Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра інформатики

Курсова робота

освітній ступінь – бакалавр

на тему: «**Фазинг тестування веб-застосунків на основі попередньо зібраних запитів або описаних структур прикладних програмних інтерфейсів**»

Виконав: студент 3-го року навчання,

Спеціальності

121 «Інженерія Програмного Забезпечення»

Студента Синицина Владислава

Керівник Бабич Т.А.

магістр комп’ютерних наук, асистент

«11» травня 2021 р.

Київ – 2021

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра інформатики

Освітній ступінь бакалавр

Спеціальність 121 «Інженерія Програмного Забезпечення»

Освітня програма бакалавр

ЗАТВЕРДЖУЮ

Завідувач кафедри інформатики

Гороховський С. С.

“10” жовтня 2020 року

ЗАВДАННЯ
ДЛЯ КУРСОВОЇ РОБОТИ СТУДЕНТУ

Синицину Власдиславу

1. Тема роботи **«Фазинг тестування веб-застосунків на основі попередньо зібраних запитів або описаних структур прикладних програмних інтерфейсів»**, керівник роботи Бабич Трохим Анатолійович, магістр комп’ютерних наук, асистент

2. Строк подання студентом роботи 17 травня 2021

3. План роботи

    Анотація

    Вступ

    Розділ 1. Дослідження та аналіз предметної області

        1.1 Потокова трансляція відео
        1.2

    Розділ 2. Проектування та розробка системи

2.1 Опис складових системи та використаних технологій для розробки

Висновки

Список використаних джерел

ГРАФІК ПІДГОТОВКИ КУРСОВОЇ РОБОТИ ДО ЗАХИСТУ

№ з/п	ПЕРЕЛІК РОБІТ	Термін виконання	Дата ознайомлення наукового керівника	Підпис наукового керівника	Примітки
1.	Вибір теми, затвердження її на засіданні кафедри та закріплення наукового керівника Узгодження календарного графіка підготовки кваліфікаційної роботи. Ознайомлення студента з критеріями оцінювання кваліфікаційної роботи (п. 8.5).	10 жовтня 2020			
2.	Вивчення джерел літератури, матеріалів архівів, періодичних видань, збір та узагальнення фактів, даних	10 жовтеня 2020 – 2 листопада 2020			
3.	Складання плану каліф. роботи та узгодження з науковим керівником	2 листопада 2020			
4.	Написання розділів роботи	2 листопада 2020 – 01 березня 2021			
5.	Проміжний контроль виконання роботи	01 лютого 2021			
6.	Написання кваліфікаційної роботи в цілому, ознайомлення з її першим варіантом наукового керівника	11 січня 2021 – 29 березня 2021			
	Розділ 1 (постановка проблеми, теоретичні основи, огляд літературних джерел)	25 січня 2021			
	Розділ 2 (аналітично-дослідницька частина)	01 березня 2021			
	Розділ 3 (проектно-рекомендаційна частина)	29 березня 2021			
7.	Повне завершення написання кваліфікаційної роботи, оформлення її згідно з вимогами й подання на відгук науковому керівнику	01 квітня 2021 – 06 травня 2021			
8.	Подання кваліфікаційної роботи для перевірки письмових робіт студентів НаУКМА на відповідність вимогам академічної доброчесності,	17 травня 2021			
9.	Публічний захист кваліфікаційної роботи перед екзаменаційною комісією	згідно з розкладом роботи ЕК			

Графік узгоджено 10 жовтня 2020 р.
Науковий керівник Бабич Трохим Анатолійович
Виконавець курсової роботи Синицин Владислав

ЗМІСТ

Анотація

У даній роботі розглядаються особливості тестування програмного забезпечення; найбільш поширені та часто використовувані вразливості web застосунків. Розглядається Фазинг метод тестування веб-застосунків на основі попередньо зібраних запитів або описаних структур прикладних програмних інтерфейсів. На прикладі фазинг-застосунку розглядається принцип дії останнього, та описуються особливості його розробки.

Вступ

Розділ 1. Дослідження та аналіз предметної області

1.1. Методики тестування програмного забезпечення

Тестування програмного забезпечення – це будь-яка діяльність, спрямована на оцінку властивостей або можливостей програми чи системи та визначення відповідності її необхідним результатам. [1] Метою тестування може бути забезпечення якості, валідація та верифікація або оцінка надійності. Тестування також може бути використано як загальний показник. Тестування на правильність та надійність - два основні напрямки тестування. Тестування програмного забезпечення - це компроміс між бюджетом, часом та якістю. [2]

Оскільки якість – це суб’єктивне, а не абсолютне поняття, то тестування не може повністю гарантувати коректну роботу програмного забезпечення. Воно тільки порівнює реальну поведінку програми з очікуваною, описаною в специфікації продукту. Також варто розрізняти тестування власне програмного забезпечення та забезпечення якості програмного забезпечення, до якого належать всі етапи розробки, починаючи з проектування, а не тільки власне тестування.

Поняття якості, зазвичай, передбачають у таких логічних категоріях, як коректність, надійність, практичність, безпечність, але можуть додавати більше технічних вимог, описаних у стандарті ISO 9126. Для визначення складу та змісту супутньої до процесу тестування документації, існує стандарт IEEE 829—1998 Standard for Software Test Documentation.

Тестування програмного забезпечення — процес перевірки відповідності заявлених до продукту вимог і реально реалізованої функціональності, здійснюваний шляхом спостереження за його роботою в штучно створених ситуаціях і на обмеженому наборі тестів, обраних певним чином.

Тестування програмного забезпечення передбачає виконання програмного компонента або системного компонента для оцінки однієї або декількох

властивостей, в перевірці яких зацікавлені. Загалом, ця оцінка вказує на те, наскільки компонент або система, що тестується:

- відповідає вимогам, якими керувалися проектувальники та розробники
- правильно відповідає на усі можливі вхідні дані
- виконує свої функцій за прийнятний час
- є практичною
- може бути встановлена та запущена на призначеній операційній системі
- відповідає задачам замовника.

Оскільки число можливих тестів навіть для нескладних програмних компонентів практично нескінченне, тому стратегія тестування полягає в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів. Як результат програмне забезпечення (ПЗ) тестується стандартним виконанням програми з метою виявлення помилок через несправності програми (багів) [3]

Тестування ПЗ може надавати об'єктивну, незалежну інформацію про якість програмного забезпечення та про ризики виходу його з ладу, як для користувачів, так і для замовників.[4]

Тестування може проводитись, як тільки створено виконуваний код (навіть частково завершений). Процес розробки зазвичай передбачає, коли та як буде відбуватися тестування. Наприклад, при поетапному процесі, більшість тестів відбувається після визначення системних вимог і тоді вони реалізуються в тестових програмах. На противагу цьому, відповідно до вимог гнучкої розробки ПЗ, програмування і тестування часто відбувається одночасно.

1.2. Основні поняття та визначення

Тестування — це одна з технік контролю якості, що включає в себе

- Планування робіт (Test Management)
- Проектування тестів (Test Design)
- Виконання тестування (Test Execution)
- Аналіз отриманих результатів (Test Analysis).

Верифікація (Verification) — це процес оцінки системи або її компонентів із метою визначити чи задовольняють результати поточного етапу розробки умовам, сформованим на початку цього етапу. Тобто чи виконуються цілі, терміни, завдання з розробки проекту, визначені на початку поточної фази. [6]

Валідація (Validation) — це визначення відповідності розроблюваного програмного забезпечення очікуванням і потребам користувача, вимогам до системи. [6]

План випробувань (Test Plan) - це документ, що детально описує підхід, який буде застосовано до передбачених тестових заходів. План може включати такі аспекти, як цілі, обсяг, процеси та процедури, стратегії, розклад, вимоги до персоналу, спеціального обладнання та плани на випадок надзвичайних ситуацій. [9]

Тест дизайн (Test Design) — це етап процесу тестування програмного забезпечення, на якому проектуються і створюються тестові випадки (тест кейси), відповідно до визначених раніше критеріями якості та цілями тестування.

Тестовий випадок (Тест кейс/Test Case), як правило, складається з унікального ідентифікатора, посилань на вимоги із специфікації проекту, передумов, подій, серії кроків (також відомих як дії), які слід виконати, введення, виведення, очікуваного результату та фактичного результату.

Клінічно визначено, тестовий випадок - це вхідні дані та очікуваний результат. [10]

Баг/Дефект Репорт (Bug Report) — це звіт, що описує ситуацію або послідовність дій (Steps), що призвела до некоректної роботи об'єкта тестування (Misbehavior), із зазначенням причин та очікуваного результату (Expected Result).

Тестове Покриття (Test Coverage) — це оцінка якості тестування, яка полягає в співвідношення кількості коду та тестів, які звертаються до цього коду задля тестування.

1.3. Статичне та динамічне тестування

До тестування програмного забезпечення існує багато підходів. Огляди, вичитка документації або перевірки називаються статичним тестуванням, тоді як виконання самого коду із заданим набором тестових випадків називається динамічним тестуванням.

Статичне тестування часто є неявним, наприклад, корекція документації. Статичним аналізом програми також є перевірка засобами програмування / текстовими редакторами структури вихідного коду або компіляторами (попередніми компіляторами) синтаксису та потоку даних. Динамічне тестування відбувається під час запуску самої програми. Динамічне тестування може розпочатися до завершення програми для тестування окремих розділів коду та застосування їх до дискретних функцій або модулів. [5][6]

Статичне тестування передбачає верифікацію, тоді як динамічне тестування також передбачає валідацію. [7]

1.3.1. Динамічне тестування

Динамічне тестування - це методика, спрямована на перевірку функціоналу програми, під час виконання коду. Тобто, даний тип тестування передбачає експлуатацію програми і визначення того, чи відповідає результат виконання певного функціоналу очікуваному результату.

Динамічний тип тестування складається з безпосереднього тестування програмного забезпечення в реальному часі, способом надання інформації на вхід і дослідження отриманого результату поведінки програми.

Даний метод тестування допомагає команді перевірити різні критичні моменти програмного забезпечення. Якщо закрити очі на їхнє існування і

ніяк не відреагувати на них, це може певним чином позначитися на продуктивності, функціональній стороні і надійності програмного забезпечення.

Переваги методу динамічного тестування:

- В процесі тестування проводиться ретельне вивчення всього функціоналу програми, і, як результат, отримуємо високу якість перевірки
- Динамічне тестування здійснює перевірку програмного забезпечення з боку кінцевого користувача, що допомагає значно підвищити якість програмного забезпечення.
- Фіксація комплексних багів (дефектів), які могли залишитися непоміченими.
- За допомогою спеціальних засобів динамічний тип тестування можна автоматизувати.

Недоліки методу динамічного тестування:

- Налагодження та впровадження динамічного тестування потребує багато ресурсів та часу.
- Динамічне тестування - дорогий процес.
- В основному, даний метод тестування застосовується під час розробки програмного забезпечення, а баги та дефекти виявляються в процесі життєвого циклу розробки.

1.3.2. Статичне тестування

Метод статичного тестування - це тип тестування програмного забезпечення, під час якого останнє перевіряється без запуску коду; є процесом або інструментом, спрямованим на виявлення можливих багів в програмному забезпечені. Крім цього, він знаходить і усуває помилки в

різного роду супровідних документах, наприклад, в специфікації вимог до програмного забезпечення.

Статичне тестування ділиться на два типи:

- Огляди
- Статичний аналіз

Огляди - тестування, спрямоване на виявлення дефектів в документації (вимоги, дизайнерське оформлення, тестові випадки тощо).

Статичний аналіз - аналіз на наявність прогалин у структурі, здатних привести до некоректного функціонування програмного забезпечення. Також застосовується для пошуку коду, що потенційно містить вразливості (іноді це застосування називається Static Application Security Testing, SAST). [8]

Під час статичного аналізу зазвичай перевіряється якість написаного розробниками коду. Для його проведення використовуються різні інструменти, такі як компілятори, лінкери, спеціальні утиліти.

За допомогою статичного аналізу можна виявити наступні дефекти:

- Змінні, які не використовуються
- Не ініційовані змінні
- Некоректний синтаксис
- Переповнення буфера
- Нескінченні цикли
- Незмінний параметр, який передається у функцію
- Дублікати коду.

Переваги методу статичного тестування:

- Виконується до розгортання та виконання коду

- Знаходить помилки на перших етапах розробки програмного забезпечення, що сприяє значному зменшенню вартості їх виправлення;
- Відгуки, отримані в процесі даного тестування допомагають удосконалити процес розробки програмного забезпечення та зменшити вірогідність виникнення схожих багів
- Надає високий рівень інформативності щодо проблем програмного забезпечення
- Сприяє ліпшому обміну інформацією між співробітниками
- Виправлення багів та дефектів на початкових етапах потребує значно меншої кількості зусиль, що дозволяє прискорити розробку та зменшити затрати на неї.

Недоліки методу статичного тестування:

- Часто потребує сторонніх засобів тестування, їх налагодження та використання в ручному режимі, тобто довший час тестування
- Приховує певні вразливості, які стосуються середовища виконання програмного забезпечення
- Не дозволяє виявити дефекти в самій логіці програмного забезпечення.

1.4. Вразливості web-застосунків

Вразливості веб-додатків виникають тоді, коли розробники додають небезпечний код в веб-додаток. Це може відбуватися як на етапі розробки, так і на етапі доопрацювання або виправлення знайдених раніше вразливостей. Недоліки часто класифікуються за ступенем критичності і їх поширеності. Об'єктивною і найбільш популярною класифікацією вразливостей вважається OWASP Top 10. Рейтинг складається фахівцями OWASP Project і актуалізується кожні 3-4 роки. Поточний реліз випущений в 2017 році, а наступний очікується в 2021.

1.4.1. OWASP TOP 10 [11]

- A1 Ін'єкції - Вразливості, пов'язані з впровадженням SQL, NoSQL, OS і LDAP. Виникають, коли неперевірені дані відправляються інтерпретатору в складі команди або запиту. Шкідливі дані можуть змусити інтерпретатор виконати непередбачувані команди або звернутися до даних без проходження відповідної авторизації.
- A2 Недоліки аутентифікації - Функції додатків, пов'язані з аутентифікацією і управлінням сесіями, часто некоректно реалізуються, що дозволяє зловмисникам скомпрометувати паролі, ключі або сесійні токени, а також експлуатувати інші помилки реалізації для тимчасового або постійного перехоплення облікових записів користувачів.
- A3 Розголошення конфіденційних даних - Багато веб-додатків та API мають поганий захист критичних фінансових, медичних або персональних даних. Зловмисники можуть викрасти або змінити ці дані, а потім здійснити шахрайські дії з кредитними картами або персональними даними. Конфіденційні дані вимагають додаткових заходів захисту, наприклад їх шифрування при зберіганні або передачі, а також спеціальних запобіжних заходів при роботі з браузером.

- A4 Впровадження зовнішніх сутностей XML - Старі або погано налаштовані XML-процесори обробляють посилання на зовнішні сутності всередині документів. Ці сутності можуть бути використані для доступу до внутрішніх файлів через обробники URI файлів, загальні папки, сканування портів, віддалене виконання коду і відмову в обслуговуванні.
- A5 Недоліки контролю доступу - Дії, дозволені аутентифікованим користувачам, часто некоректно контролюються. Зловмисники можуть скористатися цими недоліками і отримати несанкціонований доступ до облікових записів інших користувачів або конфіденційної інформації, а також змінити призначені для користувача дані або права доступу.
- A6 Некоректні налаштування параметрів безпеки - Некоректне налаштування безпеки є поширеною помилкою. Це відбувається через використання стандартних параметрів безпеки, неповного або специфічного налаштування, відкритого хмарного зберігання, некоректних HTTP-заголовків і докладних повідомлень про помилки, що містять критичні дані. Всі ОС, фреймворки, бібліотеки і додатки повинні бути не тільки налаштовані належним чином, але і своєчасно коректуватися і оновлюватися.
- A7 міжсайтові виконання сценаріїв (Cross Site Scripting) - XSS має місце, коли додаток додає неперевірені дані на нову веб-сторінку без їх відповідної перевірки або перетворення, або коли оновлює відкриту сторінку через API браузера, використовуючи надані вами дані, що містять HTML- або JavaScript-код. За допомогою XSS зловмисники можуть виконувати сценарії в браузері жертви, що дозволяють їм перехоплювати сесії користувачів, підміняти сторінки сайту або перенаправляти користувачів на шкідливі сайти.

- A8 Небезпечна десеріалізація - Небезпечна десеріалізація часто призводить до віддаленого виконання коду. Помилки десеріалізації, що не приводять до віддаленого виконання коду, можуть бути використані для атак з повторним відтворенням, впровадженням і підвищенням привілеїв.
- A9 Використання компонентів з відомими вразливостями - Компоненти, такі як бібліотеки, фреймворки і програмні модулі, запускаються з привілеями програми. Експлуатація уразливого компонента може привести до втрати даних або перехоплення контролю над сервером. Використання додатками і API компонентів з відомими вразливостями може порушити захист додатку і призвести до серйозних наслідків.
- A10 Недоліки логування і моніторингу - Недоліки логування і моніторингу, а також відсутність або неефективне використання системи реагування на інциденти, дозволяє зловмисникам розвинути атаку, приховати свою присутність і проникнути в інші системи, а також змінити, перезавантажити або знищити дані. Проникнення в систему зазвичай виявляють тільки через 200 днів і, як правило, сторонні дослідники, а не в рамках внутрішніх перевірок або моніторингу.

1.4.2. SQL injection

SQL injection - це атака, при якій зловмисний код вставляється в рядки, які згодом передаються екземпляру SQL Server для аналізу та виконання (наприклад, для того, щоб зловмисник міг отримати вміст бази даних). Будь-яку процедуру, яка створює оператори SQL, слід перевіряти на наявність вразливостей, оскільки SQL Server буде виконувати всі синтаксично допустимі запити, які він отримує. Навіть параметризованими даними може керувати кваліфікований та рішучий зловмисник. [12]

Ін'єкція SQL повинна використовувати вразливість безпеки в програмному забезпеченні програми, наприклад, коли ввід користувача або неправильно фільтрується для екранованих символів вхідного рядка, який вбудовується в оператори SQL, або ввід користувача не перевіряється на відповідність очікуваного типу і несподівано виконується. Ін’єкція SQL в основному відома як вектор атаки для веб-сайтів, але може використовуватися для атаки на будь-який тип бази даних SQL.

Атаки SQL ін’єкціями дозволяють зловмисникам підробляти особисті дані, втручатися в існуючі дані, спричиняти проблеми з відмовами, такі як скасування транзакцій або зміна балансу, дозволяють повністю розкрити всі дані в системі, знищити дані або хоча б зробити їх недоступними, і стати адміністраторами серверу баз даних.

У дослідженні 2012 року було помічено, що середній веб-додаток отримував чотири кампанії атак на місяць, а роздрібні продавці отримували вдвічі більше атак, ніж інші галузі. [13]

1.4.2.1 Принцип атаки впровадження SQL

Припустимо, що програмне забезпечення серверу отримує на вхід параметр isbn та створює з ним SQL-запит:

```
String jql = "from BookEntity where isbn = '" + bookIsbn + "'";
TypedQuery<BookEntity> q = entityManager.createQuery(jql,
BookEntity.class);
```

Якщо на сервер переданий параметр isbn, що дорівнює 1 (наприклад, так: http://example.com/book?isbn=5), то запит матиме вигляд:

```
select * from book where isbn = 1;
```

Але, якщо зловмисник передасть як параметр isbn рядок 1 OR ‘1’ = ‘1’ (наприклад, так: http://example.com/book?isbn=1+OR+’1’=’1’), то виконається наступний запит:

```
select * from book where isbn = 1 or '1' = '1';
```

Таким чином, додавання інструкції мови SQL у вхідний запит викликає зміни в роботі застосунку (В даному випадку, повернуться всі книги, а не лише ті, які були передбачені даним запитом).

1.5. Физинг

Fuzz-тестування або Fuzzing - це техніка тестування програмного забезпечення за методом чорної скриньки (Black box), яка в основному полягає у пошуку помилок реалізації за допомогою автоматичного введення неправильно- або напівнеправильно-сформованих даних. [14]

Тестування чорної скриньки (також відоме як функціональне тестування) розглядає програмне забезпечення як "чорну скриньку", що перевіряє функціональність без будь-яких знань про внутрішню реалізацію, не бачачи вихідного коду. Тестери знають лише про те, що має робити програмне забезпечення, а не про те, як це робить. [15] Методи тестування «чорної скриньки» включають: розподіл еквівалентності, аналіз граничного значення, тестування всіх пар, таблиці переходів стану, тестування таблиць рішень, тестування нечіткості, тестування на основі моделі, тестування випадків використання, дослідницьке тестування та тестування на основі специфікації. [16][17]

1.5.1 Типи фазерів

Фазер можна класифікувати кількома способами: [18]

- Фазер може бути заснований на генерації або мутації, залежно від того, чи генеруються вхідні дані з нуля, чи модифікуються існуючі вхідні дані.
- Фазер може бути примітивним або розумним, залежно від того, чи він знає структуру введення.
- Фазер може використовувати тестування за методом білого, сірого або чорного ящика, залежно від того, чи він знає структуру програми.

Розділ 2. Проектування та розробка системи

2.1. Опис складових системи та використаних технологій для розробки

Було вирішено розробити фазер, який буде тестувати web застосунки за методом чорного ящика. Причин цьому є декілька.

По-перше, підхід тестування чорної скриньки дає змогу написати універсальний фазер, який можна було б використовувати для тестування багатьох web застосунків, не зважаючи на внутрішню реалізацію, мову програмування, на якій розроблений застосунок тощо.

По-друге, такий підхід дозволяє покрити ширшу область вразливостей та тестувати web застосунок одразу на декілька з них.

Розроблена система – це консольний застосунок, написаний на мові програмування Java з використанням фреймворку Spring Boot (зокрема, Spring Boot Data). Він є одним з найпопулярніших фреймворків для розробки застосунків на Java, допомагає значно спростити написання коду та дозволяє притримуватися більшої кількості прийнятих практик у розробці програмного забезпечення під час роботи.

В застосунку використовується база даних PostgreSQL, для збереження даних, необхідних для тестування. Ці дані використовуються під час побудови http-запитів до web застосунку, що тестується, а саме: читання шляхів, які оброблює застосунок, формування параметрів запиту на ці шляхи та підстановка замість очікуваних параметрів – тих, що дозволяють виявити вразливості.

Для тестування були обрані декілька типів вразливостей: SQL ін’єкції, XSS атаки та тестування на крайні значення. Було вирішено використовувати готовий збірник параметрів, які зловмисники могли б використати для атаки, оскільки придумати самому такий список було б малоефективно: передбачити всі можливі запити та параметри, без використання готових

списків, сформованих спеціалістами в області комп’ютерної безпеки – неможливо. А невелика кількість даних для тестування зробила б фазер неефективним. З усіх можливих варіантів, був обраний SecLists [19], оскільки він має велику кількість контриб’ютерів, та постійно оновлюється, що дозволяє тестувати застосунки на останніх можливих тестових даних.

Спосіб надходження даних

Полінг клієнт

Висновки

Було досліджено особливості тестування програмного забезпечення, методи та способи тестування, різні підходи до впровадження тестів. У цій роботі був використаний фазинг метод тестування веб-застосунків, оскільки, на думку автора, цей підхід дозволяє охопити широкий спектр вразливостей web застосунків. Також були розглянуті найбільш поширені та часто використовувані вразливості останніх. На фазинг застосунку було показано принцип дії останнього, та особливості його розробки. У майбутніх дослідженнях може розглядатися спосіб покращення застосунку для забезпечення більшої універсальності та гнучкості під час тестування.

Список використаних джерел

1. Hetzel, William C., The Complete Guide to Software Testing, 2nd ed. Publication info: Wellesley, Mass. : QED Information Sciences, 1988. ISBN: 0894352423.
2. Software Testing Carnegie Mellon University 18-849b Dependable Embedded Systems Spring 1999 [Електронний ресурс] – Режим доступу до ресурсу: http://www.sci.brooklyn.cuny.edu/~sklar/teaching/s08/cis20.2/papers/software-testing.pdf.
3. 610.12-1990 - IEEE Standard Glossary of Software Engineering Terminology, IEEE, 1990, ISBN 9781559370677 [Електронний ресурс] – Режим доступу до ресурсу: https://ieeexplore.ieee.org/document/159342
4. Kaner, Cem; Falk, Jack; Nguyen, Hung Quoc (1999). Testing Computer Software, 2nd Ed. New York, et al: John Wiley and Sons, Inc. c. 480. ISBN 0-471-35846-0. [Електронний ресурс] – Режим доступу до ресурсу: https://books.google.com.ua/books?id=Q-hTDwAAQBAJ&printsec=frontcover&dq=ISBN+0471358460&hl=uk&sa=X&ved=2ahUKEwj63oCK09DwAhXh-yoKHepCBMUQ6AEwBnoECAEQAg#v=onepage&q&f=false
5. Graham, D.; Van Veenendaal, E.; Evans, I. (2008). *Foundations of Software Testing*. Cengage Learning. pp. 57–58. ISBN 9781844809899. [Електронний ресурс] – Режим доступу до ресурсу: https://books.google.com/books?id=Ss62LSqCa1MC&pg=PA57
6. Oberkampf, W.L.; Roy, C.J. (2010). *Verification and Validation in Scientific Computing*. Cambridge University Press. pp. 154–5. ISBN 9781139491761. [Електронний ресурс] – Режим доступу до ресурсу: https://books.google.com/books?id=7d26zLEJ1FUC&pg=PA155
7. Oberkampf, W.L.; Roy, C.J. (2010). *Verification and Validation in Scientific Computing*. Cambridge University Press. pp. 154–5. ISBN 9781139491761.
8. Improving Software Security with Precise Static and Runtime Analysis, Benjamin Livshits, section 7.3 "Static Techniques for Security, " Stanford doctoral thesis, 2006. http://research.microsoft.com/en-us/um/people/livshits/papers/pdf/thesis.pdf
9. Lewis, W.E. (2016). *Software Testing and Continuous Quality Improvement* (3rd ed.). CRC Press. pp. 92–6. ISBN 9781439834367.
10. IEEE (1998). *IEEE standard for software test documentation*. New York: IEEE. ISBN 978-0-7381-1443-9.
11. OWASP Top Ten https://owasp.org/www-project-top-ten/
12. Microsoft. "SQL Injection" https://web.archive.org/web/20130802094425/http://technet.microsoft.com/en-us/library/ms161953(v=sql.105).aspx
13. Imperva Web Application Attack Report Edition #4 – July 2013 https://web.archive.org/web/20130907005249/http://www.imperva.com/docs/HII_Web_Application_Attack_Report_Ed4.pdf
14. https://owasp.org/www-community/Fuzzing
15. Patton, Ron (2005). *Software Testing* (2nd ed.). Indianapolis: Sams Publishing. ISBN 978-0672327988.
16. Limaye, M.G. (2009). *Software Testing*. Tata McGraw-Hill Education. pp. 108–11. ISBN 9780070139909.
17. Saleh, K.A. (2009). *Software Engineering*. J. Ross Publishing. pp. 224–41. ISBN 9781932159943.

18. Michael Sutton; Adam Greene; Pedram Amini (2007). *Fuzzing: Brute Force Vulnerability Discovery*. Addison-Wesley. ISBN 978-0-321-44611-4.
19. danielmiessler/SecLists https://github.com/danielmiessler/SecLists/tree/master/Fuzzing/Databases