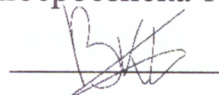


Міністерство освіти і науки України
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки

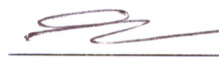
ПОЯСНЮВАЛЬНА ЗАПИСКА

до випускної кваліфікаційної роботи магістра
на тему: «Методи та засоби виявлення вразливості
Web-додатків»

Виконала: студентка 2-го курсу групи ЗКБм-23-1
спеціальності 125 «Кібербезпека та захист інформації»

 Вікторія Рейнська

Керівник, доцент кафедри комп'ютерної інженерії та
кібербезпеки, кандидат технічних наук, доцент

 Юрій Россінський

Рецензент, доцент кафедри комп'ютерної інженерії та
кібербезпеки, кандидат технічних наук

 Олександр Пірог

Житомир – 2024

Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки
Спеціальність 125 «Кібербезпека та захист інформації»
Освітня програма «Кібербезпека»

ЗАТВЕРДЖУЮ

Завідувач

кафедри комп'ютерної
інженерії та кібербезпеки

 Андрій ЄФІМЕНКО

17 вересня 2024 р.

ЗАВДАННЯ

на випускню кваліфікаційну роботу магістра

Студент: Рейнська Вікторія Борисівна

Тема роботи: «Методи та засоби виявлення вразливості Web-додатків»

затверджена Наказом університету від 17 вересня 2024 р. № 480/с

Термін здачі студентом закінченої роботи 16 грудня 2024 р.

Вихідні дані роботи (зазначається предмет і об'єкт дослідження):

1. Об'єкт дослідження — процес виявлення вразливостей у веб-додатках.
2. Предмет дослідження — методи, алгоритми та інструменти для ідентифікації й аналізу вразливостей у веб-додатках.
3. Наукові джерела, що висвітлюють процедуру виявлення вразливостей у веб-додатках

Обладнання та програмне забезпечення:

1. **Обладнання:**
 - Потужний комп'ютер або ноутбук (мінімум 16 ГБ ОЗП, процесор Intel Core i7 або аналогічний).
 - Сервер або хостинг (локальний або хмарний) для розгортання тестового середовища.
2. **Програмне забезпечення:**
 - Операційна система: Windows, Linux.

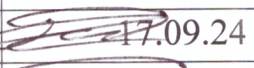
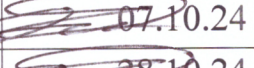
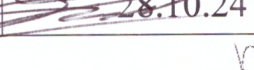
- Інструменти для виявлення вразливостей: Burp Suite (професійна або ком'юніті версія); OWASP ZAP; Nmap для мережевого сканування; Nikto для тестування веб-серверів.

- Мови програмування та середовища розробки: Python (з бібліотеками: Scapy, Requests, BeautifulSoup); JavaScript (Node.js для серверного коду); IDE: Visual Studio Code, PyCharm, або Eclipse.

- Системи для управління базами даних: MySQL, PostgreSQL або MongoDB.

- Тестове середовище: Віртуальні машини (наприклад, VirtualBox або VMware) або Docker.

Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються

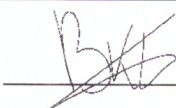
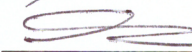
Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Андрій Єфіменко	 17.09.24	 17.09.24
2	Андрій Єфіменко	 07.10.24	 07.10.24
3	Андрій Єфіменко	 28.10.24	 28.10.24

Студент  Вікторія Рейнська

Керівник  Юрій Россінський

Календарний план

№ з/п	Назви етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка завдання	17 вересня 2024	виконано
2	Пошук огляд та аналіз аналогічних розробок	18 вересня 2024 – 19 жовтня 2024	виконано
3	Опрацювання літературних джерел	20 жовтня 2024 – 27 жовтня 2024	виконано
4	Інтерпретація проблеми і формулювання завдань дослідження	27 жовтня 2024 – 07 листопада 2024	виконано
5	Конструювання технік виявлення веб-вразливостей	08 листопада 2024 – 15 листопада 2024	виконано
6	Проектування сканера для виявлення веб-вразливостей	16 листопада 2024 – 27 листопада 2024	виконано
7	Аналіз отриманих результатів	28 листопада 2024	виконано
8	Формування пояснювальної записки	29 листопада 2024 – 6 грудня 2024	виконано
9	Підготовка презентації до виступу	7 грудня 2024	виконано
11	Захист	25 грудня 2024	виконано

Студент  Вікторія Рейнська
Керівник  Юрій Россінський

РЕФЕРАТ

Випускна кваліфікаційна робота магістра складається з проекту розробки сканера для виявлення вразливостей веб-додатків та пояснювальної записки.

Пояснювальна записка до випускної роботи викладена на 85 сторінках, містить 32 рисунки, які розміщено на 20 сторінках. Перелік джерел посилань містить 30 літературних джерел і займає 3 сторінки.

Метою роботи є розробка та впровадження ефективного інструменту для виявлення основних загроз інформаційній безпеці веб-додатків. Особлива увага приділяється підвищенню точності аналізу та зменшенню кількості хибнопозитивних результатів.

Об'єктом дослідження є процес виявлення вразливостей у веб-додатках.

Предметом дослідження є методи, алгоритми та інструменти для ідентифікації й аналізу вразливостей у веб-додатках.

У першому розділі кваліфікаційної роботи представлено огляд сучасних методів виявлення веб-вразливостей, а також проведено аналіз еволюції веб-додатків та основних загроз їхній безпеці. Другий розділ присвячений розробці алгоритмів та технік ідентифікації вразливостей, а також тестуванню компонентів розробленого інструменту. У третьому розділі наведено архітектуру, алгоритми роботи та інтерфейс користувача розробленого сканера для виявлення веб-вразливостей. Проведено тестування інструменту на реальних і симульованих веб-додатках та оцінено його ефективність.

Результати роботи можуть бути використані розробниками веб-додатків, адміністраторами систем та підприємствами для своєчасного виявлення та усунення вразливостей, що сприятиме підвищенню рівня інформаційної безпеки.

Ключові слова: ВЕБ-ДОДАТКИ, ВРАЗЛИВОСТІ, СКАНЕР, ІНФОРМАЦІЙНА БЕЗПЕКА, АЛГОРИТМИ.

ABSTRACT

The master's thesis consists of a project to develop a scanner for detecting web application vulnerabilities and an explanatory note.

The explanatory note to the final thesis is set out on 85 pages, contains 32 figures, which are placed on 20 pages. The list of references includes 30 literary sources and occupies 3 pages.

The aim of the work is to develop and implement an effective tool for identifying the main threats to the information security of web applications. Particular attention is paid to improving the accuracy of the analysis and reducing the number of false positives.

The object of research is the process of identifying vulnerabilities in web applications.

The subject of research is methods, algorithms and tools for identifying and analysing vulnerabilities in web applications.

The first chapter of the qualification work provides an overview of modern methods of detecting web vulnerabilities, as well as an analysis of the evolution of web applications and the main threats to their security. The second section is devoted to the development of algorithms and techniques for identifying vulnerabilities, as well as testing the components of the developed tool. The third section describes the architecture, algorithms and user interface of the developed web vulnerability scanner. The tool was tested on real and simulated web applications and its effectiveness was evaluated.

The results of the work can be used by web application developers, system administrators and enterprises for timely detection and elimination of vulnerabilities, which will help to increase the level of information security.

Keywords: WEB APPLICATIONS, VULNERABILITIES, SCANNER, INFORMATION SECURITY, ALGORITHMS.

ЗМІСТ

ВСТУП.....	8
РОЗДІЛ 1. ІНТЕРПРЕТАЦІЯ ПРОБЛЕМИ І ФОРМУЛЮВАННЯ ЗАВДАНЬ ДОСЛІДЖЕННЯ	10
1.1 Ретроспектива розвитку веб додатків	10
1.2 Методи збереження конфіденційності веб-додатків	12
1.3 Алгоритм роботи засобів виявлення вразливостей в веб-додатках	13
РОЗДІЛ 2. Конструювання технік виявлення веб-вразливостей.....	20
2.1 Візуалізація вмісту додатка	20
2.2 Оцінка ефективності веб-додатку.....	24
2.3. Тестування компонентів розширення браузера	27
2.4 Перевірка контролю доступу.....	41
2.5 Визначення вразливостей на основі вхідних даних.....	45
РОЗДІЛ 3. Проектування сканера для виявлення веб-вразливостей.....	61
3.1 Інструменти та технології розробки	61
3.2 Алгоритм виявлення вразливостей системи	64
3.3 Інтерфейс взаємодії користувача з програмою	78
ВИСНОВКИ	81
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ	83

ВСТУП

У сучасному світі веб-додатки стали невід'ємною частиною повсякденного життя, охоплюючи різні аспекти діяльності — від електронної комерції до державного управління. Водночас зі зростанням популярності веб-додатків зростає і кількість загроз для їхньої безпеки. Виявлення та усунення вразливостей веб-додатків є одним із ключових завдань сучасної інформаційної безпеки, що визначає актуальність теми дослідження.

Актуальність теми. Веб-додатки часто містять критичні дані, включно з персональною інформацією користувачів, що робить їх привабливими цілями для кіберзлочинців. Відсутність належних заходів захисту може призвести до серйозних наслідків: витоку даних, компрометації бізнес-процесів, фінансових втрат та втрати репутації. Оскільки існує значна кількість методів атак, таких як SQL-ін'єкції, міжсайтовий скриптинг (XSS) та атаки зловмисного програмного забезпечення, розробка ефективних засобів виявлення цих загроз стає вкрай важливою.

Об'єкт дослідження — процес виявлення вразливостей у веб-додатках.

Предмет дослідження — методи, алгоритми та інструменти для ідентифікації й аналізу вразливостей у веб-додатках.

Мета дослідження полягає у розробці та впровадженні ефективного сканера веб-вразливостей, здатного ідентифікувати основні загрози інформаційній безпеці.

Завдання дослідження:

- Проаналізувати сучасні методи та техніки виявлення веб-вразливостей.
- Дослідити алгоритми роботи інструментів для аналізу безпеки веб-додатків.
- Розробити архітектуру сканера для виявлення веб-вразливостей.
- Реалізувати сканер, використовуючи сучасні технології програмування.
- Провести тестування створеного інструменту на реальних і симульованих веб-додатках.
- Оцінити ефективність запропонованого рішення у виявленні найпоширеніших вразливостей.

- Запропонувати рекомендації щодо підвищення рівня захисту веб-додатків у контексті виявлених загроз.

Методи дослідження. У роботі застосовано аналітичні методи для дослідження існуючих підходів до виявлення вразливостей, методи математичного моделювання для створення алгоритмів роботи сканера, а також емпіричні методи для оцінки ефективності розробленого інструменту.

Наукова новизна. Розроблено інноваційний підхід до виявлення веб-вразливостей, що поєднує існуючі техніки сканування з удосконаленими алгоритмами обробки вхідних даних. Запропоноване рішення дозволяє зменшити ймовірність хибнопозитивних результатів та підвищити ефективність ідентифікації критичних загроз.

Практичне значення. Створений сканер може бути використаний підприємствами, розробниками веб-додатків та адміністраторами систем для своєчасного виявлення вразливостей, що сприятиме зменшенню ризиків несанкціонованого доступу до інформаційних систем.

У магістерській роботі структуру дослідження поділено на три розділи. У першому розділі розглянуто теоретичні основи виявлення веб-вразливостей, ретроспективу розвитку веб-додатків та методи забезпечення їхньої безпеки. Другий розділ присвячено розробці алгоритмів та технік виявлення вразливостей, а також аналізу ефективності запропонованих рішень. У третьому розділі представлено розроблений сканер веб-вразливостей, описано його архітектуру, інтерфейс користувача та результати тестування.

Результати цього дослідження можуть бути основою для подальшого вдосконалення систем аналізу безпеки веб-додатків, а також для розробки нових методів забезпечення інформаційної безпеки.

РОЗДІЛ 1. ІНТЕРПРЕТАЦІЯ ПРОБЛЕМИ І ФОРМУЛЮВАННЯ ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Ретроспектива розвитку веб додатків

У перші дні існування Інтернету Всесвітня павутина складалася тільки з веб-сайтів. По суті, це були інформаційні сховища, що містили статичні документи. Веб-браузери були винайдені як засіб вилучення і відображення цих документів. Потік інформації був одностороннім, від сервера до браузера. Більшість сайтів не перевіряли автентичність користувачів, тому що в цьому не було потреби. З кожним користувачем поводитися однаково і їм була представлена однакова інформація. Будь-які загрози безпеці, що виникали під час розміщення веб-сайту, були пов'язані здебільшого з уразливостями в програмному забезпеченні веб-сервера (яких було багато). Якщо злоумисник скомпрометував веб-сервер, він зазвичай не отримував доступу до будь-якої конфіденційної інформації, оскільки інформація, що зберігається на сервері, вже була відкрита для публічного перегляду. Найімовірніше, злоумисник зазвичай змінював файли на сервері, щоб зіпсувати вміст вебсайту або використати сховище і пропускну здатність сервера.

Сьогодні Всесвітня павутина майже невпізнанна порівняно з її більш ранньою формою. Більшість сайтів в Інтернеті насправді є додатками. Вони дуже функціональні і покладаються на двосторонній потік інформації між сервером і браузером. Вони підтримують реєстрацію та вхід у систему, фінансові операції, пошук і створення контенту користувачами. Контент, представлений користувачам, генерується динамічно на льоту і часто адаптується до кожного конкретного користувача. Велика частина оброблюваної інформації є приватною і дуже конфіденційною. Безпека, тому це велика проблема. Ніхто не хоче використовувати веб-додаток, якщо він вважає, що його інформацію буде розкрито неавторизованим особам.

Розвиток веб-додатків несе з собою нові та значні загрози безпеці. Кожен додаток відрізняється і може містити унікальні вразливості. Більшість застосунків розробляються власними силами - багато з них розробниками, які

мають лише часткове уявлення про проблеми безпеки, що можуть виникнути в створюваному ними коді. Для забезпечення своєї основної функціональності веб-додаткам зазвичай потрібне під'єднання до внутрішніх комп'ютерних систем, що містять високочутливі дані і здатні виконувати потужні бізнес-функції. П'ятнадцять років тому, якщо хочеш здійснити переказ коштів, ви відвідали свій банк, і касир виконав переказ за вас; сьогодні можна завітати до веб-додатку і виконати переказ самостійно[1]. Зловмисник, який компрометує веб-додаток, може вкрати особисту інформацію, здійснити фінансове шахрайство та виконати шкідливі дії проти інших користувачів.

Веб-додатки були створені для виконання практично всіх корисних функцій, які ви могли б реалізувати в Інтернеті. Ось деякі функції веб-додатків, які стали помітними в останні роки:

Покупки (Amazon) Соціальні мережі (Facebook) Пошук в Інтернеті (Google) Аукціони (eBay)

Додатки, доступ до яких здійснюється за допомогою браузера комп'ютера, все частіше перетинаються з мобільними додатками, доступ до яких здійснюється за допомогою смартфона або планшета. Найбільш мобільні додатки використовують або браузер, або налаштований клієнт, який використовує API на основі HTTP для зв'язку з сервером. Функції та дані програми зазвичай спільно використовуються між різними інтерфейсами, які додаток надає різним призначенням для користувача платформам. На додаток до загальнодоступного Інтернету, веб-додатки широко використовуються всередині організацій для підтримки ключових бізнес-функцій. Багато з них забезпечують доступ до високочутливих даних і функцій:

1) HR-додатки, що дають змогу користувачам отримувати доступ до інформації про заробітну плату, надавати й одержувати відгуки про результати роботи та керувати процедурами найму й дисциплінарними процедурами.

2) Адміністративні інтерфейси для ключових інфраструктур, як-от веб- і поштові сервери, робочі станції користувачів і адміністрування віртуальних машин.

3) Програмне забезпечення для спільної роботи, що використовується для

обміну документами, управління потоком робіт і проектами, а також відстеження проблем. Ці типи функцій часто пов'язані з критичними проблемами безпеки та управління, і організації часто повністю покладаються на елементи управління, вбудовані в їхні веб-додатки.

4) Бізнес-додатки, як-от програмне забезпечення для планування корпоративних ресурсів (ERP), доступ до яких раніше здійснювали за допомогою пропрієтарного клієнтського застосунку, тепер можна отримати за допомогою веб-браузера.

У всіх цих прикладах те, що сприймається як «внутрішні» додатки, все частіше розміщується зовні, оскільки організації переходять до зовнішніх постачальників послуг, щоб скоротити витрати. У цих так званих хмарних рішеннях критично важливі для бізнесу функції та дані відкриті для ширшого кола потенційних злоумисників, і організації дедалі більше покладаються на цілісність засобів захисту, які перебувають поза їхнім контролем[2].

Швидко наближається час, коли єдиним клієнтським програмним забезпеченням, яке знадобиться більшості користувачів комп'ютерів, буде веб-браузер. Широкий спектр функцій буде реалізовано з використанням загального набору протоколів і технологій, і водночас буде успадковано особливу низку загальних вразливостей безпеки.

1.2 Методи збереження конфіденційності веб-додатків

Як і у випадку з будь-яким новим класом технологій, веб-додатки принесли із собою новий спектр вразливостей у системі безпеки. Набір найпоширеніших дефектів дещо еволюціонував із плином часу. Були придумані нові атаки, які не враховувалися під час розроблення наявних застосунків. Деякі проблеми стали менш поширеними в міру підвищення обізнаності про них. Було розроблено нові технології, які відкрили нові можливості для експлуатації. Деякі категорії недоліків значною мірою зникли внаслідок змін, внесених у програмне забезпечення веб-браузера.

Найсерйознішими атаками на веб-додатки є ті, які розкривають конфіденційні дані або отримують необмежений доступ до внутрішніх систем,

на яких працює додаток. Гучні компроміси такого роду, як і раніше, відбуваються часто. Однак для багатьох організацій будь-яка атака, що призводить до простою системи, є критичною подією. Атаки типу «відмова в обслуговуванні» на рівні додатків можуть використовуватися для досягнення тих самих результатів, що і традиційні атаки на вичерпання ресурсів проти інфраструктури. Однак вони часто використовуються з більш тонкими методами та цілі. Вони можуть бути використані для порушення роботи конкретного користувача або сервісу, щоб отримати конкурентну перевагу перед колегами в галузі фінансової торгівлі, ігор, онлайн-торгів та бронювання квитків.

1.3 Алгоритм роботи засобів виявлення вразливостей в веб-додатках

Існує два основні механізми, за допомогою яких сканер перевіряє наявність уразливості:

Сканування - механізм пасивного аналізу, за допомогою якого сканер намагається визначити наявність уразливості без фактичного підтвердження її наявності, тобто за непрямыми ознаками. Цей метод є найбільш швидким і простим для реалізації. Цей процес ідентифікує відкриті порти, знайдені на кожному мережевому пристрої, і збирає пов'язані з портами заголовки (banner), знайдені під час сканування кожного порту. Кожен отриманий заголовок порівнюється з таблицею правил визначення мережевих пристроїв, операційних систем і потенційних вразливостей. На основі проведеного порівняння робиться висновок про наявність або відсутність уразливості.

Зондування - механізм активного аналізу, який дає змогу переконатися, присутня чи ні на аналізованому вузлі вразливість. Зондування виконується шляхом імітації атаки, яка використовує вразливість, що перевіряється. Цей метод повільніший, ніж сканування, але майже завжди набагато точніший, ніж він. Цей процес використовує інформацію, отриману в процесі сканування, для детального аналізу кожного мережевого пристрою. Цей процес також використовує відомі методи реалізації атак для того, щоб повністю підтвердити передбачувані вразливості та виявити інші вразливості, які не можуть бути виявлені пасивними методами, наприклад схильність до атак типу «відмова в

обслуговуванні» (DoS - denialofservice).

На практиці зазначені механізми реалізуються такими методами:

1. Перевірка заголовків (bannercheck) - зазначений механізм являє собою низку перевірок типу «сканування» і дає змогу робити висновок про вразливість, спираючись на інформацію в заголовку відповіді на запит сканера. Типовий приклад такої перевірки - аналіз заголовків програми Sendmail або FTP-сервера, що дає змогу дізнатися їхню версію і на основі цієї інформації зробити висновок про наявність у них уразливості.

Це найшвидший і найпростіший для реалізації метод перевірки присутності на сканованому вузлі вразливості. Однак ефективність перевірок заголовків досить ефемерна. І ось чому. По-перше, можна змінити текст заголовка, завбачливо видаливши з нього номер версії або іншу інформацію, на підставі якої сканер буде свої висновки. По-друге, найчастіше, версія, що вказується в заголовку відповіді на запит, не завжди свідчить про вразливість програмного забезпечення. Особливо це стосується програмного забезпечення, що поширюється разом з вихідними текстами. Можна самотійно усунути вразливість шляхом модифікації вихідного тексту, при цьому, забувши змінити номер версії в заголовку. По-третє, усунення вразливості в одній версії ще не означає, що в наступних версіях ця вразливість відсутня. Процес, описаний вище, є першим і дуже важливим кроком при скануванні мережі. Він не призводить до порушення функціонування сервісів або вузлів мережі.

Активні зондувальні перевірки (activeprobingcheck) - також належать до механізму сканування. Однак вони засновані на порівнянні «цифрового зліпка» фрагмента програмного забезпечення зі зліпком відомої вразливості. Аналогічним чином чинять антивірусні системи, порівнюючи фрагменти сканованого програмного забезпечення з сигнатурами вірусів, що зберігаються в спеціалізованій базі даних. Різновидом цього методу є перевірки контрольних сум або дати програмного забезпечення, що сканується, які реалізуються в сканерах, які працюють на рівні операційної системи. Спеціалізована база даних із мережевої безпеки містить інформацію про вразливості та способи їх використання (атаки). Ці дані доповнюються відомостями про заходи їх

усунення, що дають змогу знизити ризик безпеки в разі їх виявлення. Найчастіше ця база даних використовується і системою аналізу захищеності, і системою виявлення атак. Цей метод також досить швидкий, але реалізується важче, ніж «перевірка заголовків».

1. імітація атак (exploitcheck) - дані перевірки належать до механізму зондування і засновані на експлуатації різних дефектів у програмному забезпеченні. деякі вразливості не виявляють себе, поки ви не «підштовхнете» їх. Для цього проти підозрілого сервісу або вузла запускаються реальні атаки. Перевірки заголовків здійснюють первинний огляд мережі, а метод «exploitcheck», відкидаючи інформацію в заголовках, дає змогу імітувати реальні атаки, тим самим з більшою ефективністю (але меншою швидкістю) виявляючи вразливості на сканованих вузлах. Імітація атак є надійнішим способом аналізу захищеності, ніж перевірки заголовків, і зазвичай надійніші за активні зондувальні перевірки. Однак існують випадки, коли імітація атак не завжди може бути реалізована. Такі випадки можна розділити на дві категорії: ситуації, в яких тест призводить до «відмови в обслуговуванні» аналізованого вузла або мережі, і ситуації, за яких уразливість у принципі непридатна для реалізації атаки на мережу[3].

Багато проблем захисту не можуть бути виявлені без блокування або порушення функціонування сервісу або комп'ютера в процесі сканування. У деяких випадках небажано використовувати імітацію атак (наприклад, для аналізу захищеності важливих серверів), тому що це може призвести до великих витрат (матеріальних і часових) на відновлення працездатності виведених з ладу елементів корпоративної мережі. У цих випадках бажано застосувати інші перевірки, наприклад, активне зондування або, в крайньому разі, перевірки заголовків.

Однак, є деякі вразливості (наприклад, перевірка схильності до атак типу «PacketStorm»), які просто не можуть бути протестовані без можливого виведення з ладу сервісу або комп'ютера. У цьому разі розробники чинять так, - за замовчуванням такі перевірки вимкнені, і користувач може сам увімкнути їх, якщо бажає. Під час увімкнення будь-якої з перевірок цієї групи, системи

видають повідомлення типу «WARNING:"

(«Увага: ці перевірки можуть вивести з ладу або перезавантажити скановані вузли»).

Практично будь-який сканер проводить аналіз захищеності в кілька етапів:

1. Збір інформації про мережу. На цьому етапі ідентифікуються всі активні пристрої в мережі та визначаються запущені на них сервіси й демони. У разі використання систем аналізу захищеності на рівні операційної системи цей етап пропускають, оскільки на кожному вузлі, що аналізується, встановлено відповідних агентів системного сканера.

2. Виявлення потенційних вразливостей. Сканер використовує описану вище базу даних для порівняння зібраних даних із відомими вразливостями за допомогою перевірки заголовків або активних зондувальних перевірок. У деяких системах усі вразливості ранжуються за ступенем ризику: висока (High), середня (Medium) і низька (Low).

3. Підтвердження обраних вразливостей. Сканер використовує спеціальні методи і моделює (імітує) певні атаки для підтвердження факту наявності вразливостей на обраних вузлах мережі.

4. Генерація звітів. На основі зібраної інформації система аналізу захищеності створює звіти, що описують виявлені вразливості. У деяких системах звіти створюються для різних категорій користувачів, починаючи від адміністраторів мережі і закінчуючи керівництвом компанії. Якщо перших насамперед цікавлять технічні деталі, то для керівництва компанії необхідно представити красиво оформлені із застосуванням графіків і діаграм звіти з мінімумом подробиць. Важливим аспектом є наявність рекомендацій щодо усунення виявлених проблем, коли для кожної вразливості видаються покрокові інструкції для усунення вразливостей, специфічні для кожної операційної системи. У багатьох випадках звіти також містять посилання на FTP- або Web-сервери, що містять patch і hotfix, які усувають виявлені вразливості.

5. Автоматичне усунення вразливостей. Цей етап досить рідко реалізується в мережевих сканерах, але широко застосовується в системних сканерах. При цьому ця можливість може реалізовуватися по-різному. Наприклад, створюється

спеціальний сценарій (fixscript), який адміністратор може запустити для усунення вразливості. Одночасно зі створенням цього сценарію, створюється і другий сценарій, що скасовує зроблені зміни. Це необхідно в тому випадку, якщо після усунення проблеми, нормальне функціонування вузла було порушено. В інших системах можливості «відкату» не існує. У адміністратора, який здійснює пошук вразливостей, є кілька варіантів використання системи аналізу захищеності - це різний набір наявних етапів.

Якщо сканер не знаходить вразливостей на тестованому вузлі, то це ще не означає, що їх немає. Просто сканер не знайшов їх. І залежить це не тільки від самого сканера, а й від його оточення. Наприклад, якщо Ви тестуєте сервіс Telnet або FTP на віддаленій машині, і сканер повідомляє Вам, що вразливостей не виявлено, - це може означати не тільки те, що вразливостей немає, а ще й те, що на комп'ютері, що сканується, встановлено, наприклад, TCP Wrapper. Та хіба мало ще чого? Можна намагатися отримати доступ до комп'ютера через міжмережевий екран або спроби доступу блокуються відповідними фільтрами у провайдера тощо. Для ОС Windows NT характерний інший випадок. Сканер намагається дистанційно проаналізувати системний реєстр (registry). Однак у разі заборони на аналізованому вузлі віддаленого доступу до реєстру, сканер жодних вразливостей не виявить. Існують і складніші випадки. І взагалі різні реалізації одного й того самого сервісу по-різному реагують на системи аналізу захищеності. Дуже часто на практиці можна побачити, що сканер показує вразливості, яких на аналізованому вузлі немає. Це стосується мережевих сканерів, які проводять дистанційний аналіз вузлів мережі. І віддалено визначити, чи існує насправді вразливість чи ні, практично неможливо. У цьому разі можна порекомендувати використовувати систему аналізу захищеності на рівні операційної системи, агенти якої встановлюються на кожен контрольований вузол і проводять усі перевірки локально.

Для розв'язання цієї проблеми деякі компанії-виробники пішли шляхом надання своїм користувачам кількох систем аналізу захищеності, що працюють на всіх зазначених вище рівнях, - мережевому, системному і рівні додатків. Сукупність цих систем дає змогу з високим ступенем ефективності виявити

практично всі відомі вразливості. Наприклад, компанія InternetSecuritySystems пропонує сімейство SAFEsuite, що складається з чотирьох сканерів: InternetScanner, SystemScanner, SecurityManager і DatabaseScanner. Наразі це єдина компанія, яка пропонує системи аналізу захищеності, що функціонують на всіх трьох рівнях інформаційної інфраструктури. Інші компанії пропонують або два (Axent) або, як правило, один (NetworkAssociates, NetSonar та ін.) сканер.

Компанія Cisco, що пропонує тільки систему аналізу захищеності на рівні мережі, пішла іншим шляхом для усунення проблеми помилкового спрацювання. Вона ділить усі вразливості на два класи:

1. Потенційні - ті, що впливають із перевірок заголовків і т.зв. активних «підштовхувань» (nudge) аналізованого сервісу або вузла. Потенційна вразливість, можливо, існує в системі, але активні зондувальні перевірки не підтверджують цього.

2. Підтверджені - виявлені та наявні на аналізованому хості. Перевірки на потенційну вразливість проводяться через колекцію заголовків і використання «несильних підштовхувань». «Підштовхування» використовується для сервісів, які не повертають заголовки, але реагують на прості команди, наприклад, надсилання команди HEAD для отримання версії HTTP-сервера. Щойно цю інформацію отримано, система NetSonar використовує спеціальний механізм (rulesengine), який реалізує низку правил, що визначають, чи є потенційна вразливість, таким чином, адміністратор знає, які з виявлених вразливостей справді присутні в системі, а які потребують підтвердження.

Однак у цьому разі залишаються вразливості, які важко виявляються або зовсім не виявляються через мережу. Наприклад, перевірка «слабкості» паролів, використовуваних користувачами та іншими обліковими записами. У разі використання мережевого сканера вам буде потрібно витратити дуже багато часу на віддалену перевірку кожного облікового запису. Водночас аналогічна перевірка, здійснювана на локальному вузлі, проводиться на кілька порядків швидше. Іншим прикладом може слугувати перевірка файлової системи сканованого вузла. У багатьох випадках її не можна здійснити дистанційно.

Переваги сканування на рівні ОС полягають в прямому доступі до

низькорівневих можливостей ОС хоста, конкретних сервісів і деталей конфігурації. Тоді як сканер мережевого рівня імітує ситуацію, яку міг би мати зовнішній злоумисник, сканер системного рівня може розглядати систему з боку користувача, який вже має доступ до аналізованої системи і має в ній обліковий запис. Це є найважливішою відмінністю, оскільки мережевий сканер за визначенням не може надати ефективного аналізу можливих ризиків діяльності користувача. Багато сканерів використовують більш ніж один метод перевірки однієї й тієї самої вразливості або класу вразливостей. Однак у разі великої кількості перевірок використання декількох методів пошуку однієї вразливості приносить свої проблеми. Пов'язано це зі швидкістю проведення сканування. У будь-якому разі найкращою є перевірка типу «імітація атак», яка забезпечує найбільший відсоток точного виявлення вразливостей. Не всі перевірки, розроблені в лабораторних умовах, функціонують так, як повинні. Навіть, незважаючи на те, що ці перевірки тестуються, перш ніж будуть внесені в остаточну версію сканера. На це можуть впливати деякі фактори:

1. Особливості конфігурації користувацької системи.
2. Спосіб, яким було скопійовано аналізований демон або сервіс.
3. Помилки віддаленої системи тощо.

У таких випадках автоматична перевірка може пропустити вразливість, яка легко виявляється вручну і яка може бути широко поширена в багатьох системах. Перевірка заголовка в сукупності з активним зондуванням у такому разі може допомогти визначити підозрілу ситуацію, сервіс або вузол. І хоча вразливість не виявлено, ще не означає, що її не існує. Необхідно іншими методами, зокрема й неавтоматизованими, досліджувати кожен підозрілий випадок.

Крім того, якщо у створеному звіті не сказано про ту чи іншу вразливість, то іноді варто звернутися до журналів реєстрації (log) системи аналізу захищеності. У деяких випадках, коли сканер не може зі 100%-ою впевненістю визначити наявність уразливості, він не записує цю інформацію у звіт, однак зберігає її в логах. Існують відмінності й між тим, як впливає одна й та сама перевірка на різні версії сервісів у різних операційних системах.

РОЗДІЛ 2. Конструювання технік виявлення веб-вразливостей

2.1 Візуалізація вмісту додатка

На першому етапі пошуку вразливостей важливо здійснити розвідку, щоб зібрати необхідну інформацію. Для цього слід виконати наступні кроки (відповідно до рисунка 2.1).

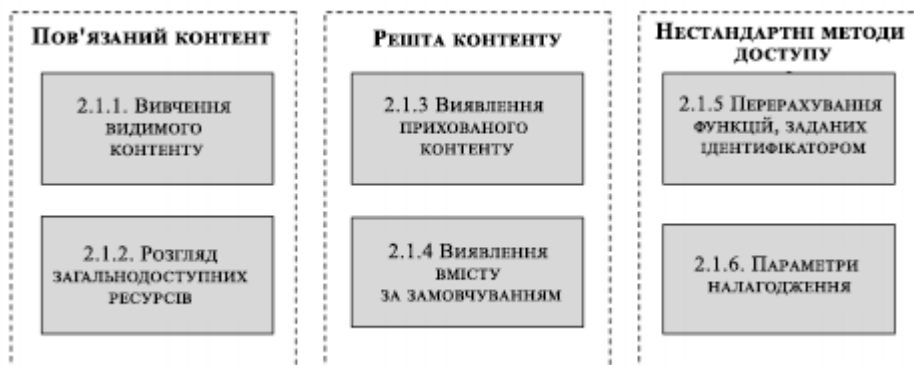


Рисунок 2.1 - Відображення вмісту додатка

Вивчення видимого контенту.

Необхідно налаштувати свій браузер на використання вашого улюбленого вбудованого проксі-сервера/інструменту spider. Як Burp, так і WebScarab можна використовувати для пасивного спайдера сайту шляхом моніторингу та аналізу веб-контенту, оброблюваного проксі-сервером.

Потрібно налаштувати свій браузер на використання розширення, такого як IEWatch, для моніторингу та аналізу вмісту HTTP і HTML, оброблюваного браузером.

Необхідно переглянути весь застосунок у звичайний спосіб, відвідуючи кожне посилання і URL, надсилаючи кожен форму і проходячи через усі багатоступеневі функції до завершення. Слід спробувати переглядати сторінки з увімкненим і вимкненим JavaScript, а також з увімкненими і вимкненими файлами cookie. Багато додатків можуть обробляти різні конфігурації браузера, і можна отримати доступ до різних контентів шляхів коду в додатку.

Якщо додаток використовує аутентифікацію, і якщо є можливість створити обліковий запис для входу, потрібно використовувати його для доступу до захищених функцій.

Під час перегляду потрібно стежити за запитами і відповідями, що

проходять через проксі-сервер, який перехоплює, щоб отримати уявлення про типи даних, які надсилаються, і способи використання клієнта для управління поведінкою серверного застосунку.

Потрібно переглянути карту сайту, створену пасивним спайдерінгом, і визначити будь-який контент або функціональність, які не були переглянуті за допомогою браузера. З результатів spider потрібно визначити, де було виявлено кожен елемент (наприклад, у Burp Spider потрібно перевірити посилання з відомостей). Необхідно отримати доступ до кожного елемента за допомогою браузера, щоб spider проаналізував відповідь із сервера для ідентифікації будь-якого додаткового контенту. Цей крок потрібно повторювати рекурсивно доти, доки не буде ідентифіковано додатковий вміст або функціональність.

Після того як перегляд вручну і пасивний спайдерінг буде закінчено, можна використати свого спайдера для активного обходу програми, використовуючи набір виявлених URL-адрес як вихідні дані. Іноді це може призвести до виявлення додаткового контенту, який було пропущено під час збору даних вручну. Перш ніж виконувати автоматичний обхід, потрібно визначити всі URL-адреси, які небезпечні або можуть порушити сеанс застосунку, а потім налаштувати spider, щоб виключити їх зі своєї сфери дії.

Розгляд загальнодоступних ресурсів.

Потрібно використовувати пошукові системи Інтернету та архіви (наприклад, Wayback Machine), щоб визначити, який контент вони проіндексували та зберегли для цільового застосунку.

Слід використовувати розширені параметри пошуку, щоб підвищити ефективність ваших досліджень. Наприклад, у Google можна використовувати site: для отримання всього контенту для необхідного цільового сайту та link: для отримання інших сайтів, які посилаються на нього. Якщо проводиться пошук контенту, якого більше немає в додатку, все одно можна переглянути його з кешу пошукової системи. Цей старий контент може містити посилання на додаткові ресурси, які ще не були видалені.

Потрібно виконати пошук за будь-якими іменами та адресами електронної пошти, які було виявлено у вмісті додатка, наприклад за контактною

інформацією. Для підвищення ефективності слід увімкнути елементи, які не відображаються на екрані, такі як HTML-коментарі. На додаток до пошуку в Інтернеті слід виконати пошук новин і груп. Потрібно шукати будь-які технічні подробиці, розміщені на інтернет-форумах, що стосуються цільового додатка та його інфраструктури.

Слід переглянути всі опубліковані файли WSDL, щоб створити список імен функцій і значень параметрів, які потенційно використовуються додатком.

Виявлення прихованого контенту. Потрібно підтвердити, як додаток обробляє запити на неіснуючі елементи. Зробити кілька запитів вручну для відомих допустимих і неприпустимих ресурсів і порівняти відповіді сервера, щоб встановити простий спосіб визначити, коли елемент не існує.

Отримавши список загальних імен файлів і каталогів, а також загальних розширень файлів, необхідно додати в ці списки всі елементи, які фактично спостерігаються в додатках, а також елементи, виведені з них. Також слід звернути увагу на імена функцій і параметрів, які використовуються розробниками додатків. Наприклад, якщо існують сторінки з іменами `addDocument.jsp` і `ViewDocument.jsp`, можуть також існувати сторінки з іменами `EditDocument.jsp` і `RemoveDocument.jsp`.

Слід переглянути весь код на стороні клієнта, щоб визначити будь-які підказки про прихований вміст на стороні сервера, включно з HTML-коментарями та вимкненими елементами форми.

Використовуючи методи автоматизації, потрібно виконувати велику кількість запитів на основі списків каталогів, імен файлів і розширень файлів. Потрібно стежити за відповідями сервера на підтвердження того, які елементи присутні і доступні.

Слід виконувати ці пункти з виявлення вмісту рекурсивно, використовуючи новий перерахований вміст і шаблони як основу для подальшого орієнтованого на користувача спайдерінгу і подальшого автоматичного виявлення.

Виявлення вмісту за замовчуванням

Слід запустити Nikto проти веб-сервера, щоб виявити будь-який присутній

за замовчуванням або добре відомий контент. Слід використовувати можливості Nikto, щоб максимізувати його ефективність. Наприклад, можна використовувати параметр `-root`, щоб вказати каталог для перевірки вмісту за замовчуванням, або `-404`, щоб вказати рядок, що ідентифікує сторінку «Користувачський файл не знайдено».

Слід перевірити всі потенційно цікаві результати вручну, щоб виключити будь-які помилкові спрацьовування в результатах.

Потрібно запросити кореневий каталог сервера, вказавши IP-адресу в заголовку хоста, і необхідно визначити, чи відповідає додаток будь-яким іншим контентом. Якщо це так, слід запустити сканування Nikto за IP-адресою, а також за ім'ям сервера.

Слід зробити запит у кореневий каталог сервера, вказавши діапазон заголовків агента користувача.

Перерахування функцій, заданих ідентифікатором

Необхідно визначити будь-які екземпляри, у яких доступні певні функції додатка, передавши ідентифікатор функції в параметрі запиту (наприклад, `//admin.jsp?action=editUser` або `/main.php?func=A21`).

Слід застосувати методи виявлення вмісту, що використовуються в кроці 1.3, до механізму, що використовується для доступу до окремих функцій. Наприклад, якщо застосунок використовує параметр, що містить ім'я функції, спочатку необхідно визначити його поведінку під час зазначення неприпустимої функції, а також потрібно спробувати знайти простий спосіб визначити, коли було запитано допустиму функцію. Слід скласти список загальних імен функцій або Потрібно пройти по синтаксичному діапазону використовуваних ідентифікаторів. Автоматизуйте вправу, щоб перерахувати допустимі функції якомога швидше і простіше.

Якщо це може бути застосовано, слід скласти карту вмісту програми на основі функціональних шляхів, а не URL-адрес, показуючи всі перелічені функції, а також логічні шляхи та залежності між ними.

Перевірка параметрів налагодження.

Необхідно вибрати одну або кілька сторінок програми або функцій, на

яких можуть бути реалізовані приховані параметри налагодження (наприклад, debug=true). Вони, найімовірніше, з'являться в ключових функціях, таких як вхід у систему, пошук і завантаження або завантаження файлів.

Необхідно використовувати списки загальних імен параметрів налагодження (таких як debug, test, hide і source) і загальних значень (таких як true, yes, on і 1). Повторіть усі їхні перестановки, відправляючи кожну пару ім'я/значення кожній цільовій функції. Для запитів POST вкажіть параметр як у рядку запиту URL, так і в тілі запиту. Необхідно використовувати методи, щоб автоматизувати цю вправу. Наприклад, ви можете використовувати тип атаки касетної бомби в Burp Intruder для об'єднання всіх перестановок двох списків корисного навантаження.

Слід переглянути відповіді застосунку на наявність будь-яких аномалій, які можуть вказувати на те, що доданий параметр вплинув на обробку застосунку.

2.2 Оцінка ефективності веб-додатку

Аналіз веб-додатка складається з таких важливих кроків (відповідно до рис. 2.2):



Рисунок 2.2 - Аналіз програми

Визначення функціональних можливостей.

Необхідно визначити основні функціональні можливості, для яких було створено застосунок, і дії, для виконання яких призначена кожна функція під час використання за призначенням.

Необхідно визначити основні механізми безпеки, що використовуються додатком, і те, як вони працюють. Зокрема, зрозумійте ключові механізми, які обробляють автентифікацію, управління сесіями і контроль доступу, а також функції, які їх підтримують, такі як реєстрація користувачів і відновлення облікових записів.

Необхідно визначити все більш периферійні функції і поведінки, такі як використання перенаправлень, зовнішніх посилань, повідомлень про помилки, адміністративних функцій і функцій ведення журналу.

Необхідно визначити будь-яку функціональність, що відрізняється від стандартного зовнішнього вигляду графічного інтерфейсу, іменування параметрів або механізму навігації, що використовується в інших частинах програми, і потрібно виділити її для поглибленого тестування.

Визначення точок введення даних.

Необхідно визначити всі різні точки входу, які існують для введення користувацьких даних в обробку застосунку, включно з URL-адресами, параметрами рядка запиту, даними POST, файлами cookie та іншими HTTP-заголовками, які обробляються застосунком.

Вивчіть будь-які настроювані механізми передачі даних або кодування, які використовуються додатком, такі як нестандартний формат рядка запиту. Необхідно визначити, чи інкапсулюють дані, що передаються, імена і значення параметрів, чи використовуються альтернативні засоби подання.

Необхідно визначити будь-які позасмугові канали, якими в обробку додатка вводяться контрольовані користувачем або інші сторонні дані. Прикладом може слугувати додаток веб-пошти, який обробляє і відображає повідомлення, отримані через SMTP.

Необхідно визначити використовувані технології, що використовуються.

Необхідно визначити кожен з різних технологій, що використовуються на стороні клієнта, такі як форми, сценарії, файли cookie, Java-аплети, елементи управління ActiveX і флеш-об'єкти.

Наскільки це можливо, необхідно визначити, які технології використовуються на стороні сервера, включно з мовами сценаріїв,

платформами додатків і взаємодією з внутрішніми компонентами, такими як бази даних і системи електронної пошти.

Потрібно перевірити заголовок HTTP-сервера, що повертається у відповідях додатка, а також потрібно перевірити наявність будь-яких інших ідентифікаторів програмного забезпечення, що містяться в призначених для користувача HTTP-заголовках або коментарі до вихідного коду HTML. Потрібно звернути увагу, що в деяких випадках різні області програми обробляються різними внутрішніми компонентами, тому можуть бути отримані різні банери.

Слід запустити інструмент Httprint, щоб зняти відбитки пальців із веб-сервера.

Слід переглянути результати ваших вправ зі зіставлення вмісту, щоб визначити будь-які цікаві розширення файлів, каталоги або інші послідовності URL-адрес, які можуть дати уявлення про технології, що використовуються на сервері. Слід переглянути імена будь-яких токенів сеансу та інших файлів випущених за допомогою cookie. Потрібно скористатися Google для пошуку технологій, пов'язаних із цими елементами.

Нанесення на карту поверхню атаки.

Необхідно спробувати визначити ймовірну внутрішню структуру і функціональність серверного додатка, а також механізми, які він використовує за лаштунками, щоб забезпечити поведінку, видиму з точки зору клієнта. Наприклад, функція для отримання замовлень клієнтів, найімовірніше, буде взаємодіяти з базою даних.

Для кожного елемента функціональності необхідно визначити типи поширених вразливостей, які часто пов'язані з ним. Наприклад, функції завантаження файлів можуть бути вразливі для обходу шляху, повідомлення між користувачами можуть бути вразливі для XSS, а функції зв'язку з нами можуть бути вразливі для впровадження SMTP. Приклади вразливостей, зазвичай пов'язаних із конкретними функціями і технологіями.

Необхідно сформулювати план атаки, визначивши пріоритети найцікавіших функцій і найсерйозніших потенційних вразливостей, пов'язаних із ними. Необхідно використовувати свій план, щоб визначити кількість часу і

зусиль, які ви приділяєте кожній з решти областей цієї методології.

2.3. Тестування компонентів розширення браузера

Нижче представлено основні етапи тестування компонентів розширення браузера (відповідно до рис. 2.3):

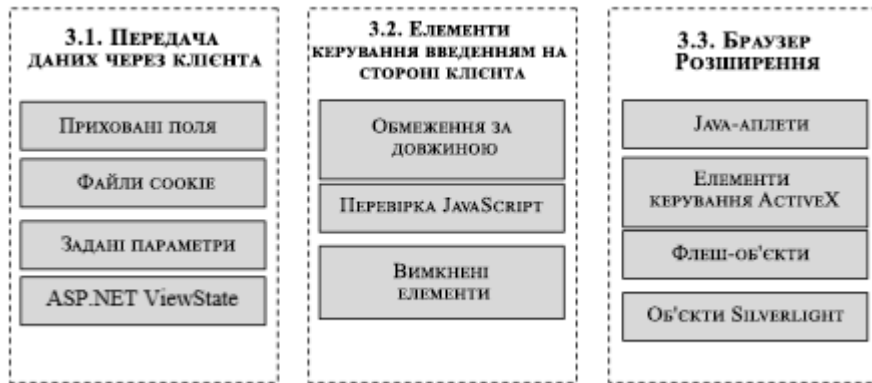


Рисунок 2.3 - Тестування елементів керування на стороні клієнта

Тестова передача даних через клієнта

Потрібно знайти всі екземпляри застосунку, в яких приховані поля форми, файли cookie і параметри URL, мабуть, використовуються для передачі даних через клієнта.

Необхідно спробувати визначити мету, яку елемент відіграє в логіці додатка, ґрунтуючись на контексті, в якому він з'являється, а також на його наявному значенні.

Слід змінити значення елемента таким чином, щоб воно відповідало його ролі у функціональності додатка. Необхідно визначити, чи обробляє застосунок довільні значення, представлені в полі, і чи може цей факт бути використаний для втручання в логіку застосунку або підриву будь-яких елементів управління безпекою.

Якщо додаток передає непрозорі дані через клієнта, можна атакувати його різними способами. Якщо елемент заплутаний, можна розшифрувати алгоритм заплутування і, отже, представити довільні дані в непрозорому елементі. Навіть якщо він надійно зашифрований, можна відтворити елемент в інших контекстах, щоб втрутитися в логіку програми.

Якщо додаток використовує ASP.NET ViewState, потрібно перевірити, чи

можна його підробити або чи містить він будь-яку конфіденційну інформацію. Потрібно звернути увагу, що стан подання може використовуватися по-різному на різних сторінках програми.

Необхідно використовувати аналізатор ViewState у Burp Suite, щоб підтвердити, чи ввімкнено опцію EnableViewStateMac, що означає, що вміст ViewState не може бути змінено.

Слід переглянути декодований стан подання, щоб визначити будь-які конфіденційні дані, що містяться в ньому.

Слід змінити одне зі значень декодованого параметра, повторно закодуєте та надішліть стан подання. Якщо додаток приймає змінене значення, ви повинні розглядати стан подання як вхідний канал для введення довільних даних в обробку додатка. Слід виконати таке саме тестування даних, що містяться в ньому, як і для будь-яких інших параметрів запиту.

Тестування елементів керування введенням на стороні клієнта за допомогою даних користувача

Необхідно визначити будь-які випадки, коли елементи керування на стороні клієнта, як-от обмеження довжини та перевірки JavaScript, використовуються для перевірки користувацьких даних перед їхнім надсиланням на сервер. Ці елементи керування можна легко обійти, оскільки можна надсилати довільні запити на сервер. Наприклад: `<form action=«order.asp» onsubmit=«return Validate(this)»>`

`<inputmaxlength=«3» name=«quantity»>`

Потрібно перевірити кожне порушене поле введення по черзі, надіславши введення, яке зазвичай блокується елементами управління на стороні клієнта, щоб перевірити, чи реплікуються вони на сервері.

Можливість обходу перевірки на стороні клієнта не обов'язково являє собою будь-яку вразливість. Проте, ви повинні уважно вивчити, яка перевірка виконується. Слід переконатися, що програма покладається на елементи управління на стороні клієнта для захисту від спотвореного введення. Також підтвердіть, чи існують будь-які умови експлуатації, які можуть бути викликані таким введенням.

Слід переглянути кожну HTML-форму, щоб визначити будь-які відключені елементи, такі як сірі кнопки надсилання. Наприклад:

```
<inputdisabled=«true» name=«product»>
```

Якщо їх буде виявлено, слід відправити їх на сервер разом з іншими параметрами форми. Необхідно подивитися, чи впливає цей параметр на обробку сервера, яку можна використовувати в атаці. Крім того, необхідно використовувати правило автоматичного проксі-сервера для автоматичного ввімкнення відключених полів, таких як правила «модифікації HTML» проксі-сервера Burp..

Тестування компонентів розширення браузера Розуміння операції клієнтського додатка.

Слід налаштувати локальний проксі-сервер перехоплення для клієнтської технології, що розглядається, і контролюйте весь трафік, що проходить між клієнтом і сервером. Якщо дані серіалізуються, необхідно використовувати інструмент десеріалізації, як-от вбудована підтримка AMF Вигрили користувацький плагін Burp для Java.

Потрібно пройти через функціональність, представлену в клієнті. Необхідно визначити будь-які потенційно чутливі або потужні функції, використовуючи стандартні інструменти в проксі-сервері перехоплення для відтворення ключових запитів або зміни відповідей сервера.

Необхідно визначити всі аплети, що використовуються додатком. Слід шукати будь-який із таких типів файлів, запитуваних через ваш проксі-сервер перехоплення:

.class, .jar : Java

.swf : Flash

.xap : Silverlight

Також слід шукати теги аплетів у вихідному коді HTML сторінок додатків. Наприклад:

```
<applet code=«input.class» id=«TheApplet» codebase=«/scripts/»></applet>
```

Слід переглянути всі виклики методів аплету з HTML, що викликає, і необхідно визначити, чи передаються дані, повернуті з аплету, на сервер. Якщо

ці дані непрозорі (тобто заплутані або зашифровані), щоб змінити їх, вам, імовірно, доведеться декомпілювати аплет, щоб отримати його вихідний код.

- Завантажте байт-код аплету, ввівши URL-адресу у свій браузер, і збережіть файл локально. Ім'я файлу байт-коду вказується в атрибуті коду тега аплету. Файл буде розташований у каталозі, зазначеному в атрибуті codebase, якщо він присутній. В іншому разі він перебуватиме в тому самому каталозі, що й сторінка, на якій відображається тег аплету.

Необхідно використовувати відповідний інструмент для декомпіляції байт-коду у вихідний код. Наприклад:

```
C:\>jad.exe input.class
```

```
Розбір input.class... Generating input.jad
```

Ось кілька відповідних інструментів для декомпіляції різних компонентів розширення браузера:

- Java - Jad
- Flash - SWFScan, Flasm/Flare
- Silverlight - .NET Reflector

Якщо аплет упакований у файл JAR, XAP або SWF, можна розпакувати його за допомогою стандартного засобу читання архівів, такого як WinRAR або WinZip. Слід переглянути відповідний вихідний код (починаючи з реалізації методу, що повертає непрозорі дані), щоб зрозуміти, яка обробка виконується.

Необхідно визначити, чи містить аплет будь-які загальнодоступні методи, які можна використовувати для виконання відповідної обробки під час довільного введення.

Якщо це не так, слід змінити джерело аплету, щоб нейтралізувати будь-яку виконувану ним перевірку. Потім можна перекомпілювати вихідний код у вихідний формат файлу за допомогою інструментів компіляції, наданих постачальником.

Для великих клієнтських застосунків часто буває непомірно складно декомпілювати весь застосунок, модифікувати його та перепакувати, не стикаючись із численними помилками. Для цих додатків, як правило, швидше підключити налагоджувач середовища виконання до процесу. JavaSnoor робить

це дуже добре для Java. Silverlight Spy - це вільно доступний інструмент, який дозволяє відстежувати виконання клієнтів Silverlight.

Потрібно знайти ключові функції та значення, що використовуються додатком для управління бізнес-логікою, пов'язаною з безпекою, і помістити брейкпойнти під час виклику цільової функції. Слід змінити аргументи або значення, що повертається, у міру необхідності, щоб вплинути на обхід безпеки. Тестування механізму аутентифікації. Основні питання тестування механізму аутентифікації наведено нижче (відповідно до рис. 2.4):

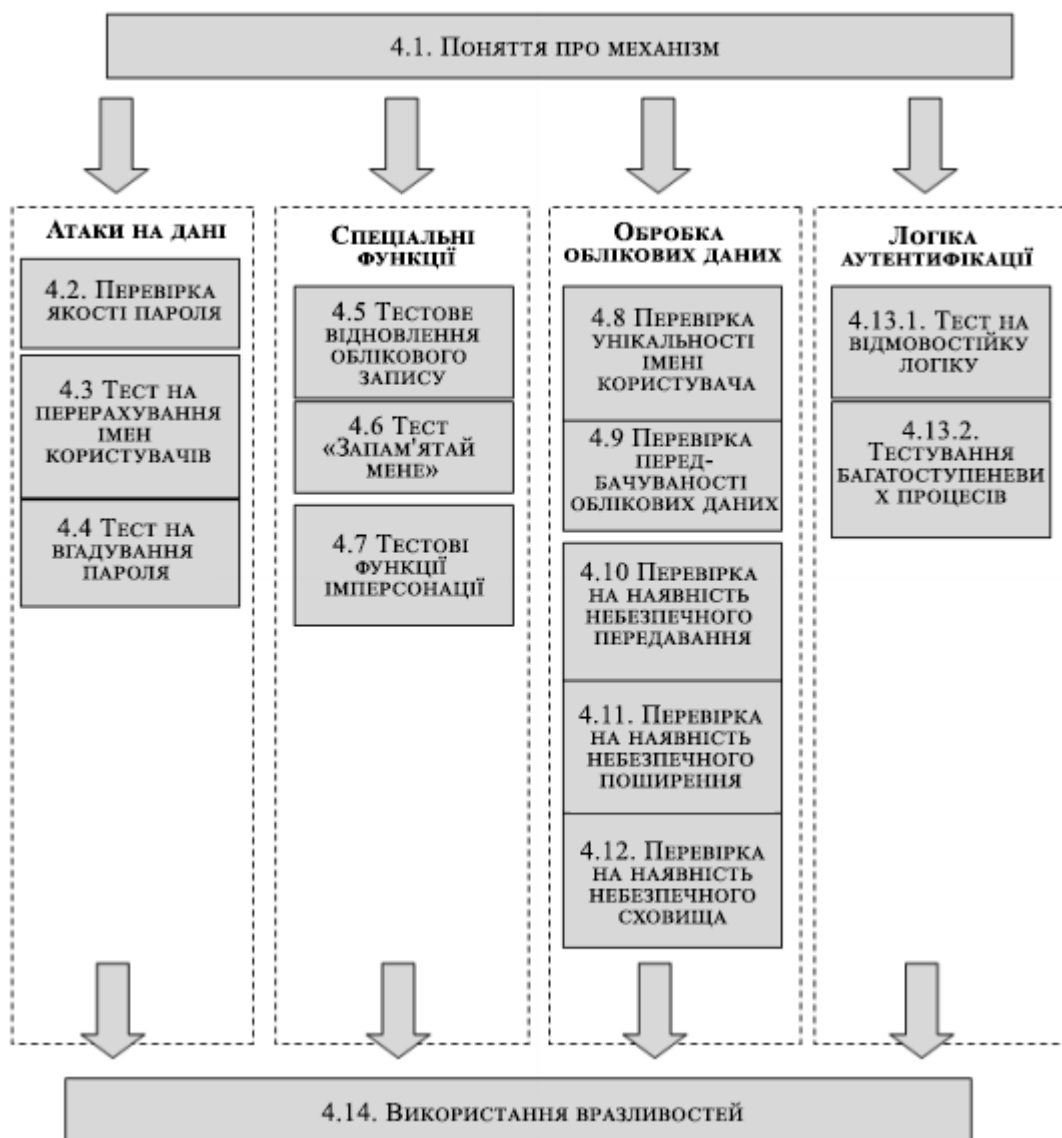


Рисунок 2.4 - Тестування механізму аутентифікації

Поняття про механізм. Слід встановити використовувані технології аутентифікації (наприклад, форми, сертифікати або мультифактор).

Потрібно знайти всі функції, пов'язані з перевіркою автентичності (включно з входом у систему, реєстрацією, відновленням облікового запису тощо).

Якщо додаток не реалізує механізм автоматичної самореєстрації, необхідно визначити, чи існують якісь інші способи отримання кількох облікових записів користувачів.

Перевірка якості пароля. Слід переглянути додаток для будь-якого опису правил мінімальної якості, що застосовуються до паролів користувачів.

Спроба встановити різні види слабких паролів, використовуючи будь-які функції реєстрації або зміни пароля, щоб встановити правила, які фактично застосовуються. Потрібно використовувати короткі паролі, тільки літерні символи, тільки символи в одному регістрі, словникові слова і поточне ім'я користувача.

Перевірка на неповну перевірку облікових даних. Слід установити надійний і складний пароль (наприклад, 12 символів зі змішаними буквами, цифрами, і друкарські знаки). Слід спробувати увійти в систему, використовуючи різні варіанти цього пароля, видаливши останній символ, змінивши регістр символів і видаливши будь-які спеціальні символи. Якщо будь-яка з цих спроб входу в систему виявиться успішною, продовжуйте систематично експериментувати, щоб визначити, яка перевірка насправді виконується.

Встановивши мінімальні правила якості пароля і ступінь перевірки пароля, необхідно визначити діапазон значень, які необхідно буде використовувати для атаки з використанням пароля, щоб мати хорошу ймовірність на успіх. Необхідно спробувати знайти будь-які вбудовані облікові записи, які, можливо, не потрапляли під стандартні вимоги до складності паролів.

Тест на перерахування імен користувачів. Необхідно визначити кожне місце розташування в рамках різних функцій аутентифікації, в яких передається ім'я користувача, зокрема за допомогою поля введення на екрані, поля прихованої форми або файлу cookie. Загальні місця розташування включають основний вхід у систему, реєстрацію, зміну пароля, вихід із системи та відновлення облікового запису.

Для кожного місця розташування слід надіслати два запити, що містять дійсне і недійсне ім'я користувача. Слід переглянути кожну деталь відповідей

сервера на кожен пар запитів, включно з кодом стану HTTP, будь-якими перенаправленнями, інформацією, що відображається на екрані, будь-якими відмінностями, схованими в джерелі HTML-сторінки, і часом, необхідним серверу для відповіді. Потрібно звернути увагу, що деякі відмінності можуть бути незначними (наприклад, одне й те саме повідомлення про помилку може містити незначні типографські відмінності). Можна використовувати функцію історії вашого проксі-сервера перехоплення для перегляду всього трафіку на сервер і з сервера. WebScarab має функцію порівняння двох відповідей, щоб швидко виділити будь-які відмінності між ними.

Якщо буде помічено будь-які відмінності між відповідями, в яких вказано дійсне і недейсне ім'я користувача, повторіть тест з іншою парою значень і переконайтеся в тому, що існує систематична різниця, яка може послужити основою для автоматичного перерахування імен користувачів.

Потрібно перевірити наявність будь-яких інших джерел виток інформації в додатку, які можуть дозволити вам скласти список допустимих імен користувачів. Прикладами можуть слугувати функції ведення журналу, фактичні списки зареєстрованих користувачів і пряма згадка імен або адрес електронної пошти в коментарях до вихідного коду.

Потрібно знайти будь-яку допоміжну аутентифікацію, яка приймає ім'я користувача, і необхідно визначити, чи можна її використовувати для перерахування імені користувача. Слід звернути особливу увагу на сторінку реєстрації, яка дозволяє вказати ім'я користувача.

Перевірка на стійкість до вгадування пароля

Необхідно визначити кожне місце розташування в додатку, в якому передаються облікові дані користувача. Зазвичай двома основними екземплярами є основна функція входу в систему і функція зміни пароля. Остання зазвичай є допустимою метою для атак із вгадуванням пароля, тільки якщо можна вказати довільне ім'я користувача.

У кожному місці, використовуючи обліковий запис, який ви контролюєте, вручну слід надіслати кілька запитів, що містять дійсне ім'я користувача, але інші недейсні облікові дані. Слідкуйте за відповідями програми, щоб виявити будь-які

відмінності. Приблизно після 10 невдалих входів у систему, якщо в застосунку і не повернулося повідомлення про блокування облікового запису, слід надіслати запит, що містить дійсні облікові дані. Якщо цей запит буде виконано успішно, політика блокування облікового запису, ймовірно, не діє.

Якщо не контролюються будь-які облікові записи, необхідно спробувати перерахувати або вгадати дійсне ім'я користувача і зробити кілька недійсних запитів, використовуючи це припущення, відстежуючи будь-які повідомлення про помилки про блокування облікового запису. Звичайно, ви повинні знати, що цей тест може призвести до призупинення або вимкнення облікового запису, що належить іншому користувачеві.

Тестування будь-якої функції відновлення облікового запису

Необхідно визначити, чи містить додаток будь-яку можливість для користувачів відновити контроль над своїм обліковим записом, якщо вони забули свої облікові дані. На це часто вказує посилання "Забули пароль" поруч з основною функцією входу в систему.

Необхідно визначити, як працює функція відновлення облікового запису, виконавши повний огляд процесу відновлення за допомогою облікового запису, який ви контролюєте.

- Якщо функція використовує виклик, такий як секретне запитання, необхідно визначити, чи можуть користувачі встановлювати або вибирати свій власний виклик під час реєстрації. Якщо це так, необхідно використовувати список перерахованих або загальних імен користувачів, щоб зібрати список проблем, і слід переглянути його для будь-яких, які здаються легко вгадуваними.
- Якщо функція використовує підказку пароля, слід виконати ті самі дії, щоб зібрати список підказок пароля і визначити ті, які здаються легко вгадуваними.

Слід виконати ті самі тести для будь-яких завдань відновлення облікового запису, які виконавши в основній функції входу в систему, щоб оцінити вразливість до атак автоматичного вгадування.

Якщо функція передбачає надсилання електронної пошти користувачеві для завершення процесу відновлення, потрібно знайти будь-які слабкі місця, які

можуть дозволити вам узяти під контроль облікові записи інших користувачів. Необхідно визначити, чи можливо контролювати адресу, на яку надсилається електронний лист. Якщо повідомлення містить унікальну URL- адресу відновлення, отримайте кілька повідомлень, використовуючи адресу електронної пошти, яку ви контролюєте, і необхідно спробувати визначити будь-які шаблони, які можуть дозволити вам передбачити URL-адреси, видані іншим користувачам.

Тестування функції "Запам'ятати мене"

Якщо основна функція входу в систему або її допоміжна логіка містить функцію Remember Me, активуйте її і слід переглянути її ефекти. Якщо ця функція дає змогу користувачеві входити в систему в наступних випадках без введення будь-яких облікових даних, ви маєте уважно вивчити її на наявність будь-яких вразливостей.

Уважно потрібно перевірити всі постійні файли cookie, які встановлюються під час активації функції "Remember Me". Слід шукати будь-які дані, які явно ідентифікують користувача або, мабуть, містять будь-який передбачуваний ідентифікатор користувача.

Навіть у тих випадках, коли збережені дані здаються сильно закодованими або заплутаними, уважно вивчіть це та порівняйте результати запам'ятовування кількох дуже схожих імен користувачів і/або паролів, щоб визначити будь-які можливості для зворотного проєктування вихідних даних.

Залежно від ваших результатів, слід змінити вміст вашого файлу cookie відповідним чином, намагаючись замаскуватися під інших користувачів програми

Тестування функції імперсонації

Якщо застосунок містить будь-які явні функціональні можливості, що дають змогу одному користувачеві видавати себе за іншого, уважно потрібно перевірити це на наявність будь-яких вразливостей, які можуть дати змогу вам видавати себе за довільних користувачів без належної авторизації.

Пошук будь-яких наданих користувачем даних, які використовуються для визначення мети імперсонації. Необхідно спробувати маніпулювати цим, щоб видавати себе за інших користувачів, особливо користувачів із правами адміністратора, що може дозволити підвищити привілеї.

Перевірка унікальності імені користувача

Якщо в застосунку є функція реєстрації, що дає змогу вказати бажане ім'я користувача, слід спробувати двічі зареєструвати одне й те саме ім'я користувача з різними паролями. Якщо додаток блокує другу спробу реєстрації, можна використовувати цю поведінку для перерахування зареєстрованих імен користувачів.

Якщо додаток реєструє обидва облікові записи, слід виконати подальші дослідження, щоб визначити його поведінку під час зіткнення імені користувача та пароля. Необхідно спробувати змінити пароль одного з облікових записів, щоб він відповідав паролю іншого. Крім того, слід спробувати зареєструвати два облікові записи з однаковими іменами користувачів і паролями.

Якщо додаток попереджає або видає помилку під час зіткнення імені користувача і пароля, ви, ймовірно, можете використовувати це для виконання автоматичної атаки вгадування, щоб виявити пароль іншого користувача. Необхідно вибрати перераховане або вгадане ім'я користувача і необхідно спробувати створити облікові записи з цим ім'ям користувача та іншими паролями. Коли додаток відхиляє вказаний пароль, ви, ймовірно, знайшли наявний пароль для цільового облікового запису.

Якщо додаток, мабуть, допускає зіткнення імені користувача і пароля без помилок, увійдіть у систему, використовуючи облікові дані, що конфліктують. Необхідно визначити, що відбувається і чи можна використовувати поведінку програми для отримання несанкціонованого доступу до облікових записів інших

користувачів.

Перевірка передбачуваності автоматично згенерованих облікових даних

Якщо програма автоматично генерує імена користувачів або паролі, слід спробувати отримати кілька значень у швидкій послідовності та визначити будь-які виявлені послідовності або шаблони.

Якщо імена користувачів генеруються передбачуваним чином, екстраполуйте їх у зворотному напрямку, щоб отримати список можливих допустимих імен користувачів. Можна використовувати це як основу для автоматичного підбору паролів та інших атак.

Якщо паролі генеруються передбачуваним чином, екстраполуйте шаблон, щоб отримати список можливих паролів, виданих іншим користувачам програми. Це може бути об'єднано з будь-якими списками імен користувачів, які ви отримаєте, щоб виконати атаку на вгадування пароля.

Перевірка на небезпечну передачу облікових даних

Потрібно пройти через усі функції, пов'язані з перевіркою автентичності, які включають передачу облікових даних, включно з основним логіном, реєстрацією облікового запису, зміною пароля та будь-якою сторінкою, яка дає змогу переглядати або оновлювати інформацію про профіль користувача. Контролюйте весь трафік, що проходить в обох напрямках між клієнтом і сервером, використовуючи перехоплюючий проксі-сервер.

Ідентифікувати кожен випадок, у якому облікові дані передаються в будь-якому напрямку. Можна встановити правила перехоплення у своєму проксі-сервері для надсилання повідомлень, що містять певні рядки.

Якщо облікові дані коли-небудь передаються в рядку запиту URL-адреси, вони потенційно вразливі для розкриття в історії браузера, на екрані, у журналах сервера і в заголовку *Referer* під час переходу за посиланнями третіх осіб.

Якщо облікові дані коли-небудь зберігаються у файлі cookie, вони потенційно вразливі для розкриття за допомогою атак XSS або локальних атак на конфіденційність.

Якщо облікові дані коли-небудь передаються з сервера клієнту, вони можуть бути скомпрометовані за допомогою будь-яких вразливостей в

управлінні сеансами або контролі доступу, а також унаслідок атаки XSS.

Якщо облікові дані будь-коли передаються незашифрованим з'єднанням, вони вразливі для перехоплення пристроєм для підслуховування.

Якщо облікові дані передають за протоколом HTTPS, але саму форму входу завантажують за протоколом HTTP, застосунок уразливий для атаки "man-in-the-middle" ("людина посередині"), яку може бути використано для захоплення облікових даних.

Перевірка на небезпечне поширення облікових даних

Якщо облікові записи створюються через будь-який позасмуговий канал або застосунок має функцію самореєстрації, яка сама по собі не визначає всі вихідні облікові дані користувача, слід встановити засоби, за допомогою яких облікові дані поширюються новим користувачам. Поширені методи включають надсилання повідомлення на адресу електронної пошти або поштову адресу.

Якщо додаток генерує URL-адреси активації облікових записів, які поширюються поза діапазоном, слід спробувати зареєструвати кілька нових облікових записів у тісній послідовності та визначити будь-яку послідовність в отриманих вами URL-адресах. Якщо шаблон можна визначити, слід спробувати передбачити URL-адреси, надіслані нещодавнім і майбутнім користувачам, і необхідно спробувати використати ці URL-адреси, щоб стати власником їхніх облікових записів.

Слід спробувати повторно використати одну URL-адресу активації кілька разів і необхідно подивитися, чи дозволяє це додаток. Якщо це не так, слід спробувати заблокувати цільовий обліковий запис до повторного використання URL-адреси, а також необхідно перевірити, чи працює воно як і раніше. Необхідно визначити, чи дозволяє це встановити новий пароль для активного облікового запису.

Перевірка на наявність небезпечного сховища

Якщо було отримано доступ до хешованих паролів, потрібно перевірити наявність облікових записів з однаковим значенням хешованого пароля. Слід спробувати увійти в систему за допомогою загальних паролів для найпоширенішого хешованого значення.

Необхідно використовувати автономну райдужну таблицю для розглянутого алгоритму хешування, щоб відновити значення відкритого тексту.

Тест на логічні помилки

Для кожної функції, у якій застосунок перевіряє облікові дані користувача, включно з функціями зміни логіна та пароля, слід виконати процес звичайним способом, використовуючи обліковий запис, який ви контролюєте. Потрібно звернути увагу на кожен параметр запиту, надісланий у застосунок.

Повторіть процес кілька разів, змінюючи кожен параметр по черзі різними несподіваними способами, призначеними для втручання в логіку програми. Для кожного параметра включіть такі зміни:

Слід надіслати порожній рядок як значення. Видаліть пару ім'я/значення.

Надсилайте дуже довгі та дуже короткі значення. Надсилайте рядки замість чисел, і навпаки.

Надсилайте один і той самий іменований параметр кілька разів з однаковими і різними значеннями.

Уважно вивчіть відповіді додатка на попередні запити. Якщо виникнуть будь-які несподівані розбіжності з базовим випадком, поверніть це спостереження у свою структуру подальших тестових випадків. Якщо одна модифікація спричиняє зміну в поведінці, слід спробувати об'єднати її з іншими змінами, щоб довести логіку додатка до межі.

Перевірка багатоступеневого механізму

Якщо будь-яка функція, пов'язана з перевіркою автентичності, містить у собі надсилання облікових даних у серії різних запитів, необхідно визначити очевидну мету кожного окремого етапу і потрібно звернути увагу на параметри, представлені на кожному етапі.

Повторіть процес кілька разів, змінюючи послідовність запитів таким чином, щоб порушити логіку застосунку, включно з наступними тестами.

Потрібно пройти всі етапи, але в іншій послідовності, ніж передбачалося.

Переходьте безпосередньо до кожного етапу по черзі і продовжуйте нормальну послідовність звідти.

Потрібно пройти через нормальну послідовність кілька разів, пропускаючи

кожен етап по черзі і продовжуючи нормальну послідовність з наступного етапу.

На основі спостережень та очевидної мети кожного етапу механізму слід спробувати подумати про подальші способи зміни послідовності та доступу до різних етапів, які розробники, можливо, не очікували.

Необхідно визначити, чи надіслано будь-яку окрему інформацію (наприклад, ім'я користувача) більш ніж на одному етапі, чи, можливо, її захоплюють більш ніж один раз від користувача, чи через те, що її передають через клієнта в прихованому полі форми, файлі cookie або заданому параметрі рядка запиту. Якщо це так, слід спробувати представити різні значення на різних етапах (як дійсні, так і недійсні) і спостерігайте за ефектом. Необхідно спробувати визначити, чи є відправлений елемент інколи надскладним, чи перевірений на одному етапі, а потім довірений згодом, чи перевірений на різних етапах за допомогою різних перевірок. Необхідно спробувати використати поведінку застосунку для отримання несанкціонованого доступу або зниження ефективності контролю, введеного механізмом.

Слід шукати будь-які дані, що передаються через клієнта, які не були отримані від користувача в будь-який момент. Якщо приховані параметри використовуються для відстеження стану процесу на послідовних етапах, можна втрутитися в логіку програми, змінивши ці параметри різними способами.

Якщо будь-яка частина процесу пов'язана з тим, що застосунок представляє проблему, яка випадково змінюється, потрібно перевірити наявність двох поширених дефектів.

Якщо параметр, що визначає виклик, надсилається разом із відповіддю користувача, необхідно визначити, чи можете ви ефективно вибрати свій власний виклик, змінивши це значення.

Слід спробувати кілька разів перейти до одного й того самого виклику з одним і тим самим ім'ям користувача і визначити, чи представлений інший виклик. Якщо це так, можна ефективно вибрати свій власний виклик, переходячи до цього етапу кілька разів, поки не буде представлено бажаний виклик.

Необхідно використовувати будь-які вразливості для отримання несанкціонованого доступу

Слід переглянути всі виявлені вами вразливості в різних функціях автентифікації та Необхідно визначити всі, які можна використовувати для досягнення своїх цілей під час атаки на застосунок. Зазвичай це включає в себе спробу автентифікації від імені іншого користувача - якщо це можливо, користувача з правами адміністратора.

Перед встановленням будь-якого виду автоматичної атаки зверніть увагу на всі засоби захисту від блокування облікового запису, які ви ідентифікували. Наприклад, під час виконання перерахування імені користувача для функції входу в систему надсилайте загальний пароль із кожним запитом, а не повністю довільне значення, щоб не марнувати даремно невдалу спробу входу в систему на кожне виявлене ім'я користувача. Аналогічно, виконуйте будь-які атаки з вгадуванням пароля. Почніть свій список слів із найпоширеніших слабких паролів і потрібно пройти цим списком, пробуєючи кожен елемент проти кожного перерахованого імені користувача.

Враховуйте правила якості паролів і повноту перевірки паролів під час побудови списків слів для використання в будь-якій атаці з використанням пароля, щоб уникнути неможливих або надскладних тестових випадків.

Необхідно використовувати методи, описані вище, щоб максимально автоматизувати процес, працюйте якомога старанніше і максимізуйте швидкість і ефективність ваших атак.

2.4 Перевірка контролю доступу

Перевірка контролю доступу здійснюється в кілька етапів як показано на рисунку 2.5.

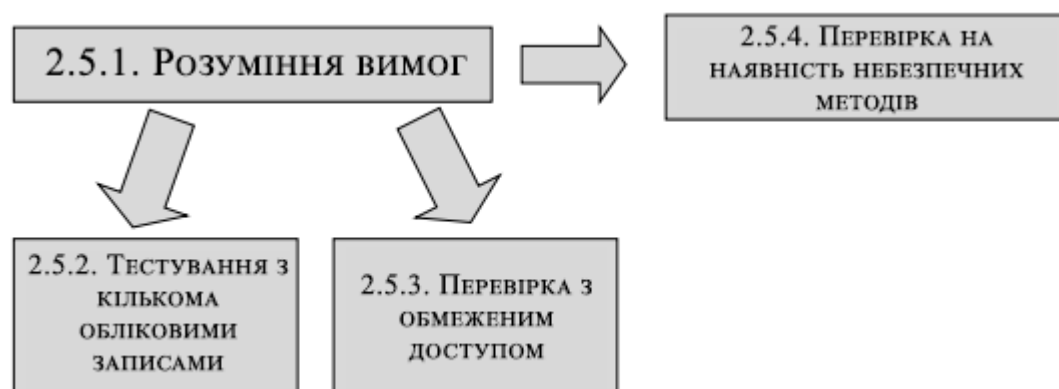


Рисунок 2.5 - Тестування засобів контролю доступу

Розуміння вимог до контролю доступу. Ґрунтуючись на основних функціях, реалізованих у додатку, вивчіть широкі вимоги до контролю доступу з погляду вертикальної сегрегації (різні рівні користувачів мають доступ до різних типів функцій) і горизонтальної сегрегації (користувачі з однаковим рівнем привілеїв мають доступ до різних підмножин даних) Часто присутні обидва типи сегрегації. Наприклад, звичайні користувачі можуть мати доступ до своїх власних даних, тоді як адміністратори можуть отримати доступ до даних усіх користувачів.

Слід переглянути результати зіставлення застосунків, щоб визначити області функціональності та типи ресурсів даних, які являють собою найефективніші цілі для атак із підвищенням привілеїв.

Для найефективнішого тестування вразливостей контролю доступу в ідеалі необхідно отримати кілька різних облікових записів із різними вертикальними та горизонтальними привілеями. Якщо можлива самотійна реєстрація, ви, ймовірно, можете отримати останню безпосередньо з програми. Щоб отримати перший, вам, імовірно, буде потрібна співпраця власника застосунку (або вам потрібно буде використати деяку вразливість, щоб отримати доступ до високо привілейованого рахунку). Доступність різних типів облікових записів вплине на типи тестування, які можна виконати, як описано нижче.

Тестування з кількома обліковими записами. Якщо додаток застосовує вертикальний поділ привілеїв, спочатку необхідно використовувати привілейований обліковий запис, щоб знайти всі функції, до яких він може отримати доступ. Потім необхідно використовувати менш привілейований обліковий запис і необхідно спробувати отримати доступ до кожного елемента цієї функції.

Використовуючи Burp, переглядайте весь вміст програми в одному користувацькому контексті.

Слід переглянути вміст карти сайту Burp, щоб переконатися, що ви визначили всі функції, які хочете протестувати. Потім необхідно вийти з програми та знову увійти в систему, використовуючи інший користувацький

контекст. Необхідно використовувати контекстне меню, щоб вибрати функцію "compare site maps", щоб визначити, які запити з високим рівнем привілеїв можуть бути доступні користувачеві з нижчим рівнем привілеїв.

Якщо додаток застосовує горизонтальний поділ привілеїв, слід виконати еквівалентний тест з використанням двох різних облікових записів на одному і тому ж рівні привілеїв, намагаючись використовувати один обліковий запис для доступу до даних, що належать іншому обліковому запису. Зазвичай це включає заміну ідентифікатора (наприклад, ідентифікатора документа) у запиті на вказівку ресурсу, що належить іншому користувачеві.

Виконати ручну перевірку логіки управління доступом до ключів.

Для кожного привілею користувача Слід переглянути ресурси, доступні користувачеві. Необхідно спробувати отримати доступ до цих ресурсів із неавторизованого облікового запису користувача, повторивши запит із використанням маркера сеансу неавторизованого користувача.

Під час виконання будь-якого виду перевірки контролю доступу обов'язково перевіряйте кожен крок багатоступеневих функцій окремо, щоб переконатися, що контроль доступу було належним чином реалізовано на кожному етапі, або що застосунок припускає, що користувачі, які отримують доступ на більш пізньому етапі, повинні пройти перевірки безпеки, реалізовані на більш ранніх етапах. Наприклад, якщо адміністративна сторінка, що містить форму, належним чином захищена, потрібно перевірити, чи піддається фактичне надсилання форми також належному контролю доступу.

Перевірка з обмеженим доступом. Якщо у вас немає попереднього доступу до облікових записів із різними рівнями привілеїв або до кількох облікових записів із доступом до різних даних, тестування на наявність порушених елементів керування без доступу не таке просте. Багато поширених вразливостей буде набагато важче виявити, оскільки ви не знаєте імен URL-адрес, ідентифікаторів і параметрів, необхідних для використання слабких місць.

У вправах зі зіставлення додатків, що використовують обліковий запис із низьким рівнем привілеїв, ви, можливо, ідентифікували URL-адреси для привілейованих функцій, таких як адміністративні інтерфейси. Якщо вони не

захищені належним чином, ви, ймовірно, вже знаєте про це.

Слід декомпілювати всі скомпільовані клієнти, які присутні, і витягніть будь-які посилання на функціональність на стороні сервера.

Доступ до більшості даних, що підлягають горизонтальному контролю доступу, здійснюється за допомогою ідентифікатора, такого як номер рахунку або посилання на замовлення. Щоб перевірити, чи ефективний контроль доступу з використанням тільки одного облікового запису, необхідно спробувати вгадати або виявити ідентифікатори, пов'язані з даними інших користувачів. Якщо можливо, створіть серію ідентифікаційних даних у швидкій послідовності (наприклад, шляхом створення кількох нових замовлень). Необхідно спробувати ідентифікувати будь-які шаблони, які можуть дозволити вам передбачити ідентифікатори, видані іншим користувачам. Якщо немає способу генерувати нові ідентифікатори, ви, ймовірно, обмежені аналізом тих, які у вас уже є, і припущенням на цій основі.

Якщо ви знайшли спосіб передбачити ідентифікатори, видані іншим користувачам, необхідно використати методи, щоб організувати автоматичну атаку для збору змістовних даних, що належать іншим користувачам. Необхідно використовувати функцію Вилучення Grep у Burp Intruder, щоб захопити відповідну інформацію з відповідей програми.

Тест на небезпечні методи контролю доступу. Деякі додатки реалізують управління доступом на основі параметрів запиту від самого початку небезпечним способом. Слід шукати такі параметри, як `edit=false` або `access=read` у будь-яких запитах ключів, і змінювати їх відповідно до їхньої очевидної ролі, щоб спробувати втрутитися в логіку управління доступом застосунку.

Деякі додатки засновують рішення з управління доступом на заголовок HTTP Referer. Наприклад, додаток може правильно контролювати доступ до `/admin.jsp` і приймати будь-який запит, що показує це в якості свого Референта. Щоб перевірити цю поведінку, слід виконати деякі привілейовані дії, до яких ви авторизовані, і надсилаєте відсутній або змінений заголовок Referer. Якщо ця зміна призведе до того, що додаток заблокує ваш запит, це може бути небезпечно використання заголовка Referer. Слід виконати ту саму дію, що й

неавторизований користувач, але вкажіть вихідний заголовок Referer і необхідно подивитися, чи вдасться виконати цю дію.

Якщо на сайті дозволено метод HEAD, потрібно перевірити наявність небезпечного контролю доступу до URL-адрес із використанням контейнерів. Слід зробити запит, використовуючи метод HEAD, щоб визначити, чи дозволяє це додаток.

2.5 Визначення вразливостей на основі вхідних даних

Багато важливих категорій вразливостей викликаються несподіваним введенням даних користувачем і можуть з'являтися в будь-якому місці програми. Ефективний спосіб для перевірки застосунку на наявність цих вразливостей - це зіставити кожен параметр із кожним запитом за допомогою набору рядків атаки (як показано на рисунку 2.6).



Рисунок 2.6 - Тестування вразливостей на основі вхідних даних.

Розмиття всіх параметрів запиту. Слід переглянути результати ваших дій зі зіставлення додатків і Необхідно визначити кожен окремий запит клієнта, який надсилає параметри, які обробляються додатком на стороні сервера. Відповідні параметри включають елементи в URL-адресі рядка запиту, параметри в тілі запиту і файли cookie HTTP. Також включити будь-які інші елементи користувацького введення, які, як було помічено, впливають на поведінку програми, такі як заголовки Referer або User-Agent.

Для розмиття параметрів можна використовувати свої власні скрипти або готовий інструмент для розмиття. Наприклад, щоб використовувати Burp Intruder, завантажте кожен запит в інструмент. Простий спосіб зробити це - перехопити запит у Burp проксі-сервері та обрати дію відправити зловмиснику або клацнути правою кнопкою миші на елементі в історії проксі- сервера Burp і

вибрати цей параметр. Використання цієї опції гарантує Burp Intruder із вмістом запиту, а також правильним цільовим хостом і портом. Він також автоматично позначає значення всіх параметрів запиту як позиції корисного навантаження, готові до фаззингу.

Використовуючи вкладку корисні навантаження, слід налаштувати відповідний набір корисних навантажень атаки для пошуку вразливостей у додатку. Можна ввести корисне навантаження вручну, завантажити його з файлу або вибрати один із попередньо встановлених списків корисного навантаження. Розмивання кожного параметра запиту в застосунку зазвичай тягне за собою видачу великої кількості запитів і перевірку результатів на наявність аномалій. Якщо ваш набір рядків атаки занадто великий, це може бути контрпродуктивно і призвести до надмірно великого обсягу вихідних даних для перегляду. Отже, розумний підхід полягає в тому, щоб націлитися на низку поширених вразливостей, які часто можна легко виявити в аномальних відповідях на конкретно оброблені вхідні дані, і які часто проявляються в будь-якому місці програми, а не в конкретних типах функцій. Ось відповідний набір корисних навантажень, які можна використовувати для тестування деяких поширених категорій вразливостей:

SQ L-

ін'єкція '

'--

'; waitfor delay '0:30:0'-- 1;

waitfordelay '0:30:0'- 1;

waitfordelay '0:30:0'-

XSS та Ін'єкція заголовка xss test"><script>a lert('xss')</script>

Впровадження команд ОС

|| ping -i 30 127.0.0.1 ; x || ping -n 30 127.0.0.1 &

| ping -i 30 127.0.0.1 |

| ping -n 30 127.0.0.1 |

& ping -i 30 127.0.0.1 &

```
& ping -n 30 127.0.0.1 &  
; ping 127.0.0.1 ;  
%0a ping -i 30 127.0.0.1 %0a  
` ping  
127.0.0.1 `
```

Обхід Шляху

```
../../../../../../../../../../../../etc/passwd  
../../../../../../../../../../../../boot.ini  
../../../../../../../../../../../../etc\passwd  
../../../../../../../../../../../../boot.ini
```

Ін'єкція скрипта

```
;echo 111111  
echo 111111  
response.write 111111  
:response.write 111111
```

Увімкнення файлу `http://<your server name>/ http://<nonexistent IPaddress>/`

Усі попередні корисні навантаження показано в їхньому буквальному вигляді. Персонажі ?, ;, &, +, =, і простір має бути закодовано URL-адресою, тому що вони мають особливе значення в HTTP-запитах. За замовчуванням Burp Intruder виконує необхідне кодування цих символів, тож слід переконатися, що цю опцію не увімкнено. (Щоб відновити всі параметри за замовчуванням після попереднього налаштування, необхідно вибрати Burp ' Відновити значення за замовчуванням).

У функції Grep Burp Intruder налаштуйте відповідний набір рядків для видалення деяких поширених повідомлень про помилки у відповідях. Наприклад:

п о м и л к а

в и н я т о к

н е л е г

а л ь н и й

недійсний

збійфайл

каталогу

дост

упу

до

стекане

знайдено

varchar

ODBC

SQ

L

SELECT

1

1

1

1

1

1

Потрібно звернути увагу, що рядок 111111 увімкнено для перевірки успішних атак впровадження скриптів. Корисні навантаження на кроці 7.1.3

включають запис цього значення у відповідь сервера.

Також необхідно вибрати опцію Payload Grep, щоб відгукуватися на відповіді, що містять саме корисне навантаження, вказуючи на потенційну вразливість XSS або ін'єкції заголовка.

Слід налаштувати веб-сервер або прослуховувач netcat на хості, зазначеному в корисному навантаженні включення першого файлу. Це допоможе вам стежити за спробами підключення, отриманими від сервера в результаті успішної атаки віддаленого включення файлу.

Почати атаку. Коли атаку буде завершено, слід переглянути результати на предмет аномальних відповідей, що вказують на наявність вразливостей. Потрібно перевірити наявність розбіжностей у коді стану HTTP, довжині відповіді, часі відповіді, появі налаштованих виразів і появі найкориснішого навантаження. Можна клацнути заголовок кожного стовпця в таблиці результатів, щоб впорядкувати результати за значеннями в цьому стовпці (і клацнути правою кнопкою миші, щоб виконати зворотне сортування результатів). Це дає змогу швидко ідентифікувати будь-які аномалії, які виділяються на тлі інших результатів. Для кожної потенційної вразливості, зазначеної в результатах тестування fuzz, зверніться до наступних розділів цієї методології. Вони описують докладні кроки, які ви маєте зробити щодо кожної категорії проблем, щоб перевірити наявність уразливості та успішно використовувати її.

Після того, як ви переконали Burp Intruder виконати тест на нечіткість одного запиту, можна швидко повторити той самий тест для інших запитів у додатку. Просто необхідно вибрати кожен цільовий запит у проксі-сервері Burp і необхідно вибрати опцію Надіслати зловмиснику. Потім негайно слід запустити атаку всередині Зловмисника, використовуючи існуючу конфігурацію атаки. Таким чином, ви можете запускати велику кількість тестів одночасно в окремих вікнах і вручну переглядати результати в міру завершення роботи кожного тесту.

Якщо ваші дії з картографування ідентифікували будь-які позасмугові вхідні канали таким чином, що кероване користувачем введення може бути введене в обробку додатка, ви маєте виконати аналогічну вправу з розмивання

на цих вхідних каналах. Надсилайте різні оброблені дані, призначені для запуску поширених вразливостей під час обробки у веб-додатку. Залежно від характеру вхідного каналу для цієї мети може знадобитися створити користувацький сценарій або інший джгут проводів.

На додаток до вашої власної обробки запитів додатків, якщо у вас є доступ до автоматизованого сканера вразливостей веб-додатків, ви маєте запустити це проти цільового додатка, щоб забезпечити основу для порівняння з вашими власними результатами. .

Тест на SQL-ін'єкцію. Якщо рядки атаки SQL, перелічені в кроці 7.1.3, призводять до якихось аномальних відповідей, потрібно перевірити обробку додатком відповідного параметра вручну, щоб визначити, чи є вразливість впровадження SQL.

Якщо були повернуті будь-які повідомлення про помилки бази даних, вивчіть їхнє значення. Необхідно використовувати розділ синтаксис SQL, щоб допомогти інтерпретувати повідомлення про помилки на деяких поширених платформах баз даних.

Якщо надсилання одинарної лапки в параметрі спричиняє помилку або іншу аномальну поведінку, слід надіслати дві одинарні лапки. Якщо це введення призводить до зникнення помилки або аномальної поведінки, застосунок, імовірно, вразливий для ін'єкції SQL.

Слід спробувати використовувати загальні функції конкатенатора рядків SQL для побудови рядка, еквівалентного доброякісному входу. Якщо це викликає ту саму відповідь, що й вихідне доброякісне введення, застосунок, імовірно, уразливий. Наприклад, якщо вихідними вхідними даними є вираз FOO, можна виконати цей тест, використовуючи такі елементи (у третьому прикладі Потрібно звернути увагу на пробіл між двома лапками):

```
'  
|  
| 'F  
OO  
,
```

`+ 'FOO`

`'FOO`

Обов'язково кодуйте символи URL, такі як `+` і пробіл, які мають особливе значення в HTTP-запитах.

Якщо вихідні вхідні дані є числовими, слід спробувати використати математичний вираз, еквівалентний вихідному значенню. Наприклад, якщо вихідне значення було 2, слід спробувати відправити `1+1` або `3-1`. Якщо додаток реагує так само, він може бути вразливим, особливо якщо значення числового виразу чинить систематичний вплив на поведінку програми.

Якщо попередній тест пройшов успішно, можна отримати додаткову впевненість у тому, що вразливість SQL-ін'єкції пов'язана з використанням математичних виразів, специфічних для SQL, для побудови певного значення. Якщо логікою програми можна систематично маніпулювати таким чином, вона майже напевно вразлива для SQL-ін'єкції. Наприклад, обидва наступні елементи еквівалентні числу 2:

`67-ASCII('A')`

`51-ASCII(1)`

Якщо якийсь із тестових випадків fuzz з використанням команди `waitfor` призвів до ненормальної затримки часу до того, як застосунок відповів, це є сильним показником того, що тип бази даних MS-SQL і застосунок є вразливим для SQL-ін'єкцій. Повторіть тест вручну, вказавши різні значення в параметрі `waitfor`, і необхідно визначити, чи систематично час, необхідний для відповіді, залежить від цього значення. Потрібно звернути увагу, що корисне навантаження атаки може бути вставлено в кілька SQL-запитів, тому спостережувана часова затримка може бути фіксованою кратною зазначеному значенню.

Якщо додаток уразливий для впровадження SQL, подумайте, які види атак можливі і, ймовірно, допоможуть вам досягти ваших цілей. Детальні кроки, необхідні для виконання будь-якої з наступних атак

Слід змінити умови в реченні `WHERE`, щоб змінити логіку додатка (наприклад, шляхом ін'єкції або `1=1`-для обходу входу в систему).

Необхідно використовувати оператор `UNION` для введення довільного

запиту SELECT і об'єднання результатів із результатами вихідного запиту програми.

Введіть тип бази даних, використовуючи специфічний для бази даних синтаксис SQL.

Якщо тип бази даних-MS-SQL і застосунок повертає повідомлення про помилки ODBC у своїх відповідях, необхідно використовувати їх для перерахування структури бази даних і вилучення довільних даних.

Якщо не вдається знайти спосіб прямого вилучення результатів із довільного запиту, необхідно використати такі розширені методи для вилучення даних:

- Витягувати рядкові дані в числовій формі по одному байту за раз.
- Необхідно використовувати позасмуговий канал.

Якщо можна викликати різні відповіді додатків на основі однієї довільної умови, необхідно використовувати Absinthe для вилучення довільних даних по одному біту за раз.

- Якщо можна викликати тимчасові затримки на основі однієї довільної умови, необхідно використовувати їх для вилучення даних по одному біту за раз.

- Якщо додаток блокує певні символи або вирази, необхідні для виконання конкретної атаки, слід спробувати різні варіанти. методи обходу, щоб обійти вхідний фільтр.

- За можливості посилюйте атаку на базу даних, використовуючи будь-які вразливості або потужні функції в базі даних.

Тест на XSS та інші відповідні ін'єкції. Ідентифікація відображених параметрів Запиту

Необхідно відсортувати результати тестування fuzz, клацнувши стовпчик Greр корисного навантаження, і Необхідно визначити будь-які збіги, які відповідають корисним навантаженням XSS, переліченим у кроці 7.1.3. Це випадки, коли рядки тесту XSS були повернуті в незмінному вигляді у відповідях застосунку.

У кожному з цих випадків слід переглянути відповідь додатка, щоб знайти місце розташування наданого введення. Якщо це з'являється в тілі відповіді,

потрібно перевірити наявність вразливостей XSS. Якщо вхідні дані відображаються в будь-якому заголовку HTTP, потрібно перевірити для вразливостей ін'єкції заголовка. Якщо вони використовуються в заголовку місця розташування відповіді

302 або використовуються для вказівки перенаправлення будь-яким іншим способом, Потрібно перевірити наявність вразливостей перенаправлення. Потрібно звернути увагу, що одне й те саме введення може бути скопійоване в кілька місць у відповіді та що може бути присутнім більш ніж один тип відображеної вразливості.

Для кожного місця в тілі відповіді, де відображається значення параметра запиту, слід переглянути навколишній HTML-код, щоб визначити можливі способи обробки вашого введення, щоб викликати виконання довільного JavaScript. Наприклад, можна ввести теги `<script>`, впровадити їх в існуючий сценарій або помістити створене значення в атрибут тега.

Необхідно використовувати різні методи відбивання фільтрів на основі сигнатур, як посилання на різні способи, за допомогою яких створене введення може бути використане для виконання JavaScript.

Слід спробувати представити застосунку різні можливі експлойти та відстежувати його відповіді, щоб визначити, чи виконується якась фільтрація або очищення вхідних даних. Якщо ваш рядок атаки повертається без змін, необхідно використовувати браузер, щоб остаточно переконатися, що вам вдалося виконати довільний JavaScript (наприклад, створивши діалогове вікно попередження).

Якщо ви виявите, що додаток блокує введення, що містить певні символи або вирази, які вам необхідно використовувати, або кодує певні символи у форматі HTML, слід спробувати різні обходи фільтра.

Якщо ви виявите вразливість XSS у запиті POST, її однаково можна використати через шкідливий веб-сайт, що містить форму з потрібними параметрами і скрипт для автоматичного надсилання форми. Проте, ширший спектр механізмів доставки атак доступний, якщо експлойт може бути доставлений за допомогою запиту GET. Слід спробувати надіслати ті самі

параметри в запиті GET і необхідно подивитися, чи вдасться атака.

Можна використовувати дію Change Request Method у проксі-сервері Burp для перетворення запиту для вас.

Для кожного місця в заголовках відповідей, де відображається значення параметра запиту, потрібно перевірити, чи приймає додаток дані, що містять кодовані URL-адресою символи повернення carriage-return (%0d) і переведення рядка line-feed (%0a), а також потрібно перевірити, чи повертаються вони в його відповіді без обробки. (Потрібно звернути увагу, що ви шукаєте самі символи нового рядка, які з'являються у відповіді сервера, а не їхні еквіваленти, закодовані в URL-адресі).

Якщо новий рядок з'являється в заголовках відповідей сервера при введенні обробленого введення, значить застосунок вразливий для ін'єкції заголовка HTTP, це може бути використано для виконання різних атак.

Якщо виявиться, що у відповідях сервера повертається тільки один із двох символів у новому рядку, все ще може бути можливо створити робочий експлойт, залежно від контексту і браузера цільового користувача. Якщо ви виявите, що програма блокує введення, що містить символи нового рядка, або очищає ці символи у своїй відповіді, слід спробувати такі

елементи введення для перевірки ефективності фільтра: foo%00%0d%0abar
foo%250d%250abar foo%%0d0d%%0a0abar

Якщо відображені вхідні дані використовуються для вказівки мети будь-якого перенаправлення, потрібно перевірити, чи можна надати оброблені вхідні дані, які призводять до довільного перенаправлення на зовнішній веб-сайт. Якщо це так, то така поведінка може бути використана для надання достовірності атаці в стилі фішингу.

Якщо додаток зазвичай передає абсолютну URL-адресу як значення параметра, слід змінити доменне ім'я в URL-адресі та потрібно перевірити, чи перенаправляє додаток вас на інший домен. Якщо параметр зазвичай містить відносну URL-адресу, слід змінити її на абсолютну URL-адресу для іншого домену і потрібно перевірити, чи перенаправляє додаток вас на цей домен.

Якщо застосунок виконує деяку перевірку параметра перед виконанням

перенаправлення, щоб запобігти зовнішньому перенаправленню, це часто вразливо для обходу. Слід спробувати різні атаки, щоб перевірити надійність фільтрів.

Якщо додаток зберігає елементи введення, надані користувачем, і пізніше відображає їх на екрані, після того, як ви розмили весь додаток, можна спостерігати, як деякі з ваших рядків атаки повертаються у відповідь на запити, які самі по собі не містили цих рядків. Потрібно звернути увагу на будь-які випадки, коли це відбувається, і необхідно визначити вихідну точку входу для збережених даних.

- У деяких випадках надані користувачем дані успішно зберігаються тільки в тому випадку, якщо ви завершуєте багатоступеневий процес, який не виконується під час базового тестування fuzz. Якщо ваші дії зі зіставлення додатків ідентифікують будь-яку функціональність такого роду, потрібно вручну виконати відповідний процес і перевірити збережені дані на наявність вразливостей XSS.

- Якщо у вас є достатній доступ для його тестування, уважно вивчіть будь-яку адміністративну функціональність, у якій дані, отримані від користувачів із низькими привілеями, зрештою відображаються на екрані в сеансі більш привілейованих користувачів. Будь-які збережені вразливості XSS у функціях такого роду зазвичай призводять безпосередньо до підвищення привілеїв.

- Перевірити кожен екземпляр, у якому зберігаються і відображаються надані користувачем дані для користувачів. Потрібно перевірити їх на наявність XSS та інших атак з відповідною ін'єкцією, описаних раніше.

- Якщо ви виявите вразливість, за якої введення, надане одним користувачем, відображається іншим користувачам, необхідно визначити найефективніше корисне навантаження атаки, за допомогою якого можна досягти своїх цілей, як-от захоплення сеансу або підробка запиту. Якщо збережені дані відображаються тільки для того самого користувача, від якого вони були отримані, слід спробувати знайти способи з'єднання будь-яких інших виявлених вразливостей (наприклад, порушених елементів управління доступом), щоб впровадити атаку в сеанси інших користувачів.

- Якщо застосунок дає змогу завантажувати і завантажувати файли, завжди перевіряйте цю функцію на наявність збережених атак XSS. Якщо застосунок допускає HTML, JAR або текстові файли і не перевіряє або не очищає їхній вміст, він майже напевно вразливий. Якщо він допускає файли JPEG і не перевіряє, чи містять вони допустимі зображення, імовірно, застосунок уразливий для атак проти користувачів Internet Explorer. Потрібно перевірити, як додаток обробляє кожен тип файлу, який він підтримує, і підтвердити, як браузері обробляють відповіді, які містять HTML замість звичайного типу контенту.

- У кожному місці, де дані, надані одним користувачем, відображаються іншим користувачам, але де фільтри застосунку не дають змоги вам виконати збережену атаку XSS, потрібно перевірити, чи не робить поведінку застосунку вразливою для підробки запитів на місці. .

Тест на впровадження команд ОС. Якщо будь-який із рядків атаки впровадження команд призвів до ненормальної затримки часу до того, як застосунок відповів, це є сильним показником того, що застосунок є вразливим для впровадження команд ОС. Повторіть тест, вручну вказавши різні значення `v` і або-параметр `n` і Необхідно визначити, чи систематично змінюється час, витрачений на відповідь, залежно від цього значення.

Використовуючи будь-який із рядків введення, який було визнано успішним, слід спробувати ввести більш цікаву команду (наприклад, `ls` або `dir`), а також визначити, чи можете ви отримати результати команди у свій браузер.

Якщо ви не можете отримати результати безпосередньо, вам доступні інші варіанти.

Можна спробувати відкрити позасмуговий канал назад на свій комп'ютер. Слід спробувати використовувати TFTP для копіювання інструментів на сервер, використовуючи telnet або netcat для створення зворотної оболонки назад на комп'ютер і використовуючи команду mail для надсилання вихідних даних команди через SMTP.

Можна перенаправити результати ваших команд у файл у кореневому каталозі веб-сайту, який потім можна отримати безпосередньо за допомогою браузера. Наприклад:

```
dir > c:\inetpub\wwwroot\foo.txt
```

Якщо був знайдений спосіб вводити команди й отримувати результати, слід визначити свій рівень привілеїв (використовуючи `whoami` або аналогічну команду чи намагаючись записати нешкідливий файл у захищений каталог), а потім можна спробувати підвищити привілеї, отримати бекдор-доступ до конфіденційних даних застосунків або атакувати інші хости, доступ до яких можна отримати зі зламаного сервера. Якщо здається, що введення передається будь-якій команді операційної системи, але перелічені рядки атаки не увінчалися успіхом, необхідно подивитися, чи можна використовувати символ `<` або `>` для направлення вмісту файлу на вхід команди або для направлення виведення команди на файл. Це може дозволити читати або записувати довільний вміст файлу. Якщо ви знаєте або можете вгадати фактичну виконувану команду, слід спробувати ввести параметри командного рядка, пов'язані з цією командою, щоб змінити її поведінку прийнятними способами (наприклад, вказавши вихідний файл у кореневому каталозі веб-сайту).

Якщо виявиться, що застосунок екранує певні ключові символи, необхідні для виконання атаки із введенням команд, слід спробувати помістити екрануючий символ перед кожним таким символом. Якщо додаток не екранує сам екрануючий символ, це зазвичай призводить до обходу цього захисного заходу. Якщо ви виявите, що пробіли заблоковані або очищені, можна використовувати `$IFS` замість пробілів на платформах UNIX.

Перевірка проходження шляху. Для кожного виконаного тесту `fuzz` слід переглянути результати, отримані за допомогою рядків атаки обходу шляху, перелічених на кроці 7.1.3. Можна клацнути верхню частину стовпця корисного навантаження в Burp Intruder, щоб впорядкувати таблицю результатів по корисному навантаженню і згрупувати результати для цих рядків. У будь-яких випадках, коли було отримано незвичне повідомлення про помилку або відповідь із ненормальною довжиною, слід переглянути відповідь вручну, щоб визначити, чи містить вона вміст зазначеного файлу або інші докази того, що сталася аномальна операція файлу.

Під час відображення поверхні атаки додатка потрібно відзначити будь-яку функціональність, яка конкретно підтримує читання і запис файлів на основі наданих користувачем вхідних даних. На додаток до загального розмивання всіх параметрів, ви маєте вручну дуже ретельно протестувати цю функціональність, щоб виявити будь-які наявні вразливості під час обходу шляху.

Якщо параметр містить ім'я файлу, частину імені файлу або каталог, слід змінити наявне значення параметра, щоб вставити довільний підкаталог і одну послідовність обходу. Наприклад, якщо додаток надсилає цей параметр:

```
file=foo/file1.txt
```

Слід спробувати надіслати це значення: `file=foo/bar/../file1.txt`

Якщо поведінка програми ідентична в обох випадках, вона може бути вразливою, і вам слід перейти до наступного кроку. Якщо поведінка відрізняється, додаток може блокувати, видаляти або очищати послідовності обходу, що призводить до неприпустимого шляху файлу. Слід спробувати використати кодування та інші атаки, у спробі обійти фільтри.

Якщо попередній тест використання послідовностей обходу в базовому каталозі пройшов успішно, слід спробувати використати додаткові послідовності, щоб піднятися над базовим каталогом і отримати доступ до відомих файлів в операційній системі сервера. Якщо ці спроби не увінчаються успіхом, додаток може вводити різні фільтри або перевірки перед наданням доступу до файлів. Вам слід продовжити дослідження, щоб зрозуміти, які елементи управління реалізовано і чи існують якісь обходи.

Додаток може перевіряти запитуване розширення файлу і дозволяти доступ тільки до певних типів файлів. Слід спробувати використовувати нульовий байт або атака нового рядка разом з відомим прийнятим розширенням файлу в спробі обійти фільтр. Наприклад:

```
../../../../../../boot.ini%00.jpg
```

```
../../../../../../etc/passwd%0a.jpg
```

Додаток може перевіряти, що шлях до файлу, наданий користувачем, починається з певного каталогу або файлу. Слід спробувати додати послідовності обходу після відомого прийнятого стрижня в спробі обійти фільтр.

Наприклад:

```
/images/../../../../../../../../etc/passwd
```

Якщо ці атаки не увінчалися успіхом, слід спробувати об'єднати кілька обходів, працюйте спочатку повністю в базовому каталозі, щоб виявити фільтри і способи, якими додаток обробляє несподівані вхідні дані. Якщо вдасться отримати доступ на читання до довільних файлів на сервері, необхідно спробувати отримати будь-який з наступних файлів, що може дозволити вам посилити атаку.

Файли паролів для операційної системи та програми

Файли конфігурації сервера і додатків для виявлення інших вразливостей або точного налаштування іншої атаки

Увімкнути файли, які можуть містити облікові дані бази даних Джерела даних, що використовуються додатком, такі як файли бази даних MySQL або XML-файли.

Вихідний код на виконуваних на сервері сторінки, щоб виконати перевірку коду в пошуках помилок

Файли журналу додатків, які можуть містити таку інформацію, як імена користувачів і маркери сеансу

Якщо вдасться отримати доступ на запис до довільних файлів на сервері, потрібно перевірити, чи можливі будь-які з наступних атак для ескалації вашої атаки:

Створення сценаріїв у папках запуску користувачів

Зміна файлів, таких як `in.ftpd`, для виконання довільних команд під час наступного підключення користувача

Запис сценаріїв до веб-каталогу з дозволами на виконання і виклик їх із браузера.

Тест на впровадження скрипта. Для кожного виконаного тесту fuzz слід переглянути результати для рядка `111111` (тобто який не передує решті частини тестового рядка), які можна швидко ідентифікувати в Burp Intruder, клацнувши правою кнопкою миші заголовок рядка `111111` Group, щоб згрупувати всі результати, які містять той рядок. Слід шукати ті, у яких немає перевірки в

стовпці Grep корисного навантаження. Будь-які виявлені випадки, ймовірно, будуть уразливі для впровадження команд сценаріїв.

- Слід переглянути всі тестові випадки, в яких використовувалися рядки впровадження скриптів, і Необхідно визначити будь-які повідомлення про помилки сценаріїв, що містять повідомлення про помилки, які можуть вказувати на те, що ваше введення виконується, але викликали помилку. Можливо, їх потрібно буде налаштувати, щоб виконати успішну ін'єкцію скрипта.

Якщо додаток здається вразливим, потрібно перевірити це, ввівши додаткові команди, зазначені у використовуваній платформі сценаріїв. Наприклад, можна використовувати корисні навантаження атаки, аналогічні тим, які використовуються під час фазингу для впровадження команд ОС: `system('ping%20127.0.0.1')`.

Перевірка на ввімкнення файлу. Якщо було отримано будь-які вхідні HTTP-з'єднання від інфраструктури цільового застосунку під час фазингу, застосунок майже напевно є вразливим для віддаленого ввімкнення файлів. Повторіть відповідні тести однопотоким і регульованим за часом способом, щоб точно визначити, які параметри змушують додаток видавати HTTP-запити.

Слід переглянути результати тестів увімкнення файлів і необхідно визначити все, що викликало аномальну затримку у відповіді програми. У цих випадках, можливо, сам застосунок вразливий, але в результаті час очікування HTTP-запитів закінчується через фільтри мережевого рівня.

Якщо буде виявлено вразливість віддаленого включення файлів, розгорніть веб-сервер, що містить шкідливий сценарій, специфічний для мови, на яку ви орієнтуєтесь, і необхідно використовувати команди, подібні до тих, що використовуються для перевірки впровадження сценарію, щоб переконатися, що ваш сценарій виконується.

РОЗДІЛ 3. Проектування сканера для виявлення веб-вразливостей

3.1 Інструменти та технології розробки

Після аналізу проблем і дослідження технічної частини проблеми було вирішено розробити програмний комплекс, що відповідає таким техніко-технологічним вимогам:

- комплекс має бути написаний простою мовою програмування, бажано скриптовою;
- мають бути використані прості в написанні та розумінні структури й моделі;
- необхідно використовувати мікросервісний підхід під час розроблення комплексу;
- за можливості використовувати методи захисту інформації такі як JWT, SSL/TLS;
- взаємодія з користувачем та іншими програмами має здійснюватися через API;
- вхідні та вихідні дані мають подаватися в універсальному вигляді JSON;
- комплекс у пріоритеті повинен розгортатися під управлінням операційної системи LINUX або іншої UNIX подібної

Під простою мовою програмування розуміють мову, легку у вивченні та читанні, поширену в ІТ-середовищі, для максимально дешевого та швидкого розроблення, а також упровадження та супроводу програмного комплексу [10]. Як таку мову програмування запропоновано використовувати мову Python, у зв'язку з її синтаксисом, що легко читається, і поширеністю.

Під простими структурами та моделями розуміють елементи програм, написані за стандартними шаблонами, що дають змогу швидко та просто розробляти програмні засоби, а також легко виправляти помилки на етапі налагодження, через зрозумілу та просту структуру всього проєкту.

Під час передавання даних важливо забезпечувати їхню цілісність і конфіденційність, щоб не допускати витоків і компрометації, у будь-яких її видах. Під час передавання даних через протокол HTTP не гарантується їхня

цілісність, тому що дані не шифрують і передають через усю мережу у відкритому вигляді, що дає змогу проводити атаки ManInTheMiddle, мета яких - отримання вихідних даних, їхнє спотворення або збір. Для цього було розроблено дві технології SSL/TLS, які дають змогу передавати дані за протоколом HTTPS, з асинхронним двостороннім шифруванням. Завдання SSL/TLS з'єднання:

- забезпечення приватності за допомогою шифрування переданих даних
- аутентифікація за допомогою сертифікатів
- забезпечення достовірності даних за допомогою перевірки цілісності даних

Сертифікат відкритого ключа (public key certificate, digital certificate або identity certificate, а також сертифікат, сертифікат ключа підпису, сертифікат ключа перевірки електронного підпису) - електронний документ, що підтверджує приналежність публічного ключа його власнику. Сертифікат видають і підписують Certificate Authorities, які підписують видані сертифікати своїм цифровим підписом, підтверджуючи, що власника сертифіката (і публічного ключа, що міститься в цьому сертифікаті) було перевірено:

- TLS сертифікат, як правило, містить у собі поля VersionNumber: версія сертифіката (див [X.509](#))
- Serial Number: використовується для ідентифікації сертифіката його Certificate Authorities
- Signature Algorithm: алгоритм, використаний для підпису публічного ключа
- Issuer: ким виданий і підписаний сертифікат (Certificate Authority)
- Not Before і Not After: термін дії (валідності) сертифіката
- Subject name: кому належить сертифікат
- Subject Public Key Info
- Public Key Algorithm: алгоритм, використаний для підпису публічного ключа
- Subject Public Key: сам публічний ключ власника сертифіката
- Certificate Signature Algorithm: алгоритм, використаний для підпису

сертифіката

- Certificate Signature: цифровий підпис сертифіката
- Asymmetric encryption (також Public Key Cryptography - криптографія за відкритим ключем) використовує пари ключів - публічний і приватний. Ці ключі пов'язані один з одним таким чином, що дані, зашифровані за допомогою одного ключа (як правило, публічного), можуть бути розшифровані іншим (приватним).

Будь-яке SSL-з'єднання використовує набір протоколів, описаних у RFC 5246 (тут і нижче розглядають TLS v 1.2 і цей RFC, як останню актуальну версію на момент написання цього посту - березень 2018 р.).

У RFC 5246 їх описано у зворотному порядку, але тут розглянемо їх, починаючи з Handshake Protocol, тому що саме в ньому описується процес встановлення сесії та створення ключів, які далі використовуються в TLS Record Protocol.

Структура виходить така:

1. Протоколи хендшейкінгу TLS
2. Протокол рукостискання
3. Змінити протокол Cipher Spec
4. Протокол оповіщення
5. Протокол запису TLS
6. RecordLayer

Веб-токен JSON (JWT) - це відкритий стандарт (RFC 7519) для створення токенів доступу на основі формату JSON. Зазвичай використовується для передавання аутентифікаційних даних у клієнт-серверних додатках. Токени створюються сервером, підписуються секретним ключем і передаються клієнту, який потім використовує цей токен для перевірки своєї особи.

АРТ-токен складається з трьох частин: заголовка, корисного навантаження і підпису або даних шифрування. Перші два елементи - це об'єкти JSON певної структури. Третій елемент обчислюється на основі першого і залежить від обраного алгоритму (у разі використання беззнакового JWT він може бути опущений). Токени можуть бути перекодовані в компактне представлення (JWS / JWE Compact Serialization): алгоритм кодування Base64-URL

застосовують до заголовка і корисного навантаження, після чого додають підпис, і всі три елементи розділяють крапками (".").

Програмний комплекс насамперед має виконувати свій безпосередній функціонал і виправдовувати розробку, а саме:

- проводити тестування інфраструктури на відкриті порти;
- перебирати піддоменні імена сайтів;
- проводити сканування на наявність вразливостей типу SQLi;
- за можливості визначати тип використовуваної ОС;
- за можливості проводити BrutForce (Перебір) паролів, для пошуку ненадійних і поширених паролів.

Програмний комплекс має розроблятися за ітераційним підходом життєвого циклу ПЗ, з можливістю на будь-якому етапі виробництва внести коригування. Приклад ітераційного циклу ПЗ показано на рисунку 3.1:

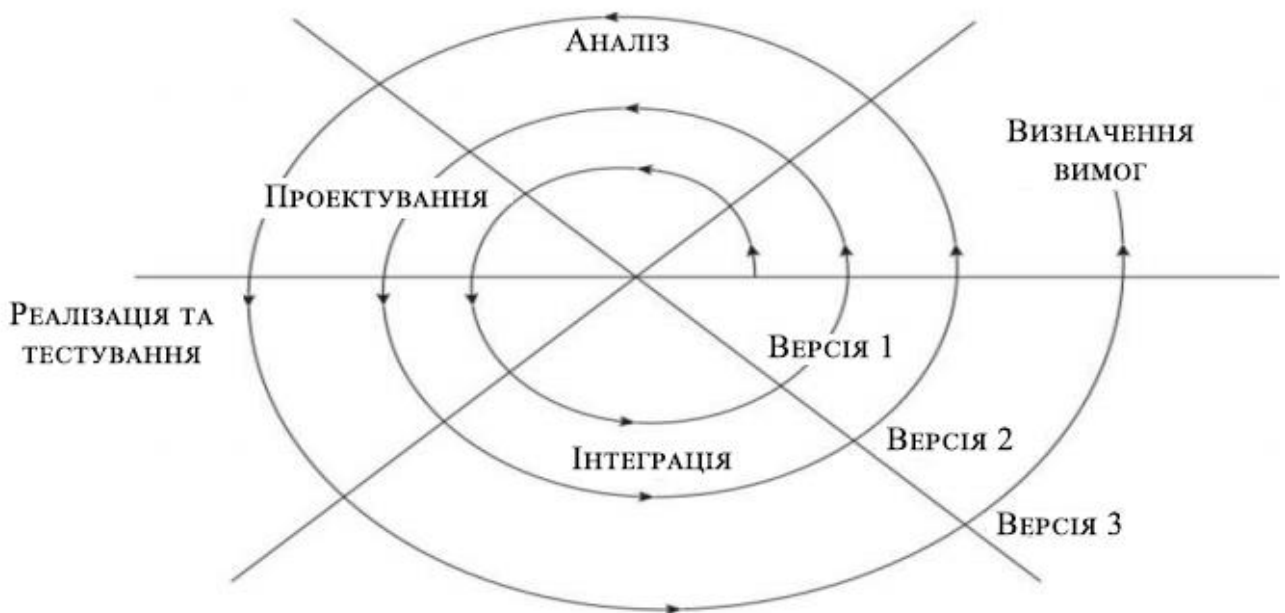


Рисунок 3.1 - Ітераційний життєвий цикл ПЗ

3.2 Алгоритм виявлення вразливостей системи

У магістерській роботі було використано великий перелік технологій, ІС і рішень, який забезпечує правильне і надійне функціонування ПЗ. Список

використовуваних технологій:

- ОС Linux Ubuntu Server 20.04
- графічне середовище робочого столу KDE
- засіб контейнеризації Docker
- веб сервер NGINX
- WSGI сервер JUNICORN
- мова програмування Python
- фреймворк Flask для Python
- JWT
- інтерактивна оболонка BASH

Linux - це сімейство Unix-подібних операційних систем ("ОС"), що використовують ядро Linux, розроблене фінсько-американським програмістом Лінусом Торвальдсом. ОС, що використовують ядро Linux, називаються дистрибутивами Linux, і це ті самі операційні системи, що й Microsoft Windows, Apple macOS, iOS, але з однією дуже важливою особливістю, а саме: їхні вихідні коди відкриті, оскільки їх поширюють за ліцензією GNU GPL, що означає створення вільного та відкритого програмного забезпечення (open - source software). Це означає, що будь-який користувач має право вивчати і змінювати вихідний код.

Вона передає процесору необхідні для обробки вхідні дані і потім, за допомогою апаратного забезпечення комп'ютера, відображає вихідний результат обчислень. Описаний процес є основною функцією операційної системи. Варто зазначити, що вона також виконує і безліч інших не менш важливих завдань (але це вже виходить за рамки цієї статті).

Linux існує навколо нас із середини 1990-х років. Можна зустріти його всюди: у наших телефонах, ноутбуках, у наручних годинниках, у суперкомп'ютерах, автомобілях і навіть у холодильниках. Він здобув популярність, як серед розробників, так і серед звичайних користувачів комп'ютерів. У співтоваристві програмістів існує суперечка про іменування операційних систем, що використовують ядро Linux і програмне забезпечення, розроблене під ліцензією GNU GPL. Оскільки ядро Linux саме по собі не є

працюючою операційною системою, то багато хто воліє використовувати термін "GNU/Linux". Структура Linux наведена на рисунку 3.2:

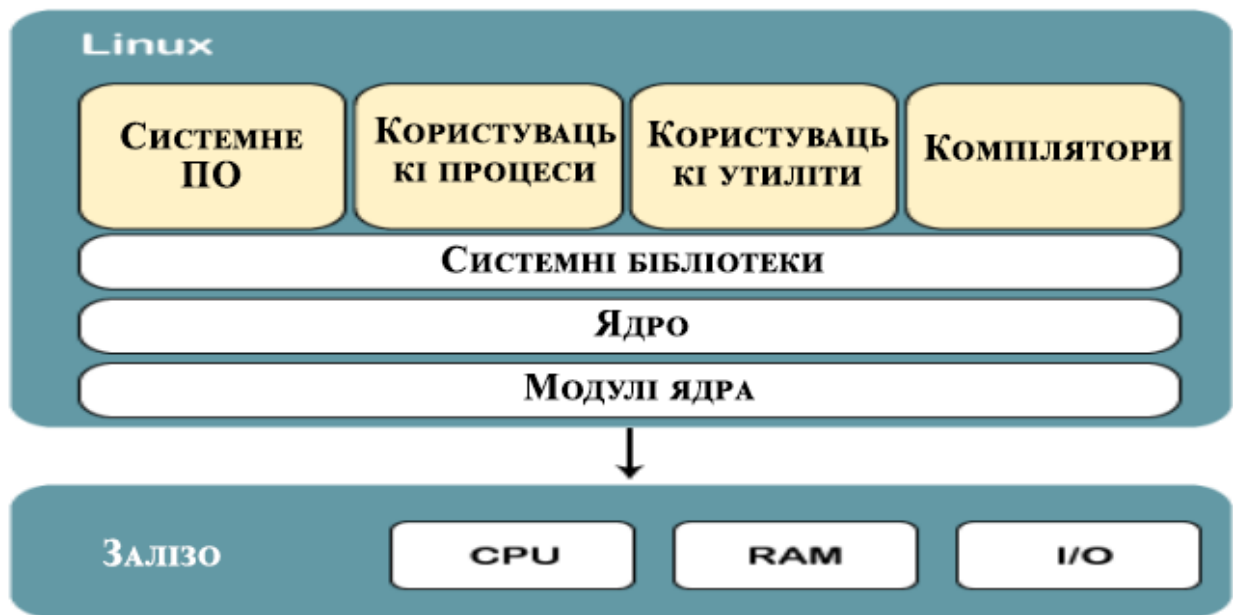


Рисунок 3.2 - Будова Linux

Ядро - це свого роду головна програма, яка є основною частиною операційної системи. Воно виступає в ролі посередника між пристроями комп'ютера (процесором, відеокартою, оперативною пам'яттю і т. д.) і його програмним забезпеченням, абстрагуючи від звичайних програм і користувачів складну, низькорівневу роботу з "залізом" комп'ютера, надаючи натомість простий, зрозумілий і зручний у використанні інтерфейс. Для цього до коду ядра було включено драйвери пристроїв, які можуть, як завантажуватися в пам'ять разом із ядром ОС, так і підключатися в міру виникнення потреби в ресурсах необхідного пристрою.

Мікроядро - це ядро, що складається з декількох незалежних модулів, які завантажуються в пам'ять у міру потреби і виконуються в окремих адресних просторах. По суті, грубо кажучи, у такому варіанті виконання воно не сильно відрізняється від звичайних прикладних програм. До переваг цього ядра можна віднести теоретично більшу надійність порівняно з іншими архітектурами (насправді ж - не все так райдужно і гладко) і його модульність (легкість у підключенні додаткових частин ядра). До мінусів же мікроядерної архітектури належить те, що ядро, побудоване за такою схемою, виходить дуже повільним (адже йому потрібно постійно перемикатися між окремими частинами).

Монолітне ядро - це повна протилежність мікроядра, тому що в пам'яті комп'ютера завжди знаходиться весь (або майже весь) код ядра.

Внаслідок чого, швидкість його роботи вища порівняно з мікроядром [11].

Гібридне ядро - це ядро, що поєднує в собі елементи як монолітної, так і мікроядерної архітектур. Ядро Linux хоч і належить до монолітних ядер, але воно також запозичує і деякі ідеї з мікроядерної архітектури, що означає, що вся операційна система працює в просторі ядра, а драйвери пристроїв (у вигляді модулів) можна легко завантажити (або вивантажити) прямо під час роботи операційної системи. На наступному рисунку 3.3 показано архітектуру системи Linux:

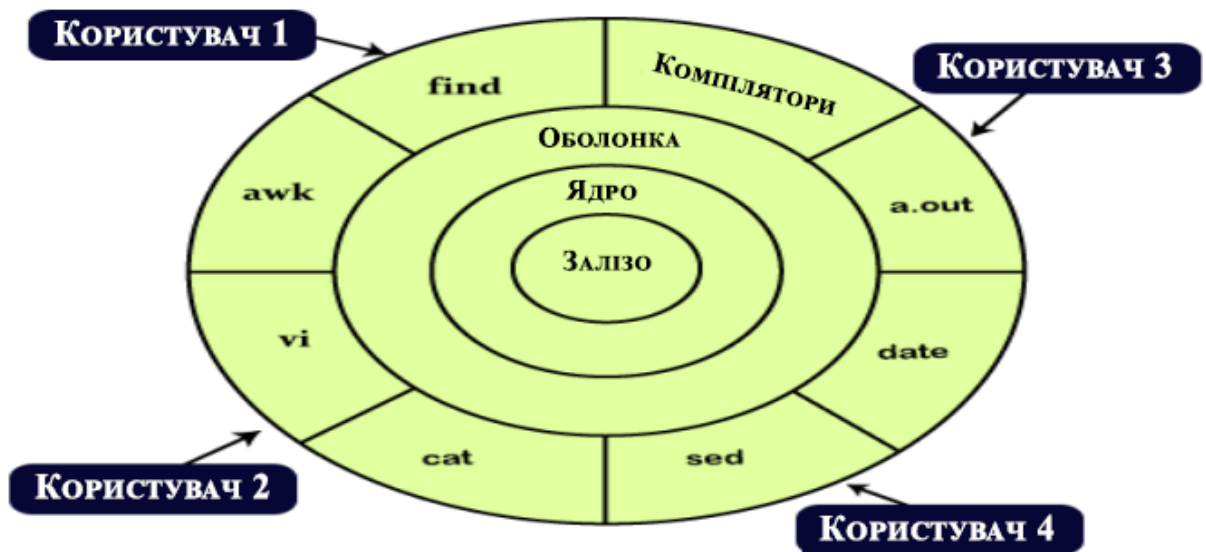


Рисунок 3.3 - Архітектура Linux

"Залізо" - апаратне забезпечення комп'ютера (процесор, відеокарта, оперативна пам'ять тощо) з усіма його периферійними пристроями.

Ядро - є основним компонентом операційної системи, взаємодіє безпосередньо з апаратним забезпеченням, граючи роль посередника між низькорівневим "залізом" і компонентами верхнього рівня.

Оболонка (або "командний інтерпретатор") - інтерфейс для взаємодії між користувачами системи і ядром ОС, що абстрагує внутрішній устрій системи. Приймає команди від користувачів і запускає на виконання відповідні функції.

Утиліти (vi, cat, sed, date тощо) - службові програми, які надають користувачеві більшу частину функціональних можливостей операційної

системи.

KDE - це міжнародне співтовариство, яке розробляє безкоштовне середовище робочого столу KDE Plasma, набір тісно інтегрованих програм для виконання повсякденної роботи, а також кілька веб-сервісів. До початку 2010 року назва KDE означала саме програмне забезпечення (скорочення від K Desktop Environment).

Програмне забезпечення KDE побудовано на основі крос- платформного інструментарію розробки користувацького інтерфейсу Qt. Він працює в основному під UNIX-подібними операційними системами, що використовують X Window System і графічні підсистеми Wayland. KDE 4 частково працював на Microsoft Windows і Mac OS. На даний момент багато додатків KDE доступні як на цих платформах, так і на Android.

Будучи одним із найвідоміших проектів, KDE Plasma є основним настільним середовищем для багатьох дистрибутивів Linux, як-от openSUSE, Manjaro, Mageia, Netrunner, OpenMandriva, Chakra, Kubuntu, KaOS і PCLinuxOS.

KDE Plasma 5-це п'яте і поточне настільне середовище, створене спільнотою KDE для систем Linux. KDE Plasma 5 є наступником KDE Plasma 4 і вперше був випущений в липні 15, 2014. Він містить нову тему за замовчуванням, відому як "Бриз", а також збільшену конвергенцію між різними пристроями. Графічний інтерфейс було повністю перенесено на QML, який використовує OpenGL для апаратного прискорення, що призвело до підвищення продуктивності та зниження енергоспоживання.

LXC - це метод віртуалізації на рівні ОС, призначений для того, щоб запускати безліч ізольованих систем Linux на одному хості.

Контейнери відокремлюють додатки від операційних систем. Це означає, що користувачі мають чисту мінімальну Linux ОС, і можна запускати всі процеси в одному або декількох ізольованих контейнерах.

Оскільки операційна система відокремлена від контейнерів, можна переміщати контейнер на будь-який Linux-сервер, який підтримує операційне середовище контейнера.

Docker - це відкрита платформа для розробки, доставки та експлуатації

додатків. Docker, який почався як проєкт, щоб будувати LXC-контейнери під один застосунок, серйозно змінив LXC і зробив контейнери портативнішими та гнучкішими.

Використовуючи контейнери Docker, можна розгортати, копіювати, переносити і робити резервні копії інформації швидше і легше, ніж за допомогою віртуальної машини. У принципі Docker привносить хмароподібну гнучкість у будь-яку інфраструктуру, яка може працювати на контейнерах.

Хоча Docker почався як проєкт із відкритим кодом для того, щоб будувати спеціалізовану LXC, він пізніше перетворився на власне контейнерне середовище виконання. Docker - це інструмент Linux, який ефективно створює, відправляє та запускає контейнери.

Контейнери Docker і LXC - легковагі механізми віртуалізації в призначеному для користувача просторі, які застосовують контрольні групи і простори імен, щоб керувати ізолюванням ресурсів. Але між Docker і LXC є кілька фундаментальних відмінностей - розглянемо їх на рисунку 3.4

VM	Docker
АПАРАТНА ВІРТУАЛІЗАЦІЯ	ВІРТУАЛІЗАЦІЯ НА РІВНІ ОПЕРАЦІЙНОЇ СИСТЕМИ
Кожній машині відповідає окрема ОПЕРАЦІЙНА СИСТЕМА	КОНТЕЙНЕРИ ПРАЦЮЮТЬ ЯК ОКРЕМИЙ ПРОЦЕС НА ХОСТІ, ПОДІЛЯЮЧИ ЯДРО
Є МОЖЛИВІСТЬ РОЗПОДІЛИТИ МІЖ СОБОЮ РЕСУРСИ ХОСТА: ПРОЦЕСОР, ОПЕРАТИВНУ ПАМ'ЯТЬ, ДИСКОВИЙ ПРОСТІР	ЗА ДОПОМОГОЮ CGROUPS, МОЖНА ОБМЕЖИТИ ОКРЕМО КОЖЕН КОНТЕЙНЕР У СПОЖИВАННІ РЕСУРСІВ
ЛІНІЙНА ВИТРАТА ДИСКОВОГО ПРОСТОРУ, КОЖНІЙ МАШИНІ НЕОБХІДНО МІСЦЕ ДЛЯ ГОСТЬОВОЇ ОПЕРАЦІЙНОЇ СИСТЕМИ	LAYERED FILE SYSTEM ДОЗВОЛЯЄ ЗАОЩАДИТИ МІСЦЕ НА ДИСКУ, ЗА РАХУНОК ПОВТОРНОГО ВИКОРИСТАННЯ ШАРІВ
	ФАЙЛОВУ СИСТЕМУ ОДНОГО ОБРАЗУ, МОЖНА ВИКОРИСТОВУВАТИ ЯК ОСНОВУ ДЛЯ ФОРМУВАННЯ НОВИХ ОБРАЗІВ

Рисунок 3.4 - Різниця між віртуальними машинами і контейнерами

Docker обмежує контейнери, змушуючи їх працювати як єдиний процес. Якщо ваше середовище програми складається з X одночасних процесів, Docker запустить X контейнерів, кожен зі своїм процесом. На відміну від Docker, LXC

контейнери можуть запускати безліч процесів[12].

Щоб запустити простий багаторівневий веб-додаток у Docker, вам знадобиться PHP контейнер, Nginx контейнер (веб-сервер), MySQL контейнер (для процесу бази даних) і кілька контейнерів даних для того, щоб зберігати таблиці баз даних та іншу інформацію програми.

У однопроцесних контейнерів багато переваг, включно з простими та дрібнішими оновленнями. Вам не потрібно вбивати процес баз даних, коли ви хочете оновити тільки веб-сервер. Також у однопроцесних контейнерів ефективна архітектура для того, щоб будувати додатки, засновані на мікросервісах.

У однопроцесних контейнерів також є обмеження. Наприклад, ви не можете запускати агенти, скрипти реєстрації або SSH-процеси, що автоматично запускаються, всередині контейнера. Також нелегко незначно оновлювати однопроцесний контейнер на рівні програми. Вам доведеться запускати новий оновлений контейнер.

Контейнери Docker зроблені так, щоб бути більш безструктурними, ніж LXC:

По-перше, Docker не підтримує зовнішнє сховище. Docker обходить це тим, що дозволяє вам підключати сховище хоста як том Docker з ваших контейнерів. Оскільки томи підключаються, вони не вважаються частиною середовища контейнера.

По-друге, контейнери Docker складаються з шарів у режимі читання. Це означає, що як тільки створюється образ контейнера, він не змінюється. Під час виконання програми, якщо процес у контейнері змінює свій внутрішній стан, створюється різниця між внутрішнім станом і образом, з якого було створено контейнер.

Якщо ви виконуєте команду `docker commit`, різниця між двома версіями стає частиною нового образу - не оригінального, а нового, з якого можна створювати нові контейнери. Якщо ви видалите контейнер, різниця версій зникне.

Безструктурний контейнер - цікава сутність. Можна оновлювати

контейнер, але серія оновлень створить серію нових образів контейнерів, тому в системі так легко відкочуватися.

Мабуть, це найважливіша перевага Docker над LXC. Docker більше відокремлює мережеві ресурси, сховище і деталі ОС, ніж LXC. З Docker додаток дійсно не залежить від налаштувань цих низькорівневих ресурсів. Коли ви переміщуєте контейнер Docker від одного хоста Docker до іншої машини з Docker, Docker гарантує, що середовище для програми залишиться незмінним.

Пряма перевага цього підходу - це те, що Docker допомагає програмістам створювати локальні середовища розробки, які виглядають як продакшн-сервер. Коли програміст закінчує писати і починає тестувати код, він може загорнути його в контейнер, опублікувати безпосередньо на сервері або в приватній хмарі, і він одразу працюватиме, оскільки це одне й те саме середовище.

CLXC програміст може запустити щось на своїй машині, але виявити, що код працює неправильно під час розгортання на сервері. Середовище сервера буде іншим, і програмісту доведеться витратити багато часу, щоб виправити цю різницю та виправити проблему. З Docker цих проблем немає.

Docker складається з декількох компонентів, (представлених на рисунку 3.5)

- . - Docker Daemon - той самий Container Engine; запускає контейнери
- Docker CLI - утиліта з управління Docker
- Dockerfile - інструкція з того, як збирати образ
- Image - образ, з якого розкочується контейнер
- Контейнер
- Docker registry - сховище образів.

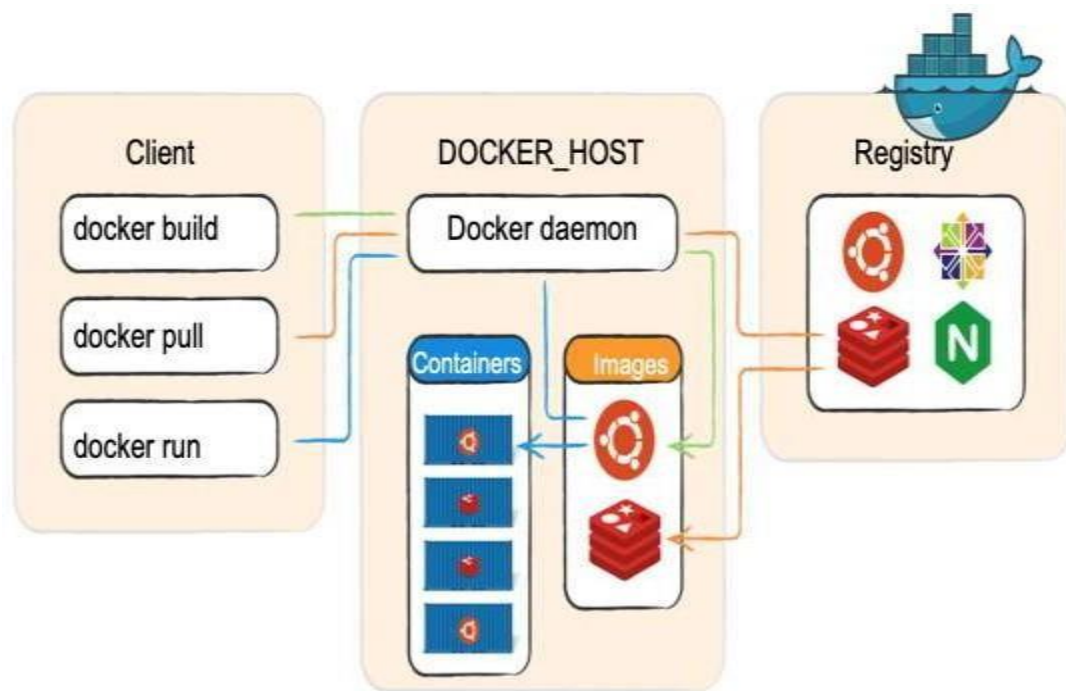


Рисунок 3.5 - Схематичне позначення Docker контейнера

Nginx - потужний інструмент для розгортання веб-сервера, який при правильному налаштуванні перевершує Apache. Галузі застосування Nginx досить широкі - від кешування HTTP до створення інвертованого проксі-сервера.

Наразі Nginx обслуговує приблизно 30, 8% усіх наявних сайтів світу, про чий веб-сервери є інформація у відкритому доступі. Розуміння, що з себе представляє Nginx і як цей програмний продукт можна застосовувати на практиці, допомагає ефективно розв'язувати задачі в багатьох галузях IT-індустрії.

Nginx (NGINX, Engine-X, "Енжін-кс") - це безоплатний веб- і поштовий проксі-сервер із непоточною (асинхронною) архітектурою і відкритим кодом.

Розробку Nginx почали 2002 року для Rambler, а 2004 року став доступний широкому колу користувачів. Через 2 роки в 2011 році серверне ПЗ запустило розширену платну версію продукту (Nginx Plus). Навесні 2019 року Nginx викупив великий американський девелопер F5 Networks.

Nginx працює на ОС Unix-типу і був успішно протестований на OpenBSD, FreeBSD, Linux, Mac OS X, Solaris. На ОС Windows він став доступний після випуску бінарної збірки 0, 7.52.

На даний момент функціоналом користуються такі відомі платформи: Rambler, Begun, Yandex, SourceForge.net, WordPress.com, vkontakte.ru.

Статистика показує, що Nginx використовують 22, 3 млн вебсайтів і 2, 03 млн додаткових активних сайтів.

На відміну від звичайного веб-сервера, Nginx не створює один потік під кожен запит, а розділяє його на менші однотипні структури, звані робочими з'єднаннями. Кожне таке з'єднання обробляється окремим робочим процесом, а після виконання вони зливаються в єдиний блок, що повертає результат в основний процес обробки даних. Одне робоче з'єднання може обробляти до 1024 запитів одного виду одночасно.

Веб-сервер Nginx порівняно з Apache працює швидше під час віддачі статички і споживає менше серверних ресурсів. Його використовують замість або спільно з Apache для прискорення обробки запитів і зменшення навантаження. Це обумовлюється тим, що більша частина тих можливостей, які пропонує Apache, більшості звичайних користувачів не потрібна.

Незважаючи на плюси Nginx і його асинхронного алгоритму, є один мінус - блокуючі операції. Буває так, що серед маленьких операцій, які виконуються паралельно, є одна, яка затримує весь потік. Наприклад, весь потік може чекати відповіді від жорсткого диска для одного з етапів. Це схоже на чергу в касі супермаркету. Якийсь час уся система йде чітко. Одна людина оплачує, друга вже викладає товар на стрічку, касир пробиває товар, паралельно покупець складає пробитий товар у пакет. Раптом на пакеті з помідорами немає штрих-коду. Покупець біжить зважувати. Уся черга стоїть. Процес зупинено.

Для вирішення цієї проблеми у версії Nginx 1.7.11 було створено новий механізм - пул потоків. Коли в процесі потрібно виконати потенційно довгу операцію, вона поміщається в окремий потік, щоб не затримувати всі інші. У нашому прикладі з касою не покупець би побіг зважувати помідори, а працівник супермаркету. Поки він зважував помідори, касир продовжував би пробивати товари, а покупець складати їх.

Раніше традиційний веб-сервер не мав можливості запускати додатки, написані мовою програмування Python. Наприкінці 1990-х було запропоновано модуль для веб-сервера Apache під назвою `mod_python` для виконання довільного коду Python. Протягом кількох років наприкінці 1990-х і початку 2000-х Apache,

налаштований за допомогою `mod_python`, запускав більшість веб-додатків Python. Були й інші засоби для зв'язку веб-сервера і веб-додатка. Але в Python-додатках був один суттєвий недолік - відсутність сумісності. Він був розроблений тільки для одного FastCGI, `mod_Python`, CGI або інших API конкретного веб-сервера, і вони не допускали взаємозамінності. Модуль `Apache mod_Python` не був офіційною специфікацією, і в ньому були проблеми з безпекою. Тут і з'явився WSGI. У нього був стандартний інтерфейс для маршрутизації веб-додатків і фреймворків на веб-сервери. Фреймворк бере свій початок від CGI, або Common Gateway Interface, і використовувався на зорі Інтернету. Успіх CGI був у тому, що він міг працювати з багатьма мовами, але недоліком було те, що він був повільним і обмеженим. Стандарт WSGI, створений Філіпом Дж. Ебі та опублікований 7 грудня 2003 р., є інтерфейсом шлюзу веб-сервера, який пояснює, як веб-сервер взаємодіє з веб-додатками і як додатки можуть бути об'єднані в ланцюжок для генерації запитів.

У стандарту WSGI є низка переваг перед попередниками.

Нижче подано їхній список із поясненням:

- гнучкість. Однією з найбільших переваг, які дає WSGI, є гнучкість. Фактично можна змінити компоненти вебстека, не змінюючи код і навіть не змінюючи додаток, на якому працюють сервери WSGI;

- масштабованість. Фреймворки не можуть обробляти занадто багато запитів. Але сервери WSGI можуть, і вони можуть одночасно обробляти тисячі запитів і направляти їх з веб-сервера найкращим з можливих способів;

- швидкість. Стандарт WSGI допомагає прискорити розробку вебзастосунків на Python, тому що достатньо знати основи роботи інтерфейсу. Багато веб-фреймворків Python постачаються з адаптером WSGI, а серверні технології, як-от Apache, `mod_python`, FastCGI, CGI тощо, можуть запускати WSGI-додаток;

- простота. WSGI простий у вивченні, що полегшує його освоєння, не вимагаючи налаштування або встановлення. Це одна з найбільших переваг WSGI, і розробники щиро підтримують її;

- багаторазове ПЗ. Можна поліпшити функціональність WSGI за

допомогою наявних компонентів проміжного програмного забезпечення, як-от аутентифікація/авторизація, кешування, фільтрація тощо. Функція повторного використання економить час.

У структурі WSGI є дві основні сторони:

- додаток WSGI, створений зі скрипта Python;
- сервер/шлюз WSGI, такий як Apache або Nginx.

Будучи об'єктом, що викликається, додаток-WSGI приймає два параметри. Це: середовище оточення WSGI, назва функції, яка запускає відповідь. Щоразу, коли він отримує запит від HTTP-клієнтів, спрямований на додаток, сервер/шлюз викликає додаток. На рисунку 3.6 показано діаграму виклику функції.

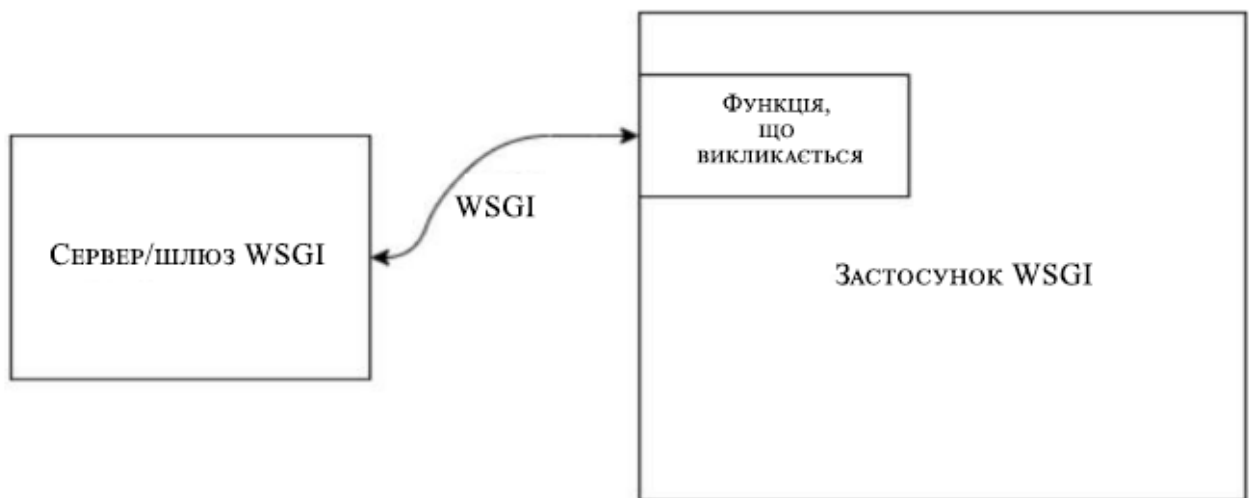


Рисунок 3.6 - Діаграма виклику функції в стандарті WSGI

Шлюз WSGI - це сервер, який запускає веб-додаток і передає відповідну інформацію за допомогою функції зворотного виклику в додаток. Обробка запиту відбувається на стороні програми, тоді як сервер отримує відповідь, використовуючи функцію зворотного виклику. Фреймворками Python, що підтримують WSGI, є Django, web2py, Flask, TurboGears і CherryPy. Між веб-додатком і веб-сервером може бути кілька проміжних програм WSGI. Їхня функція полягатиме в тому, щоб спрямовувати запити до різних об'єктів застосунку, попереднього оброблення контенту, балансування навантаження тощо.

Gunicorn, створений для використання в UNIX, являє собою Python WSGI HTTP Server. Назва є скороченою та комбінованою версією слів "Зелений

Єдиноріг". Сам сайт проекту насправді має зелений єдиноріг. Gunicorn було перенесено з проєкту Unicorn з Ruby. Він відносно швидкий, потребує мало ресурсів, простий у реалізації та працює з різними вебфреймворками. Команда Gunicorn рекомендує вам використовувати Nginx. Сервер Gunicorn працює на локальному порті 8000, а Nginx зазвичай використовується як зворотний проксі-сервер. Gunicorn не має залежностей.

Python - інтерпретована, об'єктно-орієнтована високорівнева мова програмування з динамічною семантикою. Вбудовані високорівневі структури даних у поєднанні з динамічними типізацією і зв'язуванням роблять мову привабливою для швидкого розроблення додатків (RAD, Rapid Application Development). Крім того, її можна використовувати як сценарну мову для зв'язку програмних компонентів. Синтаксис Python простий у вивченні, в ньому надається особливе значення читабельності коду, а це скорочує витрати на супровід програмних продуктів. Python підтримує модулі та пакети, заохочуючи модульність і повторне використання коду. Інтерпретатор Python і велика стандартна бібліотека доступні безкоштовно у вигляді вихідних і виконуваних кодів для всіх основних платформ і можуть вільно поширюватися.

Створення Python було розпочато Гвідо ван Россумом (Guido van Rossum) 1991 року, коли він працював над розподіленою ОС Амеба. Йому була потрібна розширювана мова, яка б забезпечила підтримку системних викликів. За основу було взято ABC і Модула-3. Як назву обрали Python.

Flask є мікрофреймворком для створення вебсайтів мовою Python. В основу статті покладено переклад з офіційної документації Flask. Тому в ній є звернення від першої особи, тобто від творця фреймворку Арміна Ронахера.

Flask у цьому контексті перекладається як порохований ріжок, на це вказує офіційне лого.

Одним із проектних рішень у Flask є те, що прості завдання повинні бути простими; вони не повинні займати багато коду, і це не повинно обмежувати. Flask використовує локальні треди всередині об'єктів, так що ви не повинні передавати об'єкти в межах одного запиту від функції до функції, залишаючись у безпечному треді. Хоч це і дуже простий підхід, який дає змогу заощадити час,

таке рішення може спричинити деякі проблеми для надто великих додатків, оскільки зміни в цих локальних тредах-об'єктах можуть статися де завгодно в цьому треді.

У Flask багато речей попередньо сконфігуровано, на основі загальної базової конфігурації. Наприклад, шаблони та статичні файли збережені в підкаталогах у межах вихідного дерева.

Основна причина чому Flask називається "мікрофреймворком" - це ідея зберегти ядро простим, але розширюваним. У ньому немає абстрактного рівня бази даних, немає валідації форм або всього того, що вже є в інших бібліотеках. Однак, Flask підтримує розширення, які можуть додати необхідну функціональність, і імплементує їх так, ніби вони вже були вбудовані спочатку. Наразі вже є розширення: форми валідації, підтримка завантаження файлів, різні технології автентифікації та багато інших.

Flask захищає вас від найпоширеніших і найвідоміших способів злому, таких як XSS (cross-site scripting). Flask і шаблонізатор Jinja2 захищають, але все одно можуть знайтися способи зламати сайт.

JSON Web Token (JWT) - це відкритий стандарт (RFC 7519) для створення токенів доступу, заснований на форматі JSON. Як правило, використовується для передачі даних для автентифікації в клієнт- серверних додатках. Токени створюються сервером, підписуються секретним ключем і передаються клієнту, який надалі використовує цей токен для підтвердження своєї особи.

У простому розумінні - це рядок у спеціальному форматі, який містить дані, наприклад, ID та ім'я зареєстрованого користувача. Він передається під час кожного запиту на сервер, коли необхідно ідентифікувати і зрозуміти, хто надіслав цей запит.

Токен розділений на три основні групи: заголовок, корисні дані та сигнатура, розділені між собою крапкою. Наочно структуру JWT заголовок показано на рисунку 3.7:

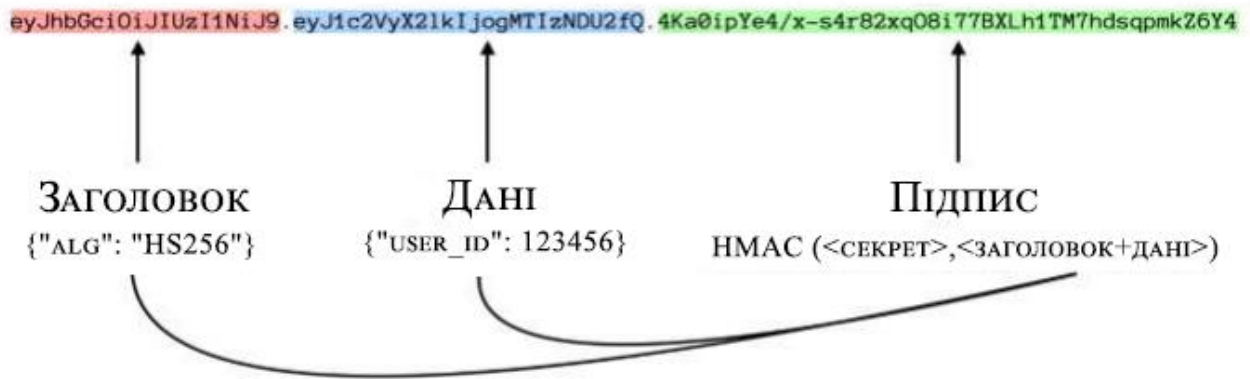


Рисунок 3.7 - Структура JWT Токена

Після того, як відвідувач пройшов авторизацію в нашій системі, вказавши свій логін і пароль, система видає йому 2 токени: access token і refresh токен. Після чого відвідувач, коли хоче отримати з сервера дані, наприклад, свій профіль, разом із запитом він передає Access токен, як на прикладі вище. Сервер, отримавши його, перевіряє, що він дійсний (про це трохи нижче), вчитує корисні дані з нього (той самий user_id) і, таким чином, може ідентифікувати користувача.

3.3 Інтерфейс взаємодії користувача з програмою

Важливу роль у роботі програмного комплексу відіграє продумана структура взаємозв'язку програмних компонентів, у якій чітко прописані потоки даних, структура ІС [13]. Так само цей розділ критично необхідний при впровадженні та відловлюванні помилок, адже за структурною схемою найпростіше зрозуміти ключові моменти, основні сервіси та місця, де можуть виникати проблеми і конфлікти. На рисунку 3.8 представлено структуру одного контейнера Docker, що містить у собі один екземпляр вебсервера NGINX,



UWSGI сервера, а також сам програмний код на python, з фреймворком Flask.

Рисунок 3.8 - Структурна схема контейнера Docker

Більш детально розглянемо взаємодію NGINX і UWSGI серверів, як один з ключових аспектів роботи мікросервісу на Python. Обидва сервіси один з одним функціонують не через протокол HTTP, а через сокети Unix і спеціальний протокол uwsgi. Сокети забезпечують двосторонній зв'язок типу "точка-точка" між двома процесами [14]. Вони є основними компонентами міжсистемного і міжпроцесного зв'язку. Кожен сокет являє собою кінцеву точку зв'язку, з якою може бути поєднано деяке ім'я. Він має певний тип, і один процес або кілька пов'язаних із ним процесів.

Сокети знаходяться в областях зв'язку (доменах). Домен сокета - це абстракція, яка визначає структуру адресації та набір протоколів. Сокети можуть з'єднуватися тільки з сокетами в тому самому домені. Загалом виділено 23 класи сокетів (див. файл <sys/socket.h>), з яких зазвичай використовуються тільки UNIX-сокети та Інтернет-сокети. Сокети можуть використовуватися для встановлення зв'язку між процесами на окремій системі подібно до інших форм IPC.

Клас сокетів UNIX забезпечує їхній адресний простір для окремої обчислювальної системи. Сокети області UNIX називаються іменами файлів UNIX. Сокети також можна використовувати, щоб організувати зв'язок між процесами на різних системах. Адресний простір сокетів між пов'язаними системами називають доменом Інтернету. Комунікації домену Інтернету використовують стек протоколів TCP/IP.

Протокол uwsgi - це власний протокол, який використовується сервером uWSGI. Це двійковий протокол, який може нести будь-який тип даних. Перші 4 байти пакета uwsgi описують тип даних, що містяться в пакеті [15].

Кожен запит uwsgi генерує відповідь у форматі uwsgi.

Навіть обробники веб-сервера підкоряються цьому правилу, оскільки HTTP-відповідь є допустимим пакетом uwsgi (Необхідно подивитися на модифікатор 1 = 72).

Протокол працює здебільшого через TCP, але головний процес може

прив'язуватися до одноадресної/багатоадресної передачі UDP для вбудованого SNMP-сервера або запитів управління кластером/обміну повідомленнями.

Ведеться робота з підтримки SCTP. Приклад взаємодії показано на рисунку 3.9:



Рисунок 3.9 - Принципова схема взаємодії NGINX і uWSGI

Після визначення терміна і структури контейнера, варто прояснити структуру кластера контейнерів. Уся віртуалізація відбувається в Docker, який апаратними засобами дає змогу створювати безліч

операційних систем, повністю незалежних одна від одної.

Так само Docker створює окрему підмережу, в якій контейнери можуть взаємодіяти один з одним, мережу можна зробити недоступною для основної системи або навпаки зробити контейнери частиною вже наявної фізичної мережі через так звані Bridge адаптери, що імітують фізичні пристрої. Так само шлюзом мережі за замовчуванням буде сам Docker, але за необхідності можна видати роль шлюзу одному з контейнерів, або винести його за межу Docker. У ролі балансувальника і головного шлюзу виконуватиме окремий контейнер з Nginx, UWSGI і Flask, які працюватимуть із кінцевими користувачами, розподілятимуть запити між мікросервісами, ставитимуть потоки в чергу і забезпечуватимуть безпечний обмін даними. Загальна схема наведена на рисунку 3.10:

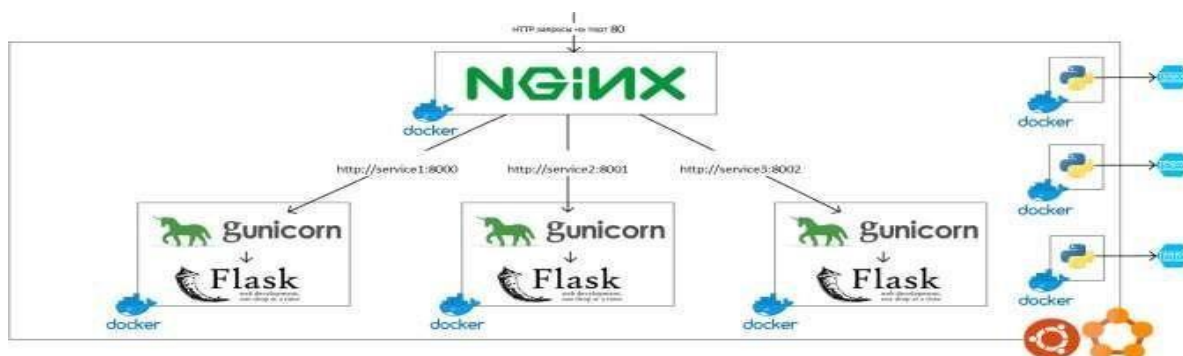


Рисунок 3.10 - Кластер контейнерів Docker Всередині системи Linux

За цими схемами можна проводити "TroubleShootig" проблем, вибудовувати структуру і проводити установку програмного комплексу. Або проводити оновлення контейнерів.

ВИСНОВКИ

У процесі виконання магістерської роботи було досліджено теоретичні та практичні аспекти виявлення вразливостей у веб-додатках, що дозволило досягти поставленої мети — розробити ефективний інструмент для аналізу безпеки веб-додатків та ідентифікації потенційних загроз. Отримані результати мають значне наукове та практичне значення.

Основні висновки дослідження:

1. **Теоретичні основи та актуальність дослідження.** Розглянуто основні поняття та підходи до виявлення вразливостей у веб-додатках. Проаналізовано сучасні методи атак, включно із SQL-ін'єкціями, міжсайтовим скриптингом (XSS) та іншими загрозами. Доведено, що актуальність даного питання визначається зростанням популярності веб-додатків та значними ризиками, пов'язаними з їхньою уразливістю до атак.

2. **Методи дослідження та інструменти.** У роботі було використано комплексний підхід, який поєднує аналітичні, математичні та емпіричні методи. Дослідження базувалося на сучасних алгоритмах аналізу безпеки, що дозволило створити ефективний механізм виявлення загроз.

3. **Розробка сканера веб-вразливостей.** На основі теоретичних досліджень було запропоновано архітектуру сканера, що включає модулі збору даних, ідентифікації вразливостей, обробки результатів та звітування. Реалізація сканера проводилася з використанням сучасних технологій програмування, які забезпечують масштабованість та зручність у використанні.

4. **Результати тестування.** Проведено тестування розробленого інструменту на реальних та симульованих веб-додатках. Отримані результати показали високу ефективність сканера у виявленні поширених вразливостей, таких як SQL-ін'єкції, XSS та CSRF-атаки. Ефективність і точність роботи інструменту підтверджено зменшенням кількості хибнопозитивних результатів порівняно з існуючими рішеннями.

5. **Наукова новизна.** Розроблено новий підхід до виявлення вразливостей, що поєднує існуючі техніки сканування з удосконаленими алгоритмами обробки даних. Це дозволяє підвищити точність аналізу та

зменшити час виявлення загроз.

6. Практичне значення. Результати роботи мають практичне застосування для розробників, адміністраторів та власників веб-додатків. Розроблений інструмент може бути інтегрований у процеси забезпечення інформаційної безпеки для запобігання кібератакам і зменшення ризиків витоку даних.

7. Рекомендації щодо вдосконалення безпеки веб-додатків. Запропоновано шляхи оптимізації процесів взаємодії між державними структурами та бізнесом для підвищення рівня інформаційної безпеки. Особливу увагу приділено розвитку державних програм підтримки малого та середнього підприємництва у сфері кібербезпеки.

Результати цього дослідження підтверджують, що розробка та впровадження сучасних інструментів для аналізу безпеки веб-додатків є важливим кроком у забезпеченні інформаційної безпеки. Запропоновані рішення можуть бути використані для зменшення ризиків, пов'язаних із вразливістю веб-додатків, та сприятимуть подальшому розвитку систем аналізу безпеки. Розроблений сканер демонструє перспективність використання комбінованих підходів у вирішенні завдань кібербезпеки, що може стати основою для майбутніх досліджень у цій галузі.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Damn Vulnerable Web Application (DVWA): Навчальний веб-додаток для тестування безпеки. URL: <https://dvwa.co.uk> (дата звернення: 10.12.2024).
2. Fuzz-тестування. Методика виявлення вразливостей у програмному забезпеченні. EITCA Academy. URL: <https://eitca.org> (дата звернення: 10.12.2024).
3. Kali Linux: Penetration Testing and Ethical Hacking Linux Distribution. URL: <https://www.kali.org> (дата звернення: 10.12.2024).
4. OWASP Top Ten. Актуальні ризики для безпеки веб-додатків. URL: <https://owasp.org> (дата звернення: 10.12.2024).
5. Аналіз вразливостей: забезпечення кібербезпеки в сучасному світі. UA5.org. URL: <https://ua5.org> (дата звернення: 10.12.2024).
6. Безпека веб-додатків: SQL-ін'єкції та інші вразливості. Repository VNAU. URL: <https://repository.vsau.org> (дата звернення: 10.12.2024).
7. Білоконь І. В., Тарасенко М. І. Захист веб-додатків від атак типу XSS. Інноваційні технології в інформаційній безпеці : матеріали наук.-практ. конф. Чернівці : ЧНУ, 2019. С. 87–94.
8. Бондаренко І. М., Сидоренко Л. В. Використання автоматизованих інструментів для аудиту безпеки веб-додатків. Інформаційні системи та технології : матеріали III наук.-практ. конф. Харків : ХНЕУ ім. С. Кузнеця, 2020. С. 75–81.
9. Василенко О. Г., Кузьменко С. В. Інструменти тестування на проникнення: порівняльний аналіз. Проблеми захисту інформації : матеріали наук. конф. Київ : ІПРІ НАН України, 2018. С. 101–108.
10. Глущенко О. І. Методи виявлення вразливостей у веб-додатках за допомогою автоматизованих систем тестування. Захист інформації та кібербезпека : зб. матеріалів доп. учасн. II Міжнар. наук.-практ. конф. Харків : Харківський нац. ун-т, 2020. С. 45–52.
11. Гончарук В. М., Кривошей А. О. Використання машинного навчання для виявлення уразливостей у веб-додатках. Інформаційні системи : зб. наук. праць. Київ : НТУУ "КПІ", 2021. С. 103–109.

12. Дяченко В. С., Мартинюк Г. О. Захист веб-додатків від SQL-ін'єкцій: практичні аспекти. Проблеми інформаційної безпеки : матеріали II Міжнар. конф. Харків : ХНУРЕ, 2018. С. 67–74.
13. Зубченко О. М., Ковальчук Л. П. Використання технологій DevSecOps для зменшення ризиків уразливостей. Системи управління та інформаційні технології : зб. наук. праць. Київ : НУХТ, 2020. С. 76–83.
14. Іванченко Д. О. Технології SQL-ін'єкцій як загроза веб-додаткам. Інформаційна безпека і сучасні технології : зб. наук. праць. Вінниця : ВНТУ, 2019. С. 42–49.
15. Литвиненко П. С., Короленко О. Г. Інструменти оцінки безпеки веб-додатків: порівняння ефективності. Сучасні інформаційні технології : матеріали наук.-практ. конф. Харків : ХНУ ім. В. Н. Каразіна, 2021. С. 120–126.
16. Марченко Ю. С. Особливості застосування Burp Suite для тестування на проникнення. Інформаційні технології у кібербезпеці : зб. тез доп. Міжнар. наук.-практ. конф. Запоріжжя : ЗНУ, 2019. С. 59–64.
17. Мельниченко П. В. Інструменти аналізу коду для виявлення помилок безпеки у веб-додатках. Проблеми інформаційної безпеки : зб. наук. праць. Київ : НТУУ "КПІ", 2018. С. 84–89.
18. Овчаренко В. М., Коваленко І. П. Веб-сканери: сучасні підходи до оцінки вразливостей веб-додатків. Інформаційно-комунікаційні технології : зб. матеріалів VI Міжнар. конф. Одеса : Одеський нац. політехн. ун-т, 2018. С. 105–112.
19. Паламарчук І. П., Лазаренко О. М. Методи аналізу логів для виявлення аномалій у веб-додатках. Інформаційна та мережна безпека : зб. матеріалів конференції. Одеса : Одеський нац. ун-т, 2020. С. 55–61.
20. Петренко М. С. Використання Fuzz-тестування для аналізу веб-додатків. Інформаційна безпека : матеріали III Всеукр. наук.-практ. конф. Львів : ЛНУ імені Івана Франка, 2019. С. 87–93.
21. Посібник з тестування веб-додатків. CiteSchool. URL: <https://citeschool.org> (дата звернення: 10.12.2024).

22. Практичні підходи до виявлення вразливостей в інформаційно-телекомунікаційних мережах. DOAJ. URL: <https://www.doaj.org> (дата звернення: 10.12.2024).
23. Сергієнко Н. О. Аналіз ризиків інформаційної безпеки в сучасних веб-додатках. Проблеми забезпечення кібербезпеки : матеріали V наук.-практ. конф. Дніпро : ДНУ ім. О. Гончара, 2020. С. 67–74.
24. Системи оцінки вразливостей (CVSS). EITCA Academy. URL: <https://eitca.org> (дата звернення: 10.12.2024).
25. Соколова А. В., Черненко І. П. Методи виявлення загроз у веб-додатках на основі аналізу поведінки користувачів. Кібербезпека у цифровому світі : зб. тез доп. конференції. Київ : НАУ, 2019. С. 45–51.
26. Ткаченко І. Б., Мороз В. Л. Використання OWASP ZAP для ідентифікації загроз у веб-додатках. Актуальні проблеми кібербезпеки : зб. тез доп. Міжнар. наук.-практ. конф. Київ : КПП ім. Ігоря Сікорського, 2021. С. 56–62.
27. Управління вразливостями: зменшення ризиків. IBM SkillsBuild. URL: <https://skillsbuild.org> (дата звернення: 10.12.2024).
28. Чумак О. Л., Дяченко П. Ю. Моделі та методи оцінки вразливостей у веб-додатках. Безпека інформаційних систем : зб. матеріалів конференції. Львів : НУ "Львівська політехніка", 2020. С. 42–49.
29. Шевченко Г. С. Методи оцінки ефективності веб-сканерів виявлення вразливостей. Кібербезпека в умовах глобалізації : матеріали Міжнар. наук.-практ. конф. Харків : ХНУРЕ, 2021. С. 123–129.
30. Ярошенко Н. І., Гребенюк С. О. Мультифакторна оцінка ризиків безпеки веб-додатків. Захист інформації : зб. наук. праць. Одеса : ОНУ ім. І. І. Мечникова, 2019. С. 34–40.