык Python заслуживает глубокого анализа, поскольку занимает лидирующие позиции в мире благодаря своей доступности для начинающих, ясному коду и широкому спектру применения в самых разнообразных областях. Этот язык, как и многие другие, не свободен от потенциальных уязвимостей, которые в руках хакеров могут стать инструментом для компрометации программного обеспечения. В рамках данной статьи мы подробно рассмотрим основные принципы Python, его применение в различных сферах, внутреннюю архитектуру, системы защиты и стратегии предотвращения известных уязвимостей.

Python — универсальный язык программирования, который охватывает широкий спектр отраслей, таких как веб-разработка, анализ данных и машинное обучение. В каждой из этих областей пользователь сталкивается с уникальными угрозами безопасности при использовании Python.

Существуют следующие угрозы безопасности в веб-разработке на Python:

– инъекционные атаки (SQL — структурированный язык запросов, XSS, CSRF): SQL-инъекции позволяют злоумышленникам манипулировать SQL-запросами для доступа к несанкционированным данным;

– XSS (Cross-Site Scripting — межсайтовый скриптинг) дает возможность вставлять вредоносный скрипт в веб-страницы, что может быть использовано для кражи данных пользователей;

– CSRF (Cross-Site Request Forgery — межсайтовая подделка запроса) позволяет злоумышленникам заставлять пользователей выполнять действия на веб-сайте без их ведома;

– недостатки аутентификаци и авторизации: использование слабых механизмов аутентификации, таких как ненадежные пароли или устаревшие протоколы, может привести к утечкам данных и несанкционированному доступу;

– уязвимости сторонних зависимостей: применение уязвимых версий сторонних библиотек (например, Django, Flask) может открыть путь к компрометации системы через известные уязвимости.

Перечисленные выше ключевые угрозы безопасности веб-приложений включают инъекционные атаки, уязвимости в механизмах аутентификации и авторизации, а также использование сторонних зависимостей, что в совокупности создает риски несанкционированного доступа, манипуляции данными и полной компрометации системы.

Рассмотрим наиболее часто встречаемые уязвимости языка Python в области анализа данных.

1. Десериализация[1] ненадежных данных [1]. При десериализации ненадежных данных возникают серьезные угрозы для приложения. Например, возможны перечисленные ниже нежелательные действия:

– удаленное выполнение кода (Remote code execution — RCE). Если в данных содержатся сериализованные объекты, это может привести к выполнению кода при десериализации, в результате чего злоумышленник получит возможность осуществить на сервере вредоносные действия;

– подмена данных. Злоумышленник может осуществить подмену сериализованных данных, в результате чего работа приложения изменится;

– отказ в обслуживании (Denial of Service — DoS), или DoS-атака. Некорректные или специально подобранные данные могут привести к возникновению бесконечного цикла, переполнению памяти и другим проблемам, которые могут вызвать закрытие приложения;

– нарушение целостности приложения. Десериализация может привести к созданию объектов, которые находятся в крайне странных состояниях, в результате чего логика работы приложения нарушится.

Использование модулей `pickle` или `marshal` [2] для десериализации данных из ненадежных источников может привести к выполнению произвольного кода. Использование модуля pickle реализуется следующим образом:

```
import pickle data =
                b"cos\nsystem\n(S'rm -rf /'\ntR." pickle.loads(data)
```

Следует избегать десериализации данных из ненадежных источников и применять безопасные форматы, такие как JSON (JavaScript Object Notation — обозначение объекта JavaScript). Данный тип десериализации безопасен, поскольку JSON не поддерживает выполнение кода и имеет ограниченный набор типов данных. Однако это не означает, что JSON полностью безопасен. Всегда следует проверять данные, использовать только проверенные источники и соблюдать меры предосторожности.

2. SQL-инъекции (SQL Injection) [3] — это один из самых распространенных и опасных типов атак на веб-приложения, которые работают с базами данных. Атака заключается во внедрении злоумышленником вредоносного SQL-кода в запросы к базе данных. Это может привести к краже данных, изменению или удалению информации, а также к полному захвату контроля над базой данных.

[1] *Десериализация* — процесс, при котором данные, сохраненные в виде потока байтов или строки, преобразуются в объекты в памяти компьютера. Это процесс, обратный *сериализации*, при которой состояние объекта преобразуется в поток байтов или строку для хранения или передачи.

 ISSN 2541-8009

Принцип работы SQL-инъекций заключается в следующем. Приложение формирует SQL-запросы динамически, используя пользовательский ввод без должной проверки или экранирования. Например:

```
SELECT * FROM users WHERE username =
                          'user_input' AND password = 'user_input';
```

Если пользователь введет строку ' OR '1'='1 в поле username, запрос примет вид

```
SELECT * FROM users WHERE username =
                         '' OR '1'='1' AND password = 'user_input';
```

Это условие всегда будет истинным ('1'='1') и злоумышленник получит доступ к данным.

В итоге злоумышленник может получить доступ к конфиденциальной информации, такой как пароли, e-mail, номера кредитных карт, атакующий может изменить или удалить данные в базе. Примеры SQL-инъекций:

```
SELECT * FROM users WHERE username =
                          'admin' --' AND password = 'any_password'
```

(обход аутентификации, вход в систему без пароля),

```
SELECT * FROM users WHERE id = 1 UNION SELECT * FROM sensitive_data
```

(получение всех данных из таблицы),

```
SELECT * FROM users WHERE id = 1; DROP TABLE users; --'
```

(удаление таблицы).

Существуют следующие методы защиты от SQL-инъекций:

– использование подготовленных выражений (Prepared Statements) на Python с системой SQLite:

```
cursor.execute("SELECT * FROM users WHERE username =
                        ? AND password = ?", (username, password))
```

– применение технологий объектно-реляционного отображения ORM (Object-Relational Mapping) на Python с библиотекой SQLAlchemy:

```
user = session.query(User).filter(User.username =
                     = username, User.password == password).first()
```

– экранирование специальных символов;

– валидация входных данных по формату (e-mail, числа);

– ограничение прав доступа базы данных;

– использование брандмауэра веб-приложений WAF (Web Application Firewall) для автоматической блокировки атак;

– регулярное обновление программного обеспечения и систем управления базами данных;

– логирование всех запросов к базе с последующим анализом на подозрительные действия.

Эти меры позволяют существенно снизить риски, связанные с SQL-инъекциями в веб-приложениях.

С. «Старость» языка С особо не повлияла на его популярность среди программистов. В силу своего возраста и ряда других особенностей данный язык имеет множество потенциальных уязвимостей.

С — очень мощный процедурный язык программирования, разработанный еще в 1970-е годы. Язык С применяют в программировании сетевых драйверов, интерпретаторов, компиляторов и др. Несмотря на то что С широко используется в различных системах, с ним все еще связано множество уязвимостей безопасности. В этой части статьи основное внимание уделяется обсуждению уязвимостей безопасности в С.

Существует мнение, что С — самый уязвимый из большинства языков. Около 50 % всех зарегистрированных уязвимостей за крайнее десятилетие были на счету С. Это означает, что любое приложение, основанное на С, имеет повышенный риск безопасности.

Уязвимости в языке С возникают по нескольким причинам [8]: отсутствие автоматической проверки границ (в отличие от некоторых высокоуровневых языков, С не имеет встроенных механизмов для защиты от переполнения буфера), работа с указателями (указатели предоставляют гибкость, но также увеличивают риск ошибок, связанных с доступом к памяти).

Рассмотрим основные уязвимости языка С, связанные с ними проблемы и пути их решения.

1. Связанные с буфером и памятью.

Проблема. Функция `gets( )` является частью стандартной библиотеки ввода-вывода языка С. Она не проверяет размер буфера, а вредоносный ввод может легко вызвать переполнение буфера [9]. Злоумышленник может предоставить большой фрагмент данных в качестве входных данных, а функция `gets( )` попытается сохранить все эти данные в буфере, не принимая во внимание размер буфера, что в конечном итоге приведет к переполнению буфера, что может привести к дальнейшему выполнению произвольного кода и утечке информации.

Решение. Чтобы устранить эту проблему с помощью функции `gets( )`, программисты могут использовать функцию `fgets( )`. Она ограничивает длину входных данных на основе размера буфера.

Аналогичная ситуация с функцией strcpy( ). Вместо нее лучше применять функцию strncpy( ).

2. Уязвимости выполнения команд.

 ISSN 2541-8009

Проблема. Рассмотрим уязвимость «удаленное выполнение кода» в упрощенном виде. В качестве примера приведем приложение, которое обрабатывает запросы, включая передачу команд в систему:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void procuest(const char *input) {
     char command[256];
     snprintf(command, sizeof(command), "ls %s", input);
     printf("Command run: %s\n", command);
system(command);
}
int main() {
char userInput[256];
     scanf("%255s", userInput);
     procuest(userInput);
     return 0;
}
```

Можно ввести стоку команд

```
; rm -rf --no-preserve-root /
```

В этом случае программа выполнит команду `ls`, а затем активируется команда `rm -rf --no-preserve-root /`, что приведет к удалению всех файлов на корневом уровне системы. Атака подобного типа может привести к выполнению злонамеренного кода на сервере или в системе и является серьезной угрозой для безопасности.

Чтобы предотвратить эти действия, нужно строго валидировать и экранировать пользовательский ввод перед использованием в операциях с системой. Избежать подобных уязвимостей помогут более безопасные альтернативы, такие как функции exec с аргументами.

Решение. Следует выполнить анализ полученных данных или taint analysis — анализ ввода пользователя для выявления типичных ситуаций, возникающих при использовании программой ввода пользователя без проверки.

3. Уязвимости формата строки.

Проблема. Спецификаторы формата могут дать злоумышленнику преимущество [10] в получении доступа для выполнения необходимого ему кода, утечки информации.

Приведем примеры параметров такого формата и покажем их последствия при некорректной реализации:

```
«%x» Чтение данных из стека
«%s» Чтение строки символов из памяти процесса
«%n» Запись целого числа в ячейки памяти процесса.
```

Далее представлен листинг программы на языке С, демонстрирующей вышеуказанную концепцию:

```
#include <stdio.h>
 int main()
{
     char str[6];
     sprintf(str, "%s",
         "VVVVVVVVVVVVVV");
     printf("%s", str);
}
```

В программе выше функция `sprintf` с `%s` может вызвать переполнение буфера, поскольку размер вывода равен 14, что больше размера полученного буфера, который имеет размер всего 6.

Решение. Следует убедиться, что всем функциям форматирования строк передается статическая строка, которой пользователь не может управлять. Функции с таким же типом уязвимости: `fprintf`, `printf`, `sprintf`, `snprintf`.

Описанные выше меры позволяют сделать использование языка С более безопасным.

C++. На протяжении последних 30 лет C++ — один из популярнейших и часто применяемых языков программирования в сфере разработки [4]. Его богатая функциональность и отличная производительность сделали его предпочтительным выбором для самых разных приложений. Согласно различным опросам специалистов в области информационных технологий, C++ обычно занимает высокие места в рейтингах популярности языков. Согласно индексу TIOBE, который измеряет относительную популярность языков программирования, в октябре 2023 г. C++ находился в первой пятерке самых популярных языков. Запросы, содержащие название этого языка, составляют около 8–10 % от общего числа поисковых запросов.

C++ — это язык, используемый программистами для разработки программного обеспечения. Области его применения включают создание операционных систем, прикладных программ всех видов, компьютерных/встраиваемых приложений, драйверов, высокопроизводительных серверов и 3D-игр на компьютерах.

C++ — компилируемый, статически типизированный язык программирования.

Сборка C++-приложения включает препроцессинг (обработка директив и макросов), компиляцию (перевод в ассемблерный код), ассемблирование (перевод в машинный код) и компоновку (объединение объектных файлов и библиотек в исполняемый файл) [5].

Защита языка устроена следующим образом. Существует два основных типа безопасности памяти: пространственная и временная.

 ISSN 2541-8009

Пространственная безопасность гарантирует, что программа будет работать корректно, даже если она попытается обратиться к памяти за пределами разрешенных границ. Это включает в себя обращение за границы массивов, неправомерный доступ к структурам и полям объединения, использование итераторов и т. д. Пространственную безопасность проще реализовать, например, в коде Chromium или при упаковке объектов в формат WebAssembly (WASM). Однако они требуют проверки границ массивов, что более затратно, чем работа без таких проверок, и может привести к некоторому снижению производительности.

Временная безопасность, со своей стороны, обеспечивает корректную работу программы во время обращения к памяти, которая уже недоступна. Примеры нарушений временной безопасности включают использование памяти после ее освобождения (Use-After-Free (UAF) — использование после освобождения), двойное освобождение памяти, доступ к неинициализированной памяти и использование объектов после их перемещения (Use-After-Move (UAM) — использование после перемещения). Проблемы с временной безопасностью часто проявляются в виде ошибок с типами данных.

Многие проблемы с C++связаны с неопределенным поведением (UB — undefined behavior), являющимся частью стандартной библиотеки.

Рассмотрим наиболее распространенные ошибки/проблемы языка C++. Невозможно полностью «исправить» C++ без кардинальных изменений в его основе. Вместо этого важно выявить и минимизировать наиболее распространенные и опасные типы UB, особенно в контексте использования веб-браузера Chromium [6].

Несмотря на мощь и гибкость C++, ручное управление памятью остается источником сложно обнаруживаемых ошибок. Для решения этой проблемы разрабатываются и внедряются различные решения:

1. Удаление / уменьшение числа «сырых»[2] указателей.

Проблема. Ручное управление временем жизни объектов оказалось слишком сложным даже для опытных инженеров, что приводит к ошибкам, связанным с использованием освобожденной памяти (UAF), а также к утечкам памяти.

Решение. Запрет на прямое использование сырых указателей, а также операторов new и delete в коде помогает избежать множества проблем, связанных с управлением памятью, таких как утечки памяти и ошибки доступа. Вместо этого рекомендуется использовать умные указатели или специальные реализации, такие как MiraclePtr [7].

[2] Это классические указатели в C/C++, которые напрямую хранят адрес памяти.

2. Аннотирование времени жизни.

Проблема. Время жизни в C++ неизвестно компилятору, его нельзя отследить с помощью статического анализатора.

Решение. В отдельных случаях можно объявить время жизни с помощью оператора [[clang::lifetimebound]]. Таким образом, мы сообщаем, что время жизни привязано к объекту. У атрибута множество ограничений. Он не является решением для обеспечения безопасности памяти, но может помочь в некоторых случаях.

3. Реализация автоматическим управлением памятью.

Проблема. Временная безопасность и корректность (UAF, утечки). Программа продолжает использовать указатель на память, которая была освобождена, что приводит к непредсказуемому поведению программы.

Решение. Нужно организовать подсчет ссылок (например, ARC-семантики (Automatic Reference Counting — система автоматического подсчета ссылок) и/или запуск сборщика мусора.

4. Внедрение анализа владения.

Проблема. Временная безопасность и корректность (UAF, утечки). Утечка памяти происходит, когда программа теряет возможность доступа к выделенной памяти, не освободив ее. Причина — указатель на память был перезаписан или потерян без вызова операторов delete или free.

Решение. Необходимо создать систему, в которой каждый объект имеет единственного «владельца». Изменять объект можно только с помощью операции перемещения (std::move). Также нужно внедрить правила заимствования, аналогичные тем, что применяются в языке программирования Rust. Это позволит избежать ситуации, когда одновременно существует несколько изменяемых ссылок на один и тот же объект, а также обеспечит доступ к объектам только в рамках их времени жизни. Решение плохо подходит для C++, но его все равно предлагают.

5. Определение всех стандартных моделей поведения библиотеки.

Проблема. В стандартной библиотеке множество ситуаций типа UB. Подразумевается отсутствие проверки границ (например, std::span::operator[] и валидности (std::optional::operator*). Доступность типа std::string_view может привести к проблемам, связанным с использованием после освобождения памяти (UAF), что вызывает беспокойство, особенно в новых версиях стандартной библиотеки. Стандартная библиотека C++ содержит множество ошибок типа UB, что затрудняет гарантирование отсутствия ошибок в программах, особенно при использовании новых функций. В связи с этим стоит рассмотреть альтернативные решения, такие как библиотека Abseil.

 ISSN 2541-8009

Решение. Следует внедрить «защищенный» режим (выбираемый на этапе компиляции) в реализацию стандартной библиотеки. Этот режим позволит сделать неопределенное поведение более предсказуемым и безопасным. Это достаточно простое решение для обеспечения безопасности памяти.

Java. Java — это типизированный объектно-ориентированный язык программирования, который был разработан компанией Sun Microsystems, а в настоящее время принадлежит компании Oracle. Одним из главных преимуществ языка является его кроссплатформенность, т. е. код, написанный на Java, может быть запущен независимо от типа программного обеспечения устройства. Это возможно благодаря байт-коду — набор указаний, которые выполняются на виртуальной машине JVM (Java Virtual Machine).

Согласно индексу TIOBE, Java занимает третье место по популярности. Число запросов, содержащих название этого языка, составляет 10,66 % от всех запросов. Сфера применения языка довольна широкая. На Java создают следующие программы:

– десктопные приложения;
– веб-приложения;
– приложения для Android;
–веб-сервера;
– сервера приложений.

Любая уязвимость языка может привести к взлому миллиардов устройств, поэтому нужно с должным вниманием относиться к контролю и устранению уязвимостей языка Java.

Рассмотрим основные уязвимости языка Java, связанные с ними проблемы и пути их решения.

Java дает возможность писать безопасный код. В языке предусмотрена строгая статическая типизация, которая позволяет разграничивать типы и не допускать неявных преобразований. Благодаря модификаторам доступа в коде программы можно разграничивать доступ к классам, полям и методам. С помощью автоматического управления памятью посредством инструмента Garbage Collector (сборщик «мусора», который проверяет, какие элементы программы больше недоступны в памяти) код программы освобождается от проблем, связанных с утечкой памяти.

Несмотря на применение объектно-ориентированного программирования и строгую типизацию число уязвимостей Java постоянно растет с каждым годом. Список CWE (Common Weakness Enumeration — перечень дефектов и уязвимостей, которые обнаружили в различных программных обеспечениях) постоянно пополняется новыми типами недостатков системы безопасно-

сти. На основе платформы mend.io [12] можно выделить следующие типы уязвимостей Java:

– CWE-502;
– CWE-200;
– CWE-20;
– CWE-79.

Уязвимость идентификатора CWE-502 (Deserialization of Untrusted Data — десериализация ненадежных данных) заключается в том, что приложение при преобразовании структуры данных из одного типа в другой (десериализация) должным образом не проверяет источник и валидность данных, что приводит к возможности для злоумышленника удаленно исполнять команды в скомпрометированной системе [11].

Суть уязвимости идентификатора CWE-200 (Exposure of Sensitive Information to an Unauthorized Actor — передача конфиденциальной информации неавторизованному субъекту) заключается в том, что продукт раскрывает конфиденциальную информацию субъекту, который явно не уполномочен иметь к ней доступ.

Уязвимость типа CWE-20 (Improper Input Validation — неправильная проверка введенных данных) заключается в том, что продукт получает входные данные, но не проверяет их или получает некорректный результат проверки, при этом входные данные обладают вредоносными свойствами, которые нуждаются в безопасной и правильной обработке.

Уязвимость типа CWE-79 (Cross-site Scripting — межсайтовый скриптинг) представляет собой уязвимость межсайтового скриптинга (XSS), которая популярна для многих программ.

В библиотеках Java также могут присутствовать уязвимости. К примеру, в 2021 г. уязвимости были обнаружены в части проекта Apache Logging Project — Log4j [13] (библиотека, которая помогает эффективно отслеживать выполнения программы и выявлять ошибки). Множество известных компаний, такие как Twitter, Amazon и Apple iCloud, используют данную библиотеку в своих продуктах. Уязвимости Log4Shell предоставляют возможность получить доступ к базе данных, LDAP-серверу (протокол, используемый для получения и хранения данных из каталога) и месседж-брокеру, а при наличии небезопасных строк кода библиотека сканирует сообщения пользователей и, если обнаруживает в них исходный код, исполняет его. Дальше злоумышленник может скомпрометировать программу загрузкой вредоносного кода на сервер.

Написанные на языке Java программы также можно скомпрометировать с помощью различных инъекций:

 ISSN 2541-8009

– SQL-инъекции (внедрение SQL-кода);

– инъекции XPath (использование вводимых пользователем данных для создания запроса XPath для доступа к XML-данным);

– командные инъекции (передача операционной системе посторонних команд);

– инъекции соединительной строки (добавление в строки подключения дополнительных параметров, которые помогают обойти систему аутентификации).

Для устранения этих уязвимостей или избежания компрометации продукта с помощью их был разработан ряд защитных мер:

– для защиты от десериализации небезопасных данных можно использовать функции signing/sealing языка программирования, чтобы гарантировать, что десериализованные данные не были испорчены. Например, код аутентификации сообщений на основе хэша (HMAC) можно использовать в качестве гарантии того, что данные не были изменены;

– использование проверенных библиотек или фреймворков помогает избежать уязвимости типа XSS;

– от уязвимости типа CWE-200 может помочь разделение системы на безопасные отсеки. Это позволит выделить «безопасные» области, где границы доверия могут быть однозначно определены.

Некоторые версии отдельных библиотек могут содержать уязвимости. Путем решения этой проблемы может быть обновление. К примеру, с устранением уязвимости Log4Shell разработчикам помогало обновление Log4j до версии 2.15 или Java до версии 11.

Архитектура и компоненты Java включают механизмы безопасности [14], которые могут помочь защитить от враждебного, некорректно работающего или небезопасного кода. Однако следование лучшим практикам безопасного кодирования по-прежнему необходимо. Рекомендации по безопасному оформлению кода можно найти на официальном сайте Oracle.

Заключение. Анализ безопасности используемых языков программирования Python, C, C++ и Java показал наличие общих и специфических проблем их использования. Некоторые уязвимости могут возникать вне зависимости от используемого языка программирования. Каждый из этих языков имеет свои особенности, которые влияют на возможность взлома программы. Несмотря на простоту и удобство использования языка Python, его динамическая типизация и использование сторонних библиотек создают риски для безопасности. В языках C и C++ из-за низкоуровневой архитектуры программы требуется уделять особое внимание управлению памятью, поскольку часто возникают уязвимости типа переполнения буфера или утечек памяти.

Java, в свою очередь, представляет собой более безопасный продукт благодаря высокой абстракции, но и у него есть свои плюсы и минусы: это управление зависимостей и виртуальной машиной JVM.

Результаты исследования свидетельствуют о том, что эти риски можно свести к минимуму, соблюдая стандарты программирования (строгая валидация входных данных; использование качественных инструментов статического и динамического анализа; установка обновления зависимостей и используемых библиотек). Для защиты и предотвращения угроз рекомендуется использовать опробованные методы.

Литература

[1] *CWE-502: Deserialization of Untrusted Data.* URL: https://cwe.mitre.org/data/definitions/502.html (accessed 15.03.2025).

[2] *Pickle — Python object serialization.* URL: https://docs.python.org/3/library/pickle.html (accessed 15.03.2025).

[3] URL: https://owasp.org/www-community/attacks/SQL_Injection (accessed 15.03.2025).

[4] Страуструп Б. *Язык программирования C++*. URL: http://8361.ru/6sem/books/Straustrup-Yazyk_programmirovaniya_c.pdf (accessed 15.03.2025).

[5] *Процесс компиляции программ на C++*. URL: https://habr.com/ru/articles/478124/ (accessed 15.03.2025).

[6] *Безопасность типов и ресурсов в современном C++*. URL: https://habr.com/ru/articles/749046/ (дата обращения 15.03.2025).

[7] *Safer Usage of C++*. URL: https://docs.google.com/document/d/e/2PACX-1vRZr-HJcYmf2Y76DhewaiJOhRNpjGHCxliAQTBhFxzv1QTae9o8mhBmDl32CRIuaWZLt5kVeH9e9jXv/pub (accessed 15.03.2025).

[8] Howard M., LeBlanc D. *Writing Secure Code*. Microsoft Press, 2003, 768 p.

[9] Dowd M., McDonald J., Schuh J. *The Art of Software Security Assessment*. Chapter 8. C String Handling, 2006.

[10] *Exploiting Format String Vulnerabilities*. URL: https://www.ida.liu.se/~TDDC90/literature/papers/teso-fs1-1.pdf (accessed 15.03.2025).

[11] *CWE*. URL: https://cwe.mitre.org/index.html (accessed 15.03.2025).

[12] URL: https://www.mend.io/most-secure-programming-languages (accessed 15.03.2025).

[13] URL: https://ru-hexlet-io.turbopages.org/ru.hexlet.io/s/blog/posts/java-uyazvimost (accessed 15.03.2025).

[14] *Java Security Resource Center*. URL: https://www.oracle.com/java/technologies/security.html (accessed 15.03.2025).

Поступила в редакцию 20.03.2025

 ISSN 2541-8009

Андреев Артём Максимович — студент кафедры «Информационная безопасность», МГТУ им. Н.Э. Баумана, Москва, Российская Федерация.

Кузнецов Михаил Александрович — студент кафедры «Информационная безопасность», МГТУ им. Н.Э. Баумана, Москва, Российская Федерация.

Савинчуков Владимир Николаевич — студент кафедры «Информационная безопасность», МГТУ им. Н.Э. Баумана, Москва, Российская Федерация.

Садков Филипп Маркович — студент кафедры «Информационная безопасность», МГТУ им. Н.Э. Баумана, Москва, Российская Федерация.

Глинская Елена Вячеславовна — старший преподаватель кафедры «Информационная безопасность», МГТУ им. Н.Э. Баумана, Москва, Российская Федерация.

Научный руководитель — Басараб Михаил Алексеевич, доктор физико-математических наук, заведующий кафедрой «Информационная безопасность», МГТУ им. Н.Э. Баумана, Москва, Российская Федерация.

Ссылку на эту статью просим оформлять следующим образом:
Андреев А.М., Кузнецов М.А., Савинчуков В.Н., Садков Ф.М., Глинская Е.В. Анализ уязвимостей в программировании на разных языках. *Политехнический молодежный журнал*, 2025, № 05 (100). URL: https://ptsj.bmstu.ru/catalog/icec/insec/1072.html

ANALYSIS OF VULNERABILITIES IN PROGRAMMING IN DIFFERENT LANGUAGES

A.M. Andreev andreevam1@student.bmstu.ru
M.A. Kuznetsov kuznetsovma@student.bmstu.ru
V.N. Savinchukov savinchukovvn@student.bmstu.ru
F.M. Sadkov sadkovfm@student.bmstu.ru
E.V. Glinskaya glinskaya@bmstu.ru
SPIN-code: 5430-3023

Bauman Moscow State Technical University, Moscow, Russian Federation

The purpose of this work is to analyze vulnerabilities specific to the Python, C, C++, and Java programming languages, as well as to identify common and specific problems associated with their use. The study will consider typical developer errors, language implementation features that contribute to the emergence of vulnerabilities, and methods for preventing them. The results of the work can be useful both for developers seeking to improve the security of their applications and for cybersecurity researchers involved in analyzing and eliminating vulnerabilities in software. The relevance of the topic is due to the need to ensure the security of software systems in the context of a growing number of cyberattacks and tightening requirements for data protection. Understanding the nature of vulnerabilities and their sources in various programming languages allows not only to minimize risks, but also to develop more stable and reliable applications, which is an important step towards creating a secure digital environment.

Keywords: vulnerabilities, programming languages, software, cybersecurity, cyberattacks, secure digital environment

Received 20.03.2025

Andreev A.M. — Student, Department of Information Security, Bauman Moscow State Technical University, Moscow, Russian Federation.

Kuznetsov M.A. — Student, Department of Information Security, Bauman Moscow State Technical University, Moscow, Russian Federation.

Savinchukov V.N. — Student, Department of Information Security, Bauman Moscow State Technical University, Moscow, Russian Federation.

Sadkov F.M. — Student, Department of Information Security, Bauman Moscow State Technical University, Moscow, Russian Federation.

Glinskaya E.V. — Senior Lecturer, Department of Information Security, Bauman Moscow State Technical University, Moscow, Russian Federation.

 ISSN 2541-8009

Scientific advisor — Basarab M.A., Dr. Phys. and Math. Sci., Head of the Department of Information Security, Bauman Moscow State Technical University, Moscow, Russian Federation.

Please cite this article in English as:
Andreev A.M., Kuznetsov M.A., Savinchukov V.N., Sadkov F.M., Glinskaya E.V. Analysis of vulnerabilities in programming in different languages. *Politekhnicheskiy molodezhnyy zhurnal*, 2025, no. 05 (100). (In Russ.). URL: https://ptsj.bmstu.ru/catalog/icec/insec/1072.html

