Защитное и безопасное программирование (Defensive and secure coding)

весна 2023

Защитное программирование

принцип разработки ПО, при котором разработчики пытаются учесть все возможные ошибки и сбои, максимально изолировать их и при возможности восстановить работоспособность программы в случае неполадок.

```
int main(int argc, char *argv[]){
    double a, b, c, discriminant, sqrtDiscriminant, x1,x2;
    ifstream in(argv[1]);
    in >> a >> b >> c;
    discriminant = b*b - 4*a*c;
    sqrtDiscriminant = sqrt(discriminant);
    x1=(-b + sqrtDiscriminant) / (2*a);
    x2=(-b - sqrtDiscriminant) / (2*a);
    cout << x1 << ' ' << x2 << endl;
    return 0;
}
```

```
int main(int argc, char *argv[]){
    double a, b, c, discriminant, sqrtDiscriminant, x1,x2;
    ifstream in(argv[1]);
    in >> a >> b >> c;
    discriminant = b*b - 4*a*c;
    sqrtDiscriminant = sqrt(discriminant);
```

```
    x1=(-b + sqrtDiscriminant) / (2*a);
    x2=(-b - sqrtDiscriminant) / (2*a);
    cout << x1 << ' ' << x2 << endl;
    return 0;
}
```

```
ifstream in(argv[1]);
if(!in) {
    cout << "No input file found!" << endl;
    return 0;
}
```

```
in >> a >> b >> c;
if (in.fail()) {
    cout << "Wrong input!" << endl;
    return 0;
}
```

```
if (a==0) {
    cout << "Linear equation, not quadratic!" << endl;
    return 0;
}
```

```
discriminant = b*b-4*a*c;
if (discriminant<0) {
    cout << "No roots!" << endl;
    return 0;
}
```

Другие потенциальные проблемы

Вычитание близких чисел

Сравнение вещественных чисел

Деление больших чисел на малые

Умножение больших чисел между собой

…

Принципы защитного программирования

1.Общее недоверие: для каждого модуля входные данные необходимо тщательно анализировать в предположении, что они могут быть ошибочными.

2.Немедленное обнаружение: каждая ошибка должна быть выявлена как можно раньше, что упрощает установление ее причины.

3.Изолирование ошибки: ошибки в одном модуле должны быть изолированы так, чтобы не допустить их влияние на другие модули.

Рекомендации

Проверять области значений переменных.

Контролировать правдоподобность значений переменных, которые не должны превышать некоторых констант или значений других переменных.

Вычисления следует производить так, чтобы минимизировать накопление погрешностей.

Контролировать итоги вычислений.

Рекомендации

Проверять коды возврата функций.

Производить проверки правильности выполнения операций ввода-вывода.

Обрабатывать исключения.

Проверять доступность внешних сущностей (БД, сервисы, ...)

Безопасное программирование

методика написания программ, устойчивых к атакам со стороны вредоносных программ и злоумышленников.

Примеры:

- Некорректно отформатированные аргументы
- Переполнение буфера
- Чрезмерное чтение буфера
- …

Инъекции кода

Функции, позволяющие выполнить системные вызовы:

system () в C++;

eval () в JavaScript;

exec () в PHP;

os.system () в Python

и так далее.

Инъекции кода

```
string input;

cout << "Your message: ";

getline(cin, input);

string s = "notify-send '" + input + "'";

const char *command = s.c_str();

system(command);
```

Инъекции кода

Если пользователь введёт строку:

Hello! ' & & rm -rf /* & & echo 'a

то итоговая команда, переданная системе на исполнение:

notify-send 'Hello! ' & & rm -rf /* & & echo 'a'

Инъекции кода

Как защититься?

Проверять все вводимые данные на соответствие области допустимых значений и выполнение требований.

SQL инъекции

уязвимость, позволяющая злоумышленнику вмешиваться в запросы, которые приложение делает к своей базе данных.

В результате реализации SQL-инъекции злыдень может получить доступ к данным, которые он в обычном режиме не может видеть.

Или изменить данные. Или удалить.

И так далее…

SQL инъекции

Например, если http запрос

https://host.org/products?category=Gifts

приводит к выполнению SQL запроса к базе данных

```
SELECT * FROM products WHERE category = 'Gifts' AND
released = 1
```

SQL инъекции

Если не выполняется проверка, запрос

https://host.org/products?category=Gifts'--

приведёт к выполнению SQL запроса

SELECT * FROM products WHERE category = 'Gifts'--' AND released = 1

-- = комментарий в SQL

SQL инъекции

Если не выполняется проверка, запрос

https://host.org/products?category=Gifts'+OR+1=1--

приведёт к выполнению SQL запроса

SELECT * FROM products WHERE category = 'Gifts' OR 1=1--' AND released = 1

SQL инъекции

Как защититься?

Проверять значения входных данных.

Параметризированные запросы.

```
SELECT * FROM products WHERE category = :request_category
AND released = 1
```

Принцип наименьших привилегий.

И другие…

Переполнение буфера

программа при записи данных в буфер , выходит за границы буфера и перезаписывает соседние ячейки памяти.

Пример эксплуатации: перезапись адреса возврата, чтобы он указывал на код, выбранный злоумышленником.

Эволюция безопасности в языках программирования

C - разработчик должен самостоятельно заботиться о выделении и освобождении памяти

Java - выделением и освобождением памяти занимается Garbage Collector

Rust - механизмы система владения (ownership) и система заимствования (borrowing) управляют памятью, предотвращают неправильное использование памяти и race condition

Состав кода

Код разрабатываемой системы состоит из написанного самостоятельно и используемого внешнего

- open source или
- коммерческого.

Ошибки и уязвимости open source

естественного происхождения

и злонамеренные целенаправленные

CVE-2022-23812

https://github.com/advisories/GHSA-97m3-w2cp-4xx6

ориентируется на внешний IP-адрес системы и удаляет данные (путем перезаписи файлов) только для пользователей из России и Беларуси

Dependabot

https://github.com/dependabot

инструмент, который проверяет ваши проекты на GitHub на наличие зависимостей, требующих обновления, ищет множество вещей, включая проблемы безопасности, совместимость и многое другое

Public profile
Account
Appearance
Accessibility
Notifications

Access
Billing and plans
Emails
Password and authentication
Sessions
SSH and GPG keys
Organizations
Moderation

Code, planning, and automation
Repositories
Codespaces
Packages
Copilot
Pages
Saved replies

Security
Code security and analysis

Code security and analysis

Security and analysis features help keep your repositories secure and updated. By enabling these features, you're granting us permission to perform read-only analysis on your repositories.

Private vulnerability reporting Beta

Allow your community to privately report potential security vulnerabilities to maintainers and repository owners. Learn more about private vulnerability reporting.

Disable all | Enable all

- [ ] **Automatically enable for new public repositories**

Dependency graph

Understand your dependencies.

Disable all | Enable all

- [ ] **Automatically enable for new private repositories**

Dependabot

Keep your dependencies secure and up-to-date. Learn more about Dependabot.

Dependabot alerts

Receive alerts for vulnerabilities that affect your dependencies and manually generate Dependabot pull requests to resolve these vulnerabilities. Configure alert notifications.

Disable all | Enable all

- [ ] **Automatically enable for new repositories**

Dependabot security updates

Allow Dependabot to open pull requests automatically to resolve Dependabot alerts.

Disable all | Enable all

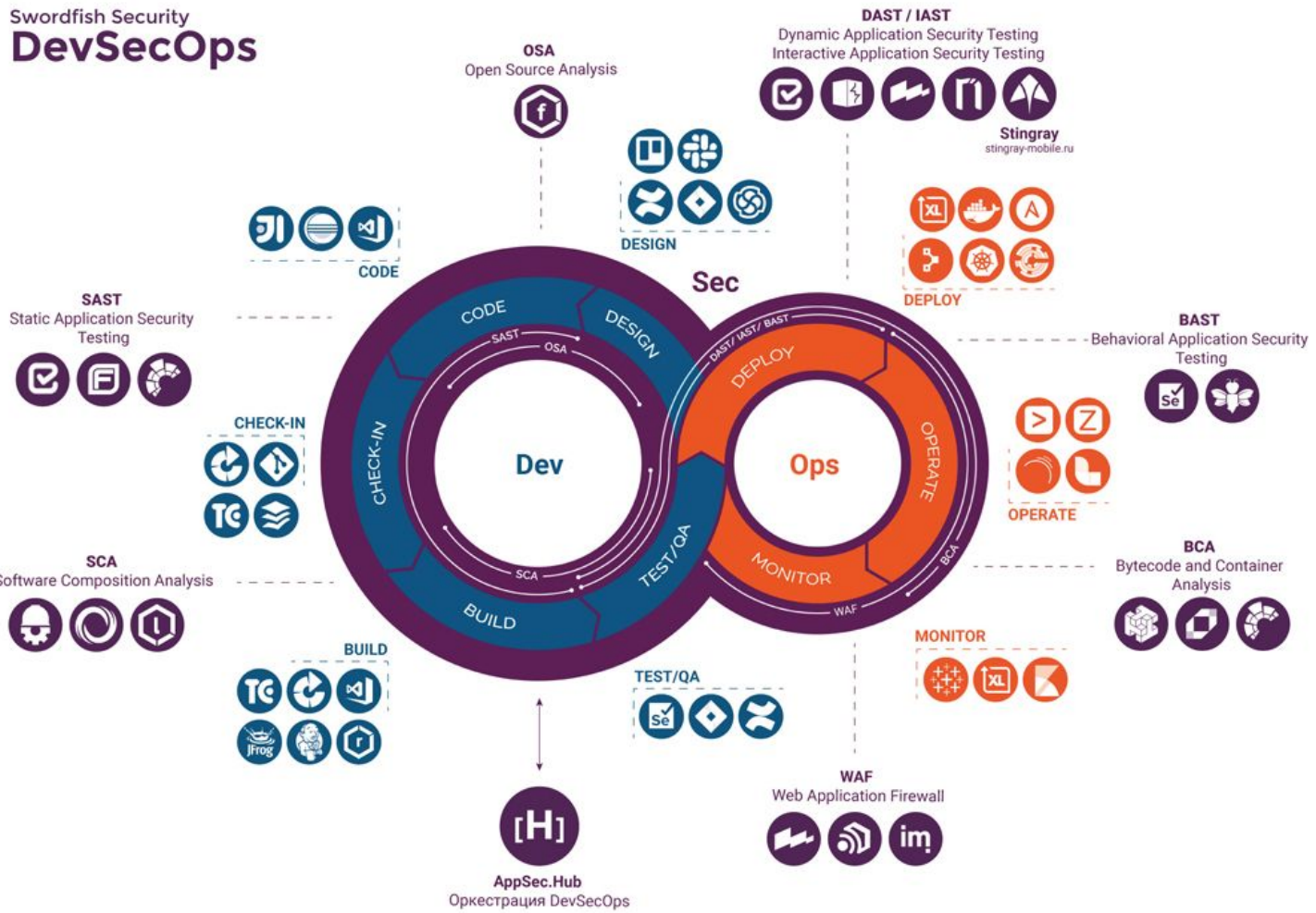
- [ ] **Automatically enable for new repositories**

Secret scanning

Receive alerts on GitHub for detected secrets, keys, or other tokens.

Disable all | Enable all

- [ ] **Automatically enable for new public repositories**


Swordfish Security
DevSecOps
OSA
Open Source Analysis
DAST / IAST
Dynamic Application Security Testing
Interactive Application Security Testing
Stingray
stingray-mobile.ru
DESIGN
CODE
DEPLOY
SAST
Static Application Security Testing
Sec
BAST
Behavioral Application Security Testing
CHECK-IN
Dev
Ops
OPERATE
SCA
Software Composition Analysis
BCA
Bytecode and Container Analysis
BUILD
MONITOR
TEST/QA
WAF
Web Application Firewall
AppSec.Hub
Оркестрация DevSecOps

Классы инструментов анализа кода

SAST (Static Application Security Testing) — статический анализ кода: проверка библиотек, поступающих в контур разработки.

OSA (Open Source Analysis) — анализ компонентов с открытым исходным кодом.

SCA (Software Composition Analysis) — анализ состава программного кода (из чего состоят программные системы).

Классы инструментов анализа кода

DAST (Dynamic Application Security Testing) — динамический анализ кода.

IAST (Interactive Application Security Testing) — интерактивный анализ кода.

BAST (Business (или Behavioral) Application Security Testing) — анализ кода бизнес-программ или поведенческий анализ кода.

Классы инструментов анализа кода

BCA (Bytecode and Container Analysis) — Анализ бинарного кода и контроль состава контейнеров.

WAF (Web Application Firewall) — межсетевой экран для приложений, осуществляющих передачу данных через HTTP и HTTPS.

Microsoft SDL

https://www.microsoft.com/en-us/securityengineering/sdl/practices

Security Development Lifecycle

(жизненный цикл безопасной разработки) концепция разработки, заключающаяся в обеспечении безопасности разрабатываемого приложения, идентификации рисков и управления ими.

Этапы SDL

Pre-SDL: Security Training — обучение информационной безопасности.

Requirements — анализ требований всех заинтересованных сторон. Здесь должны учитываться в том числе модели угроз как для корпоративной инфраструктуры, так и для разрабатываемых систем.

Этапы SDL

Design (планирование, проектирование) — преобразования требований в план для реализации их в приложении. Здесь должны учитываться требования безопасного проектирования.

Implementation (разработка ПО) — фокусирование на качестве и факте реализации ранее спроектированных требований в коде приложения. В этот этап также входит проверка зависимостей.

Этапы SDL

Verification — верификация, тестирование. При этом важно проводить тестирования в части аспектов информационной безопасности, в частности использовать проверки на наличие уязвимостей в коде.

Release — развёртывание и сопровождение, в которое входит мониторинг событий безопасности и реагирование на инциденты.

BSIMM

Building Security In Maturity Model

https://www.bsimm.com/

модель оценки зрелости процесса безопасной разработки

содержит более структурированное описание SSDL

BSIMM

1. Оценивать текущий уровень процессов безопасной разработки и находить слабые места
2. Знакомиться с best practices и понимать, к чему нужно стремиться
3. Отслеживать повышение уровня зрелости
4. Составлять дорожные карты и грамотно внедрять процессы безопасной разработки
5. Определять уровень своей эффективности относительно других компаний

Owasp SAMM

Open SAMM (Open Software Assurance Maturity Model)

https://www.opensamm.org/

https://owaspsamm.org/

модель обеспечения безопасности ПО, фреймворк базы знаний и документации, помогающий построить цикл разработки безопасных приложений

OWASP (Open Web Application Security Project)

https://owasp.org/

Открытый проект по обеспечению безопасности веб-приложений (OWASP) представляет собой некоммерческий, образовательный, благотворительный фонд, помогающий организациям начать проектировать, разрабатывать, приобретать, использовать и поддерживать безопасное ПО. Все инструменты, документы, форумы и отделения OWASP являются бесплатными и открытыми для тех, кто заинтересован в улучшении безопасности приложений.

ГОСТ Р 56939-2016

Защита информации. Разработка безопасного программного обеспечения. Общие требования.

Настоящий стандарт направлен на достижение целей, связанных с предотвращением появления и/или устранением уязвимостей программ, и содержит перечень мер, которые рекомендуется реализовать на соответствующих этапах жизненного цикла программного обеспечения.