МЕПСО

ПРИРАЧНИК

Прирачник за сајбер безбедност за ИТ програмери

Безбедно програмирање и заштита при развој на софтвер

Придружен материјал за обуката:
„Cybersecurity Awareness for IT Developers"

Документ бр. МЕПСО-ПРИ-ДЕВ-006 · Класификација: Интерна употреба
Сектор за информатичка и сајбер безбедност · МЕПСО АД Скопје

Содржина

1. Вовед

Софтверот што го создаваме е основа на дигиталната инфраструктура. Секоја линија код може да биде или заштита или ранливост. Безбедноста мора да биде дел од развојот од самиот почеток, не додаток на крајот.

Овој прирачник е наменет за програмери и развивачи. Ќе покриеме принципи на безбедно програмирање, најчести пропусти и практики за создавање сигурен софтвер. Целта е безбедноста да стане природен дел од вашиот развоен процес.

Како организација што управува со критична инфраструктура, квалитетот и безбедноста на софтверот што го создаваме и користиме имаат директни последици. Безбедното програмирање не е опција — тоа е професионална одговорност.

Клучен принцип. Безбедноста не може да се „додаде" на крајот. Таа мора да биде вградена во дизајнот и имплементацијата од самиот почеток. Поправањето на безбедносен пропуст во продукција е многукратно поскапо од спречувањето при развојот.

2. Безбедноста како дел од развојот

Безбедното програмирање значи размислување за безбедноста во секоја фаза на развојот.

2.1 Безбедност по дизајн

- ✓ Размислувајте за заканите уште при дизајнот
- ✓ Применувајте го принципот на најмала привилегија
- ✓ Претпоставете дека влезните податоци се недоверливи
- ✓ Планирајте за неуспех — како системот реагира на грешки
- ✓ Одбрана во длабочина — повеќе слоеви заштита

2.2 Начин на размислување

Добриот програмер размислува не само „како ова да работи", туку и „како ова може да се злоупотреби". Овој безбедносен начин на размислување е клучен за создавање сигурен софтвер.

Важно. Секој влез од корисник, секоја надворешна врска, секоја зависност е потенцијална точка на напад. Третирајте ги со соодветна претпазливост.

3. Најчести безбедносни пропусти

Разбирањето на честите пропусти е првиот чекор кон нивно спречување.

Пропуст	Ризик
Инјекција	Извршување малициозни команди преку влезни податоци
Слаба автентикација	Неовластен пристап до системот
Изложување на податоци	Откривање на чувствителни информации
Погрешна конфигурација	Безбедносни пропусти поради лоши поставки
Ранливи зависности	Наследени пропусти од надворешни компоненти
Недоволна валидација	Прифаќање на злонамерни влезни податоци

4. Валидација на влезни податоци

Никогаш не верувајте на влезните податоци. Секој влез — од корисник, датотека, мрежа или друг систем — мора да се валидира.

4.1 Принципи на валидација

- ✓ Валидирајте на серверска страна, не само на клиентска
- ✓ Применувајте „allow-list" пристап — дозволете само познато добро
- ✓ Проверувајте тип, должина, формат и опсег
- ✓ Отфрлете, не обидувајте се да „поправите" сомнителни податоци
- ✓ Валидирајте што поблиску до точката на внес

Златно правило. Секој влезен податок е потенцијално злонамерен додека не се докаже спротивното. Строгата валидација е првата линија на одбрана против многу видови напади.

5. Безбедно управување со лозинки и тајни

Лозинките, клучевите и другите тајни бараат посебна заштита во кодот.

5.1 Правила

- ✓ Никогаш не вградувајте тајни директно во кодот
- ✓ Не чувајте лозинки во обичен текст
- ✓ Користете безбедни методи за складирање на лозинки
- ✓ Чувајте ги тајните во заштитени конфигурациски системи
- ✓ Не ги вклучувајте тајните во системот за верзии

Честа грешка. Вградување на лозинки, API клучеви или други тајни директно во изворниот код. Ако кодот се сподели или процури, тајните се изложени. Секогаш користете безбедни методи за управување со тајни.

6. Автентикација и авторизација

Правилното управување со идентитет и пристап е основа на безбедниот софтвер.

6.1 Автентикација

- ✓ Користете докажани, стандардни механизми за автентикација
- ✓ Поддржувајте двофакторна автентикација каде е можно
- ✓ Заштитете против напади со погодување на лозинки
- ✓ Управувајте безбедно со сесиите

6.2 Авторизација

- ✓ Проверувајте ги дозволите за секое дејство
- ✓ Применувајте најмала привилегија
- ✓ Не се потпирајте само на криење на функции во интерфејсот
- ✓ Валидирајте ја авторизацијата на серверска страна

7. Заштита од инјекции

Инјекциите се меѓу најопасните пропусти — напаѓачот внесува малициозен код што системот го извршува.

7.1 Видови инјекции

- **SQL инјекција** — манипулација на бази на податоци;
- **Command инјекција** — извршување системски команди;
- **Cross-site scripting (XSS)** — инјектирање скрипти во веб-страници.

7.2 Заштита

- ✓ Користете параметризирани прашања за бази на податоци
- ✓ Валидирајте и „escape"-ирајте ги излезните податоци
- ✓ Не составувајте команди со неповерливи влезни податоци
- ✓ Применувајте соодветно кодирање на излезот

> **Клучно.** Никогаш не составувајте прашања или команди со директно спојување на влезни податоци. Секогаш користете безбедни, параметризирани методи.

8. Безбедна комуникација

- ✓ Користете шифрирана комуникација за чувствителни податоци
- ✓ Валидирајте ги сертификатите правилно
- ✓ Не пренесувајте чувствителни податоци во обичен текст
- ✓ Применувајте безбедни протоколи, избегнувајте застарени

9. Управување со зависности

Модерниот софтвер користи многу надворешни библиотеки. Секоја зависност носи потенцијални ранливости.

- ✓ Користете само доверливи, одржувани библиотеки
- ✓ Редовно ажурирајте ги зависностите
- ✓ Следете за објавени пропусти во користените библиотеки
- ✓ Минимизирајте го бројот на зависности
- ✓ Проверувајте ги зависностите пред вклучување

> **Ризик од зависности.** Ранливост во надворешна библиотека станува ранливост во вашиот софтвер. Одржувањето на зависностите ажурирани и следењето на безбедносните објави е клучно.

10. Управување со грешки и логирање

10.1 Безбедно управување со грешки

- ✓ Не откривајте чувствителни детали во пораките за грешки
- ✓ Логирајте ги грешките за анализа, но безбедно

- ✓ Не изложувајте внатрешна структура на корисниците
- ✓ Ракувајте со грешките контролирано

10.2 Безбедно логирање

- ✓ Не логирајте чувствителни податоци (лозинки, лични податоци)
- ✓ Логирајте безбедносно релевантни настани
- ✓ Заштитете ги логовите од неовластен пристап

Внимание. Деталните пораки за грешки корисни за дебагирање можат да откријат чувствителни информации на напаѓачите. Во продукција, прикажувајте општи пораки, а деталите чувајте ги во заштитени логови.

11. Безбедно чување на код

- ✓ Не вклучувајте тајни во системот за верзии
- ✓ Контролирајте го пристапот до изворниот код
- ✓ Прегледувајте го кодот за безбедносни пропусти
- ✓ Користете безбедни практики за соработка

12. Тестирање на безбедноста

Безбедносното тестирање открива пропусти пред напаѓачите.

- ✓ Вклучете безбедносни проверки во развојниот процес
- ✓ Тестирајте ги влезните податоци со неочекувани вредности
- ✓ Прегледувајте го кодот за безбедносни проблеми
- ✓ Користете алатки за автоматска анализа
- ✓ Тестирајте ја автентикацијата и авторизацијата

Практични сценарија

Сценарио 1: Кориснички внес во база

Развивате функција што прифаќа корисничко пребарување и го користи за прашање до базата на податоци.

> **Правилен пристап:** Користите параметризирани прашања, никогаш не спојувате директно корисничкиот внес во SQL. Валидирате го внесот пред употреба. Ова спречува SQL инјекција.

Сценарио 2: Складирање на API клуч

Апликацијата треба да се поврзе со надворешен сервис користејќи API клуч.

> **Правилен пристап:** Не го вградувате клучот во кодот. Го чувате во заштитен конфигурациски систем или променлива на околината, надвор од системот за верзии. Така клучот останува заштитен дури и ако кодот се сподели.

Сценарио 3: Прикажување на грешка

Апликацијата наидува на грешка во базата на податоци при обработка на барање.

> **Правилен пристап:** На корисникот прикажувате општа порака. Деталите за грешката (кои можат да откријат внатрешна структура) ги логирате безбедно за анализа. Не изложувате технички детали што напаѓач може да ги искористи.

Најчесто поставувани прашања

Зошто клиентската валидација не е доволна?

> Затоа што напаѓачите можат да ја заобиколат. Клиентската валидација е за корисничко искуство; безбедносната валидација мора да биде на серверска страна каде напаѓачот не може да ја манипулира.

Дали е безбедно да чувам лозинки ако се шифрирани?

> Лозинките треба да се чуваат со безбедни, еднонасочни методи специјално дизајнирани за лозинки, не со обично шифрирање. Никогаш во обичен текст.

Колку често треба да ажурирам зависности?

> Редовно, и веднаш кога е објавен безбедносен пропуст во користена библиотека. Следете ги безбедносните објави за вашите зависности.

Што е најважниот безбедносен принцип во програмирањето?

> Никогаш не верувајте на влезни податоци и вградете ја безбедноста од почеток. Овие два принципа спречуваат голем дел од честите пропусти.

Дали безбедносното тестирање го забавува развојот?

> Краткорочно бара напор, но долгорочно штеди многу. Поправањето на пропуст во продукција е многукратно поскапо од спречувањето при развојот.

Речник на поими

Поим	Значење
Инјекција	Внесување малициозен код што системот го извршува
SQL инјекција	Инјекција насочена кон бази на податоци
XSS	Cross-site scripting — инјектирање скрипти во веб-страници
Валидација	Проверка дека влезните податоци се исправни и безбедни
Автентикација	Потврдување на идентитетот
Авторизација	Определување на дозволените дејства
Параметризирано прашање	Безбеден начин на прашање до база без инјекција
Зависност	Надворешна библиотека што ја користи софтверот
Најмала привилегија	Доделување само на неопходни права

Безбедносна чек-листа за развивачи

При дизајн

- ✓ Размислив за потенцијалните закани

- ✓ Применив принцип на најмала привилегија
- ✓ Планирав за безбедно ракување со грешки

При имплементација

- ✓ Валидирам сите влезни податоци на серверска страна
- ✓ Користам параметризирани прашања за бази
- ✓ Не вградувам тајни во кодот
- ✓ Применувам безбедно управување со лозинки
- ✓ Не изложувам чувствителни детали во грешки

Пред продукција

- ✓ Прегледав го кодот за безбедносни пропусти
- ✓ Ажурирав ги зависностите
- ✓ Тестирав со неочекувани влезни податоци
- ✓ Проверив автентикација и авторизација
- ✓ Не вклучив тајни во системот за верзии

> **Навика над се.** Кога безбедносните проверки станат природен дел од вашиот работен тек, тие не забавуваат — тие стануваат автоматски. Најдобрите програмери размислуваат за безбедноста без да мора да се потсетуваат.

Заклучок

Безбедното програмирање е професионална одговорност. Секоја линија код што ја пишувате може да го зајакне или ослаби софтверот. Со вградување на безбедносни принципи во вашиот развоен процес, создавате сигурен софтвер што ја заштитува организацијата и корисниците.

> **Клучна порака.** Безбедноста не е нечија друга работа — таа е одговорност на секој програмер. Размислувајте за безбедноста од првата линија код до продукцијата, и вашиот софтвер ќе биде дел од решението, не од проблемот.

МЕПСО АД Скопје — Оператор на електропреносниот систем на Северна Македонија
Документ бр. МЕПСО-ПРИ-ДЕВ-006 · Класификација: Интерна употреба