Міністерство освіти і науки України
Західноукраїнський національний укніверситет
Факультет комп'ютерних інформаційних технологій
Кафедра кібербезпеки

«Безпека WEB ресурсів»

ОПОРНИЙ КОНСПЕКТ ЛЕКЦІЙ

для студентів спеціальності 125
«Кібербезпека та захист інформації»

Тернопіль

2023

Опорний конспект лекцій з курсу «Безпека WEB ресурсів» для студентів спеціальності 125 «Кібербезпека та захист інформації» – Тернопіль: ЗУНУ, 2023. – 50 с.

Укладачі: Цаволик Т.Г., к.т.н., доцент

Яцків В.В., д.т.н., доцент.

Відповідальний за випуск: Яцків Василь Васильович, д.т.н., доцент, завідувач кафедри кібербезпеки

Розглянуто та схвалено на засіданні кафедри кібербезпеки,

протокол №10 від 11.04.2023

Розглянуто та схвалено групою забезпечення спеціальності кібербезпека,

протокол №4 від 11.04.2023

ЗМІСТ

ВСТУП

В курсі «Безпека WEB ресурсів» ви дізнаєтесь, якими знаннями та практичними навичками повинен володіти спеціаліст, а також технічні та етичні обов'язки, які беруть на себе спеціалісти з тестування безпеки WEB ресурсів. Даний курс забезпечить формування навичок необхідних для досягнення успіху у області безпеки WEB ресурсів.

Зокрема, ви познайомитеся з сучасними методами злому, які використовуються в даний час. Ви також познайомитеся з прийомами, які використовують фахівці з безпеки WEB ресурсів для отримання інформації.

Крім того, розуміння здійснення певних атак на WEB ресурси, може допомогти у виявленні деталей нападу. Насправді фахівцям з безпеки WEB ресурсів потрібно думати, як хакери, щоб запобігти майбутні атаки, а потім використовувати цей досвід для тестування безпеки WEB ресурсів клієнтів.

Опорний конспект з курсу «Безпека WEB ресурсів» забезпечує уявлення про основні відомості щодо атак та захисту WEB ресурсів.

1. Введення в безпеку WEB ресурсів

Що таке безпека сайту?

Інтернет небезпечне місце! Ми регулярно чуємо про те, що веб-сайти стають недоступними через атаки типу відмовлено в обслуговуванні, або відтворення зміненої (і часто пошкодженої) інформації на їхніх сторінках. В інших випадках мільйони паролів, адрес електронної пошти і дані кредитних карт ставали загальнодоступними, піддаючи користувачів веб-сайту особистого збентеження або до фінансових ризиків.

Мета веб-безпеки полягає в запобіганні цих (або інших) видів атак. Більш формальним визначенням веб-безпеки є: способи захисту веб-сайтів від несанкціонованого доступу, використання, зміни, знищення або порушення роботи.

Для ефективної безпеки веб-сайту необхідно приділяти особливу увагу до розробки всього веб-сайту: до вашого веб-додатку, конфігурації веб-сервера, при написанні політик створення та оновлення паролів, а так само коду на стороні клієнта. Хоча все це звучить дуже зловісно, хороша новина полягає в тому, що якщо ви використовуєте веб-фреймворк для серверної частини, то він майже напевно забезпечить «за замовчуванням» надійні і продумані механізми захисту від ряду найбільш поширених атак. Інші атаки можна пом'якшити за допомогою конфігурації вашого веб-сервера, наприклад, включивши HTTPS. Нарешті, є загальнодоступні інструменти для сканування вразливостей, які можуть допомогти вам визначити, якщо ви допустили будь-які очевидні помилки.

Загрози безпеці сайту.

У цьому розділі перераховані лише деякі з найбільш поширених загроз веб-сайту і способи їх усунення. Читаючи, зверніть увагу на те, наскільки успішні загрози, коли веб-додаток довіряє, або недостатньо параноїдально відноситься до даних, що надходять з браузера.

Міжсайтовий скриптинг (XSS).

XSS (Cross-Site Scripting - Міжсайтовий скриптинг) це термін, використовуваний для опису типу атак, які дозволяють зловмисникові впроваджувати шкідливий код через веб-сайт в браузери інших користувачів. Оскільки впроваджений код надходить в браузер з сайту, він є довіреною і може виконувати такі дії, як відправка авторизационного файлу cookieпользователя зловмисникові. Коли у зловмисника є файл cookie, він може увійти на сайт, як якщо б він був користувачем, і зробити все, що може користувач, наприклад, отримати доступ до даних кредитної картки, переглянути контактні дані або змінити паролі.

Примітка: Уразливості XSS історично зустрічалися частіше, ніж будь-які інші види загроз безпеки.

Уразливості XSS діляться на відбиті і збережені, в залежності від того, як сайт повертає впроваджений код в браузер.

Відображена XSS-уразливість виникає, коли для користувача контент, який передається на сервер, негайно і без змін повертається для відображення в браузері. Будь-скрипт в вихідному призначеному для користувача контент запуститься при завантаженні нової сторінки. Наприклад, розглянемо рядок пошуку по сайту, в якій пошукові слова закодовані як параметри URL, і ці слова відображаються разом з результатами. Зловмисник може створити пошукову посилання, яка буде містити шкідливий скрипт в якості параметра (наприклад:
http://mysite.com?q=beer<script%20src="http://evilsite.com/tricky.js"></script >) і переслати його іншому користувачеві по електронній пошті. Якщо цільової користувач клацне по цій «цікавою засланні», то скрипт виконається при відображенні результатів пошуку. Як ми вже говорили, зловмисник таким чином отримує всю інформацію, необхідну йому для входу на сайт як цільовий користувача, потенційного здійснення покупок від імені користувача або отримання його контактної інформації.

Постійна вразливість XSS виникає, коли шкідливий скрипт зберігається на веб-сайті, а потім знову відображається без змін, щоб інші користувачі могли виконувати його мимоволі.

Наприклад, дошка обговорень, яка приймає коментарі, що містять незмінений HTML, може зберігати шкідливий скрипт від зловмисника. Коли коментарі відображаються, скрипт виконується і може відправити зловмисникові інформацію, необхідну для доступу до облікового запису користувача. Атака такого роду надзвичайно популярна і потужна, тому що зловмисник може навіть не мати прямого відношення до жертв.

Хоча дані з запитів POST або GET є найбільш поширеним джерелом вразливостей XSS, будь-які дані з браузера потенційно вразливі, такі як дані cookie, які відображаються браузером, або призначені для користувача файли, які завантажуються і відображаються.

Найкращим захистом від вразливостей XSS є видалення або відключення будь-розмітки, яка потенційно може містити інструкції по запуску коду. Для HTML це включає такі елементи, як <script>, <object>, <embed> і <link>.

Процес зміни призначених для користувача даних, щоб їх не можна було використовувати для запуску сценаріїв або іншим чином впливати на виконання серверного коду, називається очищенням введення. Багато веб-

фреймворки автоматично очищають користувача введення від HTML-форм за замовчуванням.

SQL injection.

Уразливості SQL-ін'єкцій дозволяють зловмисникам виконувати довільний код SQL в базі даних, дозволяючи отримувати, змінювати або видаляти дані незалежно від дозволів користувача. Успішна ін'єкційна атака може підробити посвідчення, створити нові посвідчення з правами адміністратора, отримати доступ до всіх даних на сервері або знищити / змінити дані, щоб зробити їх непридатними для використання.

Типи впровадження SQL включають запровадження SQL на основі помилок, впровадження SQL на основі логічних помилок і впровадження SQL на основі часу.

Ця вразливість є, якщо для користувача введення, який передається в базовий оператор SQL, може змінити зміст оператора. Наприклад, наступний код призначений для перерахування всіх користувачів з певним ім'ям (userName), яке було надано з форми HTML:

```
statement = "SELECT * FROM users WHERE name = '" + userName + "';"
```

Якщо користувач вказує реальне ім'я, оператор буде працювати так, як задумано. Однак зловмисний користувач може повністю змінити поведінку цього оператора SQL на новий оператор в наступному прикладі, просто вказавши текст напівжирним шрифтом для userName.

```
SELECT * FROM users WHERE name = 'a'; DROP TABLE users; SELECT * FROM userinfo WHERE 't' = 't';
```

Модифікований оператор створює дійсний оператор SQL, який видаляє таблицю користувачів і вибирає всі дані з таблиці userinfo (яка розкриває інформацію про кожного користувача). Це працює, тому що перша частина введеного тексту (a ';) завершує вихідне твердження.

Щоб уникнути такого роду атак, ви повинні переконатися, що будь-які призначені для користувача дані, які передаються в запит SQL, не можуть змінити природу запиту. Один із способів зробити це - екранувати всі символи призначеного для користувача введення, які мають особливе значення в SQL.

Примітка. Інструкція SQL обробляє символ 'як початок і кінець строкового літерала. Помістивши зворотну косу риску перед цим символом (\ '), ми екрануючи символ і говоримо SQL замість цього трактувати його як символ (тільки частина рядка).

У наступній інструкції ми екрануючи символ '. Тепер SQL буде інтерпретувати ім'я як весь рядок, виділену жирним шрифтом (це дійсно дуже дивне ім'я, але безпечне).

SELECT * FROM users WHERE name = 'a \'; DROP TABLE users; SELECT * FROM userinfo WHERE \ 't \' = \ 't';

Веб-фреймворки будуть часто піклуватися про зарезервованих символах для вас. Django, наприклад, гарантує, що будь-які призначені для користувача дані, що передаються в набори запитів (модельні запити), будуть екрануються.

Підробка міжсайтових запитів (CSRF)

CSRF-атаки дозволяють зловмиснику виконувати дії, використовуючи облікові дані іншого користувача, без його відома або згоди.

Цей тип атаки найкраще пояснити на прикладі. Джон - зловмисник, який знає, що певний сайт дозволяє користувачам, що ввійшли в систему, відправляти гроші на зазначену обліковий запис, використовуючи HTTP-запит POST, який включає ім'я облікового запису та суму грошей. Джон створює форму, яка включає в себе його банківські реквізити і суму грошей у вигляді прихованих полів, і відправляє її по електронній пошті іншим користувачам сайту (з кнопкою «Надіслати», замаскованої під посилання на сайт «швидкого збагачення»).

Якщо користувач натискає кнопку відправки, на сервер буде відправлений HTTP-запит POST, що містить відомості про транзакції і будь-які файли cookie на стороні клієнта, які браузер пов'язав з сайтом (додавання пов'язаних файлів cookie сайту в запити є нормальною поведінкою браузера). Сервер перевірить файли cookie і використовує їх, щоб визначити, увійшов користувач в систему і чи має дозвіл на здійснення транзакції.

В результаті будь-який користувач, який натискає кнопку Надіслати під час входу на торговий сайт, здійснює транзакцію. Джон стає багатим.

Примітка: Виверт тут в тому, що Джону не потрібен доступ до файлів cookie користувача (або обліковими даними). Браузер користувача зберігає цю інформацію і автоматично включає її в усі запити до відповідного сервера.

Один із способів запобігти цей тип атаки - запросити сервером запити POST, що містять секрет, створений користувачем для конкретного сайту. Секрет буде надано сервером при відправці веб-форми, використовуваної для переказів. Такий підхід не дозволяє Джону створити свою власну форму, тому що він повинен знати секрет, який сервер надає користувачеві. Навіть якщо він дізнається секрет і створить форму для конкретного користувача, він більше не зможе використовувати ту ж форму для атаки на кожного користувача.

Веб-фреймворки часто включають такі механізми запобігання CSRF.

Інші загрози. Інші поширені атаки / уразливості включають:

Clickjacking. У цій атаці зловмисник перехоплює кліки, призначені для видимого сайту верхнього рівня, і направляє їх на приховану нижче сторінку. Цей метод можна використовувати, наприклад, для відображення законного сайту банку, але захоплення облікових даних для входу в невидимий <iframe> (en-US), контрольований зловмисником. Clickjacking також можна використовувати для того, щоб змусити користувача натиснути кнопку на видимому сайті, але при цьому насправді мимоволі натискати зовсім іншу кнопку. Як захист ваш сайт може запобігти вбудовування себе в iframe на іншому сайті, встановивши відповідні заголовки HTTP.

Denial of Service (DoS). DoS зазвичай досягається за рахунок повені цільового сайту підробленими запитами, так що доступ до сайту порушується для законних користувачів. Запити можуть бути просто численними або окремо споживати великі обсяги ресурсів (наприклад, повільне читання або завантаження великих файлів). Захист від DoS зазвичай працює, виявляючи і блокуючи «поганий» трафік, пропускаючи при цьому легітимні повідомлення. Ці засоби захисту зазвичай розташовані перед веб-сервером або на ньому (вони не є частиною самого веб-додатки).

Directory Traversal (Файл і розкриття). У цій атаці зловмисник намагається отримати доступ до частин файлової системи веб-сервера, до яких у нього не повинно бути доступу. Ця вразливість виникає, коли користувач може передавати імена файлів, що містять символи навігації файлової системи (наприклад, ../../). Рішення полягає в тому, щоб очищати введення перед його використанням.

File Inclusion. У цій атаці користувач може "випадково" вказати файл для відображення або виконання в даних, переданих на сервер. Після завантаження цей файл може виконуватися на веб-сервері або на стороні клієнта (що призводить до XSS-атаки). Рішення полягає в тому, щоб дезінфікувати введення перед його використанням.

Впровадження команд. Атаки з впровадженням команд дозволяють зловмиснику виконувати довільні системні команди в операційній системі хоста. Рішення полягає в тому, щоб дезінфікувати вводяться користувачем дані до того, як їх можна буде використовувати в системних викликах.

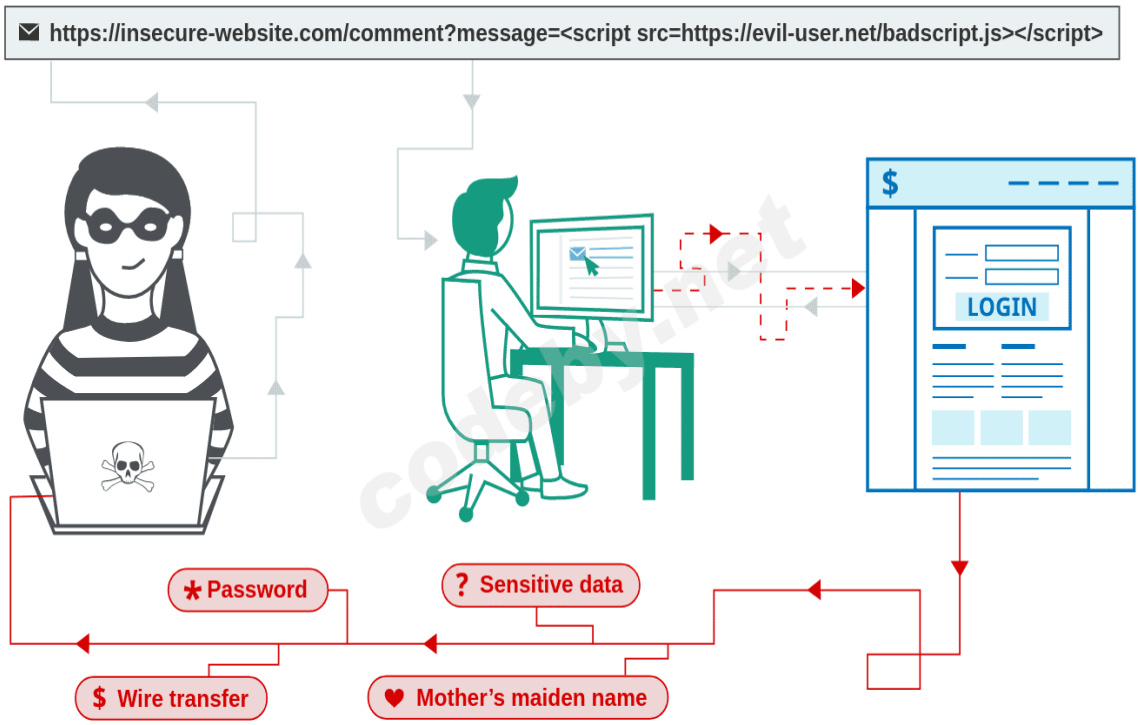
2. Міжсайтові сценарії XSS атаки

Що таке міжсайтовий сценарій (XSS)?

Міжсайтові сценарії (також відомі як XSS) - це вразливість веб-безпеки, яка дозволяє зловмисникові скомпрометувати взаємодію користувачів із вразливою програмою. Це дозволяє зловмисникові обійти одну і ту ж політику походження, яка призначена для відокремлення різних веб-сайтів один від одного. Міжсайтові вразливості сценаріїв зазвичай дозволяють зловмисникові маскуватися під користувача-жертву, виконувати будь-які дії, які користувач може виконати, і отримувати доступ до будь-яких даних користувача. Якщо користувач-жертва має привілейований доступ до програми, зловмисник може отримати повний контроль над усіма функціональними можливостями програми та даними.

Як працює XSS?

Міжсайтовий сценарій працює, маніпулюючи вразливим веб-сайтом, щоб він повертав шкідливий JavaScript користувачам. Коли шкідливий код виконується всередині браузера жертви, зловмисник може повністю скомпрометувати їх взаємодію з додатком.



Які типи XSS-атак?

Існує три основних типи XSS-атак:

- Відбитий XSS, це шкідливий сценарій походить від поточного запиту HTTP.

• Зберігається XSS, це шкідливий сценарій походить із бази даних веб-сайту.

• XSS на основі DOM, ця вразливість існує в коді на стороні клієнта, а не в коді на стороні сервера.

Відбиті міжсайтові сценарії.

Reflected XSS - це найпростіший різновид міжсайтових сценаріїв. Це виникає, коли програма отримує дані в HTTP-запиті та включає ці дані до негайної відповіді небезпечним способом.

Ось простий приклад відбитої вразливості XSS:

https://insecure-website.com/status?message=All+is+well.
Status: All is well.

Додаток не виконує жодної іншої обробки даних, тому зловмисник може легко побудувати атаку наступним чином:

https://insecure-website.com/status?message=<script>/*+Bad+stuff+here...+*/</script>
Status: <script>/* Bad stuff here... */</script>

Якщо користувач відвідує URL-адресу, сконструйовану зловмисником, то сценарій зловмисника виконується в браузері користувача в контексті сеансу цього користувача з додатком. На той момент скрипт може виконувати будь-які дії та отримувати будь-які дані, до яких користувач має доступ.

Вплив відбитих атак XSS.

Якщо зловмисник може керувати сценарієм, який виконується в браузері жертви, то він, як правило, може повністю скомпрометувати цього користувача. Крім усього іншого, зловмисник може:

• Виконайте будь-яку дію в межах програми, яку може виконати користувач.

• Перегляньте будь-яку інформацію, яку користувач може переглядати.

• Змінити будь-яку інформацію, яку користувач може змінити.

• Ініціювати взаємодію з іншими користувачами додатків, включаючи шкідливі атаки, які, здається, походять від початкового користувача-жертви.

Існують різні способи, за допомогою яких зловмисник може спонукати жертву-користувача зробити запит, яким вони керують, для здійснення відображеної атаки XSS. Сюди входить розміщення посилань на веб-сайті, керованому зловмисником, або на іншому веб-сайті, що дозволяє генерувати вміст, або надсиланні посилання в електронному листі, твіті чи іншому повідомленні. Атака може бути націлена безпосередньо на відомого

користувача або може здійснити невибіркову атаку на будь-яких користувачів програми:

Потреба у зовнішньому механізмі доставки для атаки означає, що вплив відбитого XSS, як правило, менш серйозний, ніж збережений XSS, де автономна атака може бути здійснена в межах самої вразливої програми.

Відбитий XSS у різних контекстах.

Існує багато різних різновидів відображених міжсайтових сценаріїв. Розташування відображених даних у відповіді програми визначає, який тип корисного навантаження необхідний для його використання, а також може вплинути на вплив вразливості.

Крім того, якщо додаток виконує будь-яку перевірку або іншу обробку поданих даних до їх відображення, це, як правило, впливає на те, який тип корисного навантаження XSS необхідний.

Як знайти та протестувати виявлені вразливості XSS?

Тестування відображених вразливостей XSS вручну передбачає такі кроки:

- **Перевірте кожну точку входу.** Тестуйте окремо кожну точку входу на наявність даних у HTTP-запитах програми. Сюди входять параметри або інші дані в рядку запиту URL та тілі повідомлення, а також шлях до файлу URL. Він також включає заголовки HTTP, хоча поведінка, схожа на XSS, яка може бути ініційована лише за допомогою певних заголовків HTTP, на практиці не може бути використана.
- **Введіть випадкові буквено-цифрові значення.** Для кожної точки входу подайте унікальне випадкове значення та визначте, чи відображається значення у відповіді. Значення має бути розроблено, щоб пережити більшість перевірок вводу, тому воно має бути досить коротким і містити лише буквено-цифрові символи. Але це повинно бути достатньо довгим, щоб випадкові збіги у відповіді були малоймовірними. Як правило, ідеальним є випадкове буквено-цифрове значення близько 8 символів.
- **Визначте контекст відображення.** Для кожного розташування у відповіді, де відображається випадкове значення, визначте його контекст. Це може бути в тексті між тегами HTML, в атрибуті тегу, який може бути в лапках, у рядку JavaScript тощо.
- **Перевірте корисне навантаження кандидата.** Залежно від контексту відображення, протестуйте початкову корисну навантаження XSS-кандидата, яка ініціюватиме виконання JavaScript, якщо вона відображається немодифікованою у відповіді. Найпростіший спосіб перевірити корисне навантаження - надіслати запит ретранслятору відрижки , змінити запит, щоб вставити корисне навантаження кандидата, оформити запит, а потім переглянути відповідь, щоб перевірити, чи корисне навантаження спрацювало.

Ефективний спосіб роботи - залишити в запиті вихідне випадкове значення та розмістити корисне навантаження XSS-кандидата до або після нього. Потім встановіть випадкове значення як пошуковий термін у поданні відповіді Burp Repeater. Відрижка виділить кожне місце, де з'являється пошуковий термін, дозволяючи швидко знайти відображення.

- **Перевірте альтернативні корисні навантаження.** Якщо корисне навантаження XSS-кандидата було змінено додатком або взагалі заблоковано, то вам доведеться протестувати альтернативні корисні навантаження та техніки, які можуть забезпечити робочу атаку XSS на основі контексту відображення та типу перевірки вводу, що виконується. Докладніше див. У контексті сценаріїв між сайтами

- **Перевірте атаку в браузері.**

Збережені XSS. Що зберігається в міжсайтовому сценарію?

Збережений міжсайтовий сценарій (також відомий як другий порядок або постійний XSS) виникає, коли програма отримує дані з ненадійного джерела та включає ці дані до своїх пізніших відповідей HTTP небезпечним способом.

Припустимо, веб-сайт дозволяє користувачам надсилати коментарі до публікацій в блозі, які відображаються іншим користувачам. Користувачі надсилають коментарі за допомогою HTTP-запиту, як показано нижче:

```
POST /post/comment HTTP/1.1
Host: vulnerable-website.com
Content-Length: 100

postId=3&comment=This+post+was+extremely+helpful.&name=Carlos+Montoya&
email=carlos%40normal-user.net
```

Після того, як цей коментар буде надіслано, будь-який користувач, який відвідає публікацію в блозі, отримає у відповіді програми таке:

```
<p>This post was extremely helpful.</p>
```

Припускаючи, що програма не виконує жодної іншої обробки даних, зловмисник може надіслати такий зловмисний коментар:

```
<script>/* Bad stuff here... */</script>
```

За запитом зловмисника цей коментар буде закодований за URL-адресою:

comment=%3Cscript%3E%2F*%2BBad%2Bstuff%2Bhere...%2B*%2F%3C%2Fscript%3E

Будь-який користувач, який відвідає допис у блозі, тепер отримає у відповіді програми таке:

<script>/* Bad stuff here... */</script>

Потім скрипт, наданий зловмисником, буде виконаний у браузері користувача жертви в контексті їх сеансу з додатком.

Вплив збережених XSS-атак.

Якщо зловмисник може керувати сценарієм, який виконується в браузері жертви, тоді він, як правило, може повністю скомпрометувати цього користувача. Зловмисник може виконати будь-які дії, які стосуються впливу відображених вразливостей XSS.

З точки зору експлуатаційності, ключова відмінність між відображеним та збереженим XSS полягає в тому, що збережена вразливість XSS дозволяє атаки, які містяться в самому додатку. Зловмиснику не потрібно знаходити зовнішній спосіб спонукати інших користувачів робити конкретний запит, що містить їх експлойт. Швидше за все, зловмисник розміщує свій експлойт у самому додатку і просто чекає, поки користувачі зіткнуться з ним.

Автономний характер зберігаються міжсайтових сценаріїв використання особливо актуальний у ситуаціях, коли вразливість XSS зачіпає лише користувачів, які наразі ввійшли до програми. Якщо XSS відображається, тоді атака повинна бути випадково приурочена: користувач, який спонукається зробити запит зловмисника в той час, коли вони не ввійшли в систему, не буде скомпрометований. На відміну від цього, якщо XSS зберігається, то користувач гарантовано входить в систему під час, коли він стикається з експлоїтом.

Зберігається XSS в різному контексті.

Існує багато різних різновидів зберігаються міжсайтових сценаріїв. Розташування збережених даних у відповіді програми визначає, який тип корисного навантаження необхідний для його використання, а також може вплинути на вплив вразливості.

Крім того, якщо додаток виконує будь-яку перевірку чи іншу обробку даних перед тим, як вони зберігаються, або в момент, коли збережені дані включаються у відповіді, це, як правило, впливає на те, який тип корисного навантаження XSS необхідний.

Як знайти та протестувати збережені вразливості XSS?

Тестування на збережені вразливості XSS вручну може бути складним завданням. Вам потрібно протестувати всі відповідні "точки входу", через які дані, що контролюються зловмисником, можуть увійти в обробку програми, і всі "точки виходу", в яких ці дані можуть відображатися у відповідях програми.

Точки входу в обробку заявки включають:

- Параметри або інші дані в рядку запиту URL та тілі повідомлення.
- Шлях до файлу URL.
- Заголовки запитів HTTP, які можуть бути непридатними для використання у відношенні відображеного XSS .
- Будь-які позасмугові маршрути, за допомогою яких зловмисник може доставляти дані в додаток. Існуючі маршрути повністю залежать від функціональних можливостей, реалізованих додатком: програма веб-пошти оброблятиме дані, отримані електронною поштою; програма, що відображає стрічку Twitter, може обробляти дані, що містяться в сторонніх твітах; а агрегатор новин включатиме дані, що надходять з інших веб-сайтів.

Точками виходу для збережених XSS-атак є всі можливі відповіді HTTP, які повертаються будь-якому користувачеві програми в будь-якій ситуації.

Першим кроком у тестуванні на збережені вразливості XSS є пошук зв’язків між точками входу та виходу, завдяки чому дані, подані до точки входу, викидаються з точки виходу. Причини, через які це може бути складним завданням, полягають у тому, що:

- Дані, подані до будь-якої точки входу, в принципі можуть передаватися з будь-якої точки виходу. Наприклад, надані користувачем відображувані імена можуть з'являтися в неясному журналі аудиту, який видно лише деяким користувачам додатків.
- Дані, які в даний час зберігаються програмою, часто вразливі до перезапису через інші дії, що виконуються в програмі. Наприклад, функція пошуку може відображати список останніх пошукових запитів, які швидко замінюються, коли користувачі виконують інші пошукові запити.

Для всебічної ідентифікації зв’язків між точками входу та виходу потрібно було б протестувати кожну перестановку окремо, подати конкретне значення у точку входу, перейти безпосередньо до точки виходу та визначити, чи відображається там значення. Однак такий підхід не є практичним у додатку, що має більше кількох сторінок.

Натомість більш реалістичним підходом є систематична робота через точки введення даних, подання конкретного значення в кожну з них та відстеження реакцій програми для виявлення випадків, коли представлене значення з’являється. Особливу увагу можна приділити відповідним функціям

програми, таким як коментарі до публікацій у блозі. Коли надіслане значення спостерігається у відповіді, вам потрібно визначити, чи справді дані зберігаються у різних запитах, на відміну від простого відображення у негайній відповіді.

Коли ви визначили зв'язки між точками входу та виходу в процесі обробки програми, кожне посилання потрібно спеціально протестувати, щоб виявити, чи є збережена вразливість XSS. Це включає визначення контексту у відповіді, де зберігаються дані, та тестування відповідних корисних навантажень XSS-кандидатів, які застосовуються до цього контексту. На даний момент методологія тестування в цілому така ж, як і для пошуку відображених вразливостей XSS .

Міжсайтові сценарії на основі DOM.

XSS на основі DOM (також відомий як DOM XSS) виникає, коли програма містить певний JavaScript на стороні клієнта, який обробляє дані з ненадійного джерела небезпечним способом, як правило, записуючи дані назад у DOM.

У наступному прикладі програма використовує деякий JavaScript, щоб прочитати значення з поля введення та записати це значення в елемент в HTML:

```
var search = document.getElementById('search').value;
var results = document.getElementById('results');
results.innerHTML = 'You searched for: ' + search;
```

Якщо зловмисник може контролювати значення поля введення, він може легко створити шкідливе значення, яке змушує їх власний сценарій виконуватися:

You searched for: <img src=1 onerror='/* Bad stuff here... */'>

У типовому випадку поле введення заповнюється частиною HTTP-запиту, наприклад, параметром рядка запиту URL-адреси, що дозволяє зловмиснику здійснити атаку, використовуючи шкідливу URL-адресу, таким же чином, як відображено XSS.

Для чого можна використовувати XSS?

Зловмисник, який використовує уразливість сценаріїв між сайтами, як правило, здатний:

- Видати себе за особу-жертву або маскуватися.
- Виконайте будь-які дії, які користувач може виконати.
- Прочитайте будь-які дані, до яких користувач може отримати доступ.
- Захоплення реєстраційних даних користувача.

- Виконайте віртуальну деформацію веб-сайту.
- Введіть на веб-сайт функціональність троянських програм.

Вплив вразливостей XSS.

Фактичний вплив атаки XSS, як правило, залежить від природи програми, її функціональних можливостей та даних, а також від стану скомпрометованого користувача. Наприклад:

- У брошурному додатку, де всі користувачі анонімні, а вся інформація є загальнодоступною, вплив часто буде мінімальним.
- В додатку, що містить конфіденційні дані, такі як банківські операції, електронні листи чи записи в галузі охорони здоров'я, вплив, як правило, буде серйозним.
- Якщо підданий користувачеві привілеї має підвищені повноваження в додатку, тоді вплив, як правило, буде критичним, що дозволить зловмиснику взяти повний контроль над вразливою програмою та скомпрометувати всіх користувачів та їх дані.

Як знайти та протестувати вразливості XSS.

Ручне тестування відображеного та збереженого XSS, як правило, включає подання деяких простих унікальних введень (таких як короткий буквено-цифровий рядок) у кожну точку входу в програмі; визначення кожного місця, куди надісланий вхід повертається у відповідях HTTP; і тестування кожного розташування окремо, щоб визначити, чи можна правильно створити введення для використання довільного JavaScript.

Тестування вручну XSS на основі DOM, що виникає з параметрів URL-адреси, передбачає подібний процес: розміщення простого унікального введення в параметрі, використання інструментів розробника браузера для пошуку DOM для цього введення, і тестування кожного місця, щоб визначити, чи можна його використовувати. Однак інші типи DOM XSS виявити важче. Щоб знайти вразливості, засновані на DOM, у вхідних даних, що не засновані на URL-адресах (таких як document.cookie), або в помилках, що не засновані на HTML (наприклад, setTimeout), немає заміни для перегляду коду JavaScript, що може зайняти дуже багато часу. Веб-сканер вразливості Burp Suite поєднує статичний та динамічний аналіз JavaScript для надійної автоматизації виявлення вразливостей на основі DOM.

Політика безпеки вмісту.

Політика безпеки вмісту (CSP) - це механізм браузера, який спрямований на пом'якшення впливу міжсайтових сценаріїв та деяких інших уразливостей. Якщо програма, яка використовує CSP, містить XSS-подібну поведінку, тоді

CSP може перешкоджати або запобігати використанню вразливості. Часто CSP можна обійти, щоб забезпечити можливість використання основної вразливості.

Як запобігти атакам XSS.

Запобігання міжсайтовому сценарію в деяких випадках є тривіальним, але може бути набагато складнішим залежно від складності програми та способів обробки керованих користувачем даних.

Загалом, ефективне запобігання вразливості XSS, ймовірно, включатиме комбінацію таких заходів:

- **Фільтрувати вхідні дані після прибуття.** У точці, де приймається введення користувачем, фільтруйте якомога строгіше, виходячи з того, що очікується або дійсне введення.
- **Кодувати дані на виході.** У той момент, коли керовані користувачем дані виводяться у відповіді HTTP, закодуйте вихід, щоб запобігти їх інтерпретації як активний вміст. Залежно від контексту виводу, для цього може знадобитися застосування комбінацій кодування HTML, URL, JavaScript та CSS.
- **Використовуйте відповідні заголовки відповідей.** Щоб запобігти XSS у відповідях HTTP, які не призначені для міщення HTML або JavaScript, ви можете використовувати заголовки Content-Typeта, X-Content-Type-Options щоб переконатись, що браузери інтерпретують відповіді так, як ви плануєте.
- **Політика безпеки вмісту.** В якості останньої лінії захисту ви можете використовувати Політику безпеки вмісту (CSP), щоб зменшити ступінь серйозності будь-яких вразливостей XSS, які все ще трапляються.

3. Міжсайтові сценарії XSS на основі DOM атаки

Вразливості XSS на основі DOM зазвичай виникають, коли JavaScript бере дані з контрольованого зловмисником джерела, такого як URL-адреса, що підтримує динамічне виконання коду, наприклад eval() або innerHTML. Це дозволяє зловмисникам виконувати зловмисний JavaScript, що зазвичай дозволяє їм викрадати облікові записи інших користувачів.

Щоб доставити атаку XSS на основі DOM, вам потрібно помістити дані у джерело, щоб вони розповсюджувались і викликали виконання довільного JavaScript.

Найпоширенішим джерелом для DOM XSS є URL-адреса, до якої зазвичай доступний window.location об'єкт. Зловмисник може побудувати посилання для пересилання жертви на вразливу сторінку з корисним навантаженням у рядку запиту та фрагменті частини URL-адреси. За певних обставин, наприклад, при націлюванні на сторінку 404 або веб-сайт, на якому запущено PHP, корисне навантаження також може бути розміщено на шляху.

Тестування HTML sinks.

Щоб перевірити наявність DOM XSS в раковині HTML, помістіть випадковий буквено-цифровий рядок у джерело (наприклад, location.search), а потім скористайтеся інструментами розробника, щоб перевірити HTML і знайти, де з’являється ваш рядок. Зауважте, що параметр браузера "Переглянути джерело" не працюватиме для тестування DOM XSS, оскільки він не враховує зміни, які були внесені в HTML за допомогою JavaScript. У інструментах розробника Chrome ви можете використовувати Control+F (або Command+F на MacOS) пошук DOM для вашого рядка.

Для кожного місця, де ваш рядок відображається в DOM, вам потрібно визначити контекст. Виходячи з цього контексту, вам слід уточнити свої дані, щоб побачити, як вони обробляються. Наприклад, якщо рядок відображається в атрибуті з подвійними лапками, спробуйте вставити подвійні лапки у рядок, щоб побачити, чи зможете ви вирватися з атрибута.

Зауважте, що браузери поводяться по-різному щодо кодування URL-адрес, Chrome, Firefox та Safari будуть кодувати URL-адреси, location.searchі в location.hashтой час як IE11 та Microsoft Edge (до Chromium) не будуть кодувати URL-адреси цих джерел. Якщо ваші дані отримують URL-кодування перед обробкою, то атака XSS навряд чи спрацює.

Тестування на виконання JavaScript.

Тестування раковин для виконання JavaScript для XSS на основі DOM трохи складніше. За допомогою цих мийок ваші дані не обов’язково з’являються де-небудь у DOM, тому ви не можете їх шукати. Натомість вам

потрібно буде використовувати налагоджувач JavaScript, щоб визначити, чи і як ваш вхід надсилається в раковину.

Для кожного потенційного джерела, наприклад location, спочатку потрібно знайти випадки в коді JavaScript сторінки, де посилається на джерело. За допомогою інструментів розробника Chrome ви можете використовувати Control+Shift+F (або Command+Alt+Fна MacOS) пошук усіх кодів JavaScript на сторінці для джерела.

Після того, як ви знайдете, де читається джерело, ви можете використовувати налагоджувач JavaScript, щоб додати точку розриву та простежити, як використовується значення джерела. Ви можете виявити, що джерело призначається іншим змінним. У цьому випадку вам доведеться знову скористатися функцією пошуку, щоб відстежити ці змінні та перевірити, чи передані вони в раковину. Коли ви знайдете раковину, якій присвоюються дані, що походять з джерела, ви можете використовувати налагоджувач для перевірки значення, наводячи курсор на змінну, щоб показати її значення перед тим, як воно буде надіслане в раковину. Потім, як і у випадку з поглиначами HTML, вам потрібно уточнити свої дані, щоб побачити, чи зможете ви здійснити успішну атаку XSS.

Використання DOM XSS з різними джерелами та поглиначами.

В принципі, веб-сайт вразливий для міжсайтових сценаріїв на основі DOM, якщо існує виконуваний шлях, за допомогою якого дані можуть поширюватися від джерела до потоку. На практиці різні джерела та раковини мають різні властивості та поведінку, що може вплинути на експлуатаційність, і визначають, які методи необхідні. Крім того, сценарії веб-сайту можуть виконувати перевірку або іншу обробку даних, які повинні бути застосовані при спробі використання вразливості. Існує безліч помилок, які мають відношення до вразливостей на основі DOM. document.write раковина працює з script елементами, так що ви можете використовувати просту корисне навантаження, наприклад, один нижче:

document.write('... <script>alert(document.domain)</script> ...');

Однак зауважте, що в деяких ситуаціях вміст, на який написано, document.write включає якийсь оточуючий контекст, який вам потрібно враховувати під час використання. Наприклад, вам може знадобитися закрити деякі існуючі елементи перед використанням вашого корисного навантаження JavaScript.

innerHTML Раковина не приймає script елементи на будь-якому сучасному браузері, і НЕ будуть svg onload події вогню. Це означає, що вам потрібно буде використовувати альтернативні елементи, такі як img або iframe. Обробники

подій, такі як onload і onerror можуть використовуватися разом із цими елементами. Наприклад:

element.innerHTML='... <img src=1 onerror=alert(document.domain)> ...'

Якщо використовується бібліотека JavaScript, така як jQuery, зверніть увагу на раковини, які можуть змінювати елементи DOM на сторінці. Наприклад, attr() функція в jQuery може змінювати атрибути елементів DOM. Якщо дані зчитуються з контрольованого користувачем джерела, такого як URL-адреса, а потім передаються attr() функції, тоді може бути можливо маніпулювати значенням, надісланим для спричинення XSS. Наприклад, тут ми маємо JavaScript, який змінює href атрибут прив'язуючого елемента, використовуючи дані з URL-адреси:

```
$(function(){
$('#backLink').attr("href",(new
URLSearchParams(window.location.search)).get('returnUrl'));
});
```

Ви можете використати це, змінивши URL-адресу, щоб location.search джерело містило шкідливу URL-адресу JavaScript. Після того, як JavaScript сторінки застосує цю шкідливу URL-адресу до зворотного посилання href, натисніть на зворотне посилання, щоб виконати її:

?returnUrl=javascript:alert(document.domain)

Якщо використовується такий фреймворк, як AngularJS , можливо буде виконати JavaScript без кутових дужок або подій. Коли сайт використовує ng-app атрибут для елемента HTML, він буде оброблений AngularJS. У цьому випадку AngularJS буде виконувати JavaScript всередині подвійних фігурних дужок, які можуть виникати безпосередньо в HTML або всередині атрибутів.

DOM XSS у поєднанні з відображеними та збереженими даними.

Деякі вразливості на основі DOM містяться на одній сторінці. Якщо скрипт зчитує деякі дані з URL-адреси та записує їх у небезпечну раковину, тоді вразливість повністю на стороні клієнта.

Однак джерела не обмежуються даними, які безпосередньо піддаються браузерам - вони також можуть походити з веб-сайту. Наприклад, веб-сайти часто відображають параметри URL у відповіді HTML із сервера. Це зазвичай асоціюється із звичайним XSS, але це також може призвести до так званих відбитих + DOM вразливостей.

У відображеній вразливості + DOM сервер обробляє дані із запиту і переносить дані у відповідь. Відображені дані можуть бути поміщені в літеральний рядок JavaScript або елемент даних у DOM, наприклад у поле форми. Потім сценарій на сторінці обробляє відображені дані небезпечним способом, в кінцевому підсумку записуючи їх у небезпечну раковину.

eval('var data = "reflected string"');

Веб-сайти можуть також зберігати дані на сервері та відображати їх в іншому місці. У вразливості, що зберігається + DOM, сервер отримує дані від одного запиту, зберігає їх, а потім включає дані в наступну відповідь. Сценарій у пізнішій відповіді містить раковину, яка потім обробляє дані небезпечним способом.

element.innerHTML = comment.author

Які стоки можуть призвести до уразливостей DOM-XSS?

Нижче наведено деякі основні помилки, які можуть призвести до уразливостей DOM-XSS:

document.write()
document.writeln()
document.domain
someDOMElement.innerHTML
someDOMElement.outerHTML
someDOMElement.insertAdjacentHTML
someDOMElement.onevent

Наступні функції jQuery - це також поглиначі, які можуть призвести до уразливостей DOM-XSS:

add()
after()
append()
animate()
insertAfter()
insertBefore()
before()
html()
prepend()
replaceAll()
replaceWith()
wrap()
wrapInner()
wrapAll()
has()
constructor()
init()
index()
jQuery.parseHTML()
$.parseHTML()

Як запобігти уразливості DOM-XSS?

На додаток до загальних заходів, слід уникати дозволу динамічно записувати дані з будь-якого ненадійного джерела в документ HTML.

4. SQL ін'єкції

Що таке ін'єкція SQL (SQLi)?

Ін’єкція SQL - це вразливість веб-безпеки, яка дозволяє зловмисникові втручатися в запити, які додаток робить до своєї бази даних. Як правило, це дозволяє зловмиснику переглядати дані, які вони зазвичай не можуть отримати. Це може включати дані, що належать іншим користувачам, або будь-які інші дані, до яких сама програма може отримати доступ. У багатьох випадках зловмисник може змінювати або видаляти ці дані, спричиняючи постійні зміни у вмісті або поведінці програми.

У деяких ситуаціях зловмисник може посилити атаку введення SQL для компрометації базового сервера або іншої внутрішньої інфраструктури або здійснити атаку відмови в обслуговуванні.

Який вплив має успішна атака введення SQL?

Успішна атака введення SQL може призвести до несанкціонованого доступу до конфіденційних даних, таких як паролі, дані кредитної картки або особиста інформація користувача. Багато гучних порушень даних за останні роки були результатом атак SQL-ін'єкцій, що призвело до пошкодження репутації та штрафу за регулювання. У деяких випадках зловмисник може отримати постійний бекдор у системах організації, що призведе до довгострокового компромісу, який може залишатися непоміченим протягом тривалого періоду.

Приклади введення SQL.

Існує широкий спектр вразливостей, атак та методів введення SQL, які виникають у різних ситуаціях. Деякі загальні приклади введення SQL включають:

- Отримання прихованих даних , де ви можете змінити запит SQL для повернення додаткових результатів.
- Підміна логіки програми , де ви можете змінити запит, щоб втрутитися в логіку програми.
- UNION атаки , де ви можете отримати дані з різних таблиць баз даних.
- Вивчення бази даних , де можна отримати інформацію про версію та структуру бази даних.
- Сліпа ін’єкція SQL , коли результати запиту, яким ви керуєте, не повертаються у відповіді програми.

Отримання прихованих даних.

Розглянемо торговий додаток, який відображає товари різних категорій. Коли користувач натискає категорію Подарунки, їх браузер запитує URL-адресу:

https://insecure-website.com/products?category=Gifts

Це змушує додаток робити запит SQL для отримання деталей відповідних продуктів з бази даних:

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

Цей запит SQL просить базу даних повернути:
- всі деталі (*)
- зі таблиці продуктів
- де категорія - Подарунки
- і звільнений 1.

Обмеження released = 1використовується для приховування продуктів, які не випускаються. Мабуть, для невипущених продуктів released = 0.

Додаток не реалізує жодних засобів захисту від атак введення SQL, тому зловмисник може побудувати атаку, наприклад:

https://insecure-website.com/products?category=Gifts'--

Це призводить до запиту SQL:

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

Ключовим тут є те, що послідовність подвійних тире --є показником коментаря в SQL, і означає, що решта запитів інтерпретується як коментар. Це ефективно видаляє залишок запиту, тому він більше не включає AND released = 1. Це означає, що відображаються всі товари, включаючи невипущені.

Далі, зловмисник може змусити програму відображати всі продукти в будь-якій категорії, включаючи категорії, про які вони не знають:

https://insecure-website.com/products?category=Gifts'+OR+1=1--

Це призводить до запиту SQL:

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

Модифікований запит поверне всі елементи, де або категорія - Подарунки, або 1 дорівнює 1. Оскільки 1=1це завжди істина, запит поверне всі елементи.

Підміна логіки програми.

Розглянемо додаток, яке дозволяє користувачам входити за допомогою імені користувача та пароля. Якщо користувач подає ім'я користувача wienerта пароль bluecheese, програма перевіряє облікові дані, виконуючи такий запит SQL:

SELECT * FROM users WHERE username = 'wiener' AND password = 'bluecheese'

Якщо запит повертає дані користувача, тоді вхід успішний. В іншому випадку він відхиляється.

Тут зловмисник може ввійти в систему як будь-який користувач без пароля, просто використовуючи послідовність коментарів SQL, --щоб видалити перевірку пароля з WHEREпункту запиту. Наприклад, надіславши ім’я користувача administrator'--та пустий пароль, ви отримаєте такий запит:

SELECT * FROM users WHERE username = 'administrator'--' AND password = ''

Цей запит повертає користувача, чиє ім’я користувача, administratorта успішно входить в систему зловмисника як цей користувач.

Отримання даних з інших таблиць баз даних.

У випадках, коли результати запитів SQL повертаються до відповідей програми, зловмисник може використати вразливість SQL-ін'єкції для отримання даних з інших таблиць бази даних. Це робиться за допомогою UNIONключового слова, яке дозволяє виконати додатковий SELECTзапит і додати результати до вихідного запиту.

Наприклад, якщо програма виконує такий запит, що містить введені користувачем дані "Подарунки":

SELECT name, description FROM products WHERE category = 'Gifts'

тоді зловмисник може подати дані:

' UNION SELECT username, password FROM users--

Це призведе до того, що програма поверне всі імена користувачів та паролі, а також імена та описи продуктів.

5. SQL-ін'єкційні UNION атаки

Коли програма вразлива до введення SQL і результати запиту повертаються до відповідей програми, UNION ключове слово можна використовувати для отримання даних з інших таблиць бази даних. Це призводить до атаки об'єднання SQL UNION.

UNION ключове слово дозволяє виконати одну або кілька додаткових SELECT запитів і додати результати до вихідного запиту. Наприклад:

```
SELECT a, b FROM table1 UNION SELECT c, d FROM table2
```

Цей запит SQL поверне єдиний набір результатів із двома стовпцями, що містить значення зі стовпців a і b в table1і стовпців c і d в table2.

Щоб UNION запит працював, мають бути виконані дві ключові вимоги:

- Окремі запити повинні повертати однакову кількість стовпців.
- Типи даних у кожному стовпці повинні бути сумісними між окремими запитами.

Для здійснення атаки SQL-ін'єкції UNION потрібно переконатися, що ваша атака відповідає цим двом вимогам. Це, як правило, передбачає з’ясування:

- Скільки стовпців повертається з вихідного запиту?
- Які стовпці, повернуті з вихідного запиту, мають відповідний тип даних для зберігання результатів введеного запиту?

Визначення кількості стовпців, необхідних для атаки SQL-ін'єкції UNION.

Під час виконання атаки SQL-ін'єкції UNION існує два ефективні методи, щоб визначити, скільки стовпців повертається з вихідного запиту.

Перший метод передбачає введення серії ORDER BY пропозицій та збільшення вказаного індексу стовпця до появи помилки. Наприклад, якщо припустити, що точка введення - це рядок із WHERE лапками в реченні вихідного запиту, ви подаєте:

```
' ORDER BY 1--
' ORDER BY 2--
' ORDER BY 3--
etc.
```

Ця серія корисних навантажень модифікує вихідний запит для упорядкування результатів за різними стовпцями в наборі результатів. Стовпець у ORDER BY реченні може бути заданий його індексом, тому вам не потрібно знати імена будь-яких стовпців. Коли вказаний індекс стовпця

перевищує кількість фактичних стовпців у наборі результатів, база даних повертає помилку, таку як:

ORDER BY позиція номер 3 виходить за допустимі від кількості елементів в списку вибору.

Додаток може фактично повернути помилку бази даних у своїй відповіді HTTP, або може повернути загальну помилку, або просто не повернути результатів. Якщо ви можете виявити деяку різницю у відповіді програми, ви можете зробити висновок, скільки стовпців повертається із запиту.

Другий метод передбачає подання серії UNION SELECT корисних навантажень із зазначенням різної кількості нульових значень:

' UNION SELECT NULL--

' UNION SELECT NULL,NULL--

' UNION SELECT NULL,NULL,NULL--

etc.

Якщо кількість нулів не відповідає кількості стовпців, база даних повертає помилку, таку як:

Усі запити, об’єднані за допомогою оператора UNION, INTERSECT або EXCEPT, повинні мати однакову кількість виразів у своїх цільових списках.

Знову ж таки, програма може фактично повернути це повідомлення про помилку, а може просто повернути загальну помилку або відсутність результатів. Коли кількість нулів відповідає кількості стовпців, база даних повертає додатковий рядок у наборі результатів, що містить нульові значення в кожному стовпці. Вплив на отриману відповідь HTTP залежить від коду програми. Якщо вам пощастить, ви побачите додатковий вміст у відповіді, наприклад додатковий рядок у таблиці HTML. В іншому випадку нульові значення можуть спричинити іншу помилку, таку як NullPointerException. Гірший випадок, що відповідь може не відрізнятися від відповіді, спричиненої неправильною кількістю нулів, що робить цей метод визначення підрахунку стовпців неефективним.

Пошук стовпців з корисним типом даних під час атаки SQL UNION UNION.

Причиною виконання атаки вливання SQL UNION є можливість отримання результатів із введеного запиту. Як правило, цікаві дані, які ви хочете отримати, будуть у формі рядка, тому вам потрібно знайти один або кілька стовпців у вихідних результатах запиту, тип даних яких є або сумісний із даними рядка.

Вже визначивши кількість необхідних стовпців, ви можете перевірити кожен стовпець, щоб перевірити, чи може він містити рядкові дані, подавши ряд UNION SELECT корисних навантажень, які по черзі розміщують значення

рядка в кожному стовпці. Наприклад, якщо запит повертає чотири стовпці, ви подаєте:

```
' UNION SELECT 'a',NULL,NULL,NULL--
' UNION SELECT NULL,'a',NULL,NULL--
' UNION SELECT NULL,NULL,'a',NULL--
' UNION SELECT NULL,NULL,NULL,'a'--
```

Якщо тип даних стовпця не сумісний із рядковими даними, введений запит спричинить помилку бази даних, таку як:

Помилка перетворення під час перетворення значення varchar 'a' у тип даних int. Якщо помилка не виникає, а відповідь програми містить деякий додатковий вміст, включаючи введене значення рядка, тоді відповідний стовпець підходить для отримання даних рядка.

Використання SQL-ін'єкції UNION для отримання цікавих даних.

Коли ви визначили кількість стовпців, повернутих вихідним запитом, і знайшли, в яких стовпцях можуть бути дані рядків, ви можете отримати цікаві дані.

Припустимо, що:

- Оригінальний запит повертає два стовпці, обидва можуть містити рядкові дані.
- Точка введення - це рядок, указаний у цитаті WHERE.
- База даних містить таблицю, що викликається users зі стовпцями username та password.

У цій ситуації ви можете отримати вміст users таблиці, подавши дані:

```
' UNION SELECT username, password FROM users—
```

Звичайно, найважливіша інформація, необхідна для здійснення цієї атаки, полягає в тому, що існує таблиця, що викликається users із двома стовпцями, що викликаються username та password. Без цієї інформації ви залишилися б намагатися вгадати назви таблиць і стовпців. Насправді всі сучасні бази даних надають способи вивчення структури бази даних, щоб визначити, які таблиці та стовпці вона містить.

Отримання кількох значень в одному стовпці

У попередньому прикладі припустимо, що замість цього запит повертає лише один стовпець.

Ви можете легко отримати декілька значень разом у цьому одному стовпці, об'єднавши значення разом, в ідеалі включаючи відповідний роздільник, щоб

дозволити вам розрізнити об’єднані значення. Наприклад, на Oracle ви можете подати дані:

' UNION SELECT username || '~' || password FROM users--

Тут використовується подвійна послідовність, ||яка є оператором об’єднання рядків в Oracle. Введений запит об'єднує значення полів username і password полів, розділених ~символом.

Результати запиту дозволять вам прочитати всі імена користувачів та паролі, наприклад:

...
administrator~s3cure
wiener~peter
carlos~montoya
...

Зверніть увагу, що різні бази даних використовують різний синтаксис для виконання конкатенації рядків.

6. XML (XXE) ін’єкція

Що таке введення зовнішньої сутності XML?

Введення зовнішньої сутності XML (також відоме як XXE) - це вразливість веб-безпеки, яка дозволяє зловмисникові перешкоджати обробці додатком даних XML. Це часто дозволяє зловмиснику переглядати файли у файловій системі сервера додатків та взаємодіяти з будь-якими внутрішніми або зовнішніми системами, до яких сама програма може отримати доступ.

У деяких ситуаціях зловмисник може посилити атаку XXE, щоб скомпрометувати базовий сервер або іншу внутрішню інфраструктуру, використовуючи вразливість XXE для виконання атак підробки на стороні сервера (SSRF).

Як виникають вразливості XXE?

Деякі програми використовують формат XML для передачі даних між браузером та сервером. Додатки, які роблять це практично завжди, використовують стандартну бібліотеку або API платформи для обробки даних XML на сервері. Вразливості XXE виникають через те, що специфікація XML містить різні потенційно небезпечні функції, а стандартні аналізатори підтримують ці функції, навіть якщо вони зазвичай не використовуються додатком.

Зовнішні сутності XML - це тип власних сутностей XML, визначені значення яких завантажуються поза DTD, в якому вони оголошені. Зовнішні сутності особливо цікаві з точки зору безпеки, оскільки вони дозволяють визначати сутність на основі вмісту шляху до файлу або URL-адреси.

Які типи атак XXE?

Існують різні типи атак XXE:

- Використання XXE для отримання файлів, де визначено зовнішню сутність, що містить вміст файлу, і повернуто у відповідь програми.
- Використання XXE для виконання атак SSRF, де зовнішня сутність визначається на основі URL-адреси до внутрішньої системи.
- Використовуючи сліпий XXE, фільтрують дані поза зоною, де конфіденційні дані передаються із сервера додатків в систему, якою керує зловмисник.
- Використання сліпого XXE для отримання даних за допомогою повідомлень про помилки, де зловмисник може викликати повідомлення про помилку синтаксичного аналізу, що містить конфіденційні дані.

Використання XXE для отримання файлів.

Щоб виконати атаку ін'єкції XXE, яка отримує довільний файл із файлової системи сервера, потрібно змінити представлений XML двома способами:

- Введіть (або відредагуйте) DOCTYPE елемент, який визначає зовнішню сутність, що містить шлях до файлу.
- Відредагуйте значення даних у XML, яке повертається у відповідь програми, щоб використовувати визначену зовнішню сутність.

Наприклад, припустимо, додаток для покупок перевіряє рівень запасів товару, подаючи на сервер такий XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<stockCheck><productId>381</productId></stockCheck>
```

Додаток не виконує особливих засобів захисту від атак XXE, тому ви можете скористатися вразливістю XXE для отримання /etc/passwdфайлу, надіславши наступне корисне навантаження XXE:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [ <!ENTITY xxe SYSTEM "file:///etc/passwd"> ]>
<stockCheck><productId>&xxe;</productId></stockCheck>
```

Це корисне навантаження XXE визначає зовнішню сутність &xxe;, значення якої є вмістом /etc/passwdфайлу, і використовує сутність у межах productIdзначення. Це призводить до того, що відповідь програми включає вміст файлу:

```
…
Invalid product ID: root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
...
```

Примітка

З реальними вразливостями XXE у поданому XML часто є велика кількість значень даних, будь-яке з яких може бути використано у відповіді програми. Щоб систематично тестувати на вразливості XXE, вам, як правило, доведеться тестувати кожен вузол даних у XML окремо, використовуючи визначену сутність та перевіряючи, чи не відображається вона у відповіді.

Використання XXE для виконання атак SSRF.

Окрім отримання конфіденційних даних, іншим головним наслідком атак XXE є те, що вони можуть бути використані для підробки сторонніх запитів (SSRF). Це потенційно серйозна вразливість, при якій серверну програму можна спонукати робити HTTP-запити на будь-яку URL-адресу, до якої сервер може отримати доступ.

Щоб використати вразливість XXE для здійснення атаки SSRF, потрібно визначити зовнішню сутність XML, використовуючи URL-адресу, на яку потрібно націлити, і використовувати визначену сутність у межах значення даних. Якщо ви можете використовувати визначену сутність у межах значення даних, яке повертається у відповіді програми, тоді ви зможете переглянути відповідь за URL-адресою у відповіді програми, і таким чином отримати двосторонню взаємодію з внутрішньою системою. Якщо ні, тоді ви зможете виконувати лише сліпі SSRF- атаки (що все одно може мати критичні наслідки).

У наступному прикладі XXE зовнішня сутність призведе до того, що сервер робитиме внутрішній HTTP-запит до внутрішньої системи в рамках інфраструктури організації:

<!DOCTYPE foo [<!ENTITY xxe SYSTEM "http://internal.vulnerable-website.com/">]>

Сліпі вразливості XXE.

Багато випадків уразливості XXE є сліпими. Це означає, що програма не повертає значення будь-яких визначених зовнішніх сутностей у своїх відповідях, і тому пряме отримання файлів на стороні сервера неможливе.

Сліпі вразливості XXE все ще можна виявити та використати, але потрібні більш досконалі методи. Іноді можна використовувати позасмугові методи, щоб знайти вразливі місця та використати їх для вилучення даних. І іноді ви можете викликати помилки аналізу XML, які призводять до розкриття конфіденційних даних у повідомленнях про помилки.

Пошук прихованої поверхні атаки для ін’єкції XXE.

Поверхня атаки для вразливостей введення XXE очевидна в багатьох випадках, оскільки звичайний HTTP-трафік програми включає запити, що містять дані у форматі XML. В інших випадках поверхня атаки менш помітна. Однак, якщо ви подивитесь у потрібних місцях, ви знайдете поверхню атаки XXE у запитах, які не містять XML.

Деякі програми отримують дані, подані клієнтом, вставляють їх на стороні сервера в документ XML, а потім аналізують документ. Приклад цього трапляється, коли подані клієнтом дані розміщуються у внутрішньому SOAP-запиті, який потім обробляється серверною серверною SOAP.

У цій ситуації ви не можете здійснити класичну атаку XXE, оскільки ви не контролюєте весь документ XML і тому не можете визначити або змінити DOCTYPE елемент. Однак ви можете XInclude замість цього використовувати. XInclude є частиною специфікації XML, яка дозволяє будувати XML-документ із піддокументів. Ви можете розмістити XInclude атаку в межах будь-якого

значення даних у документі XML, тому атака може бути здійснена в ситуаціях, коли ви керуєте лише одним елементом даних, який розміщується в XML-документі на стороні сервера.

Щоб здійснити XInclude атаку, вам потрібно зробити посилання на XInclude простір імен і вказати шлях до файлу, який ви хочете включити. Наприклад:

```
<foo xmlns:xi="http://www.w3.org/2001/XInclude">
<xi:include parse="text" href="file:///etc/passwd"/></foo>
```

XXE атаки через завантаження файлів.

Деякі програми дозволяють користувачам завантажувати файли, які потім обробляються на стороні сервера. Деякі поширені формати файлів використовують XML або містять підкомпоненти XML. Прикладами форматів на основі XML є формати офісних документів, такі як DOCX, та формати зображень, такі як SVG.

Наприклад, програма може дозволити користувачам завантажувати зображення та обробляти або перевіряти їх на сервері після їх завантаження. Навіть якщо програма розраховує отримати такий формат, як PNG або JPEG, використовувана бібліотека обробки зображень може підтримувати зображення SVG. Оскільки формат SVG використовує XML, зловмисник може надіслати шкідливий образ SVG і таким чином досягти прихованої поверхні атаки для вразливостей XXE.

XXE атаки через змінений тип вмісту.

Більшість запитів POST використовують тип вмісту за замовчуванням, який генерується формами HTML, такими як application/x-www-form-urlencoded. Деякі веб-сайти розраховують отримувати запити у такому форматі, але допускатимуть інші типи вмісту, зокрема XML.

Наприклад, якщо звичайний запит містить таке:

```
POST /action HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 7
foo=bar
```

Тоді ви можете надіслати такий запит із тим самим результатом:

```
POST /action HTTP/1.0
Content-Type: text/xml
```

```
Content-Length: 52

<?xml version="1.0" encoding="UTF-8"?><foo>bar</foo>
```

Якщо додаток допускає запити, що містять XML у тілі повідомлення, і аналізує вміст тіла як XML, тоді ви можете досягти прихованої поверхні атаки XXE, просто переформатувавши запити для використання формату XML.

Як знайти та протестувати на вразливості XXE

Переважна більшість вразливостей XXE можна швидко та надійно знайти за допомогою веб-сканера вразливостей Burp Suite.

Тестування на вразливості XXE вручну, як правило, включає:

- Тестування на отримання файлу шляхом визначення зовнішньої сутності на основі добре відомого файлу операційної системи та використання цієї сутності в даних, які повертаються у відповіді програми.
- Тестування на сліпі вразливості XXE шляхом визначення зовнішньої сутності на основі URL-адреси до системи, яку ви контролюєте, та моніторингу взаємодії з цією системою. Для цього ідеально підходить клієнт Burp Collaborator.
- Тестування на вразливе включення наданих користувачем даних, що не належать до XML, до XML-документа на стороні сервера за допомогою атаки XInclude для спроби отримати відомий файл операційної системи.

Як запобігти вразливості XXE.

Практично всі вразливості XXE виникають через те, що бібліотека аналізу XML додатка підтримує потенційно небезпечні функції XML, які програма не потребує або має намір використовувати. Найпростіший та найефективніший спосіб запобігти атакам XXE - вимкнути ці функції.

Як правило, достатньо вимкнути роздільну здатність зовнішніх сутностей та вимкнути підтримку XInclude. Зазвичай це можна зробити за допомогою параметрів конфігурації або програмно перевизначивши поведінку за замовчуванням. Для отримання детальної інформації про те, як вимкнути непотрібні можливості, зверніться до документації до вашої бібліотеки або API для аналізу XML.

7. Отруєння веб - кешом.

Що таке отруєння веб-кешем?

Отруєння веб-кешем - це вдосконалена техніка, за допомогою якої зловмисник використовує поведінку веб-сервера та кешу, так що шкідлива відповідь HTTP подається іншим користувачам.

По суті, отруєння веб-кешем включає дві фази. По-перше, зловмисник повинен вирішити, як викликати відповідь із внутрішнього сервера, який ненавмисно містить якусь небезпечну корисну навантаження. Після успіху їм потрібно переконатись, що їх відповідь кешована та надана надана передбачуваним жертвам.

Отруєний веб-кеш може потенційно бути руйнівним засобом розповсюдження численних різних атак, використання таких вразливих місць, як XSS, ін'єкція JavaScript, відкрите перенаправлення тощо.

Дослідження отруєння веб-кешу.

Вперше ця методика була популяризована в нашій дослідницькій роботі 2018 року „Практичне отруєння веб-кешу”, а далі розвинута у 2020 році разом із другою дослідницькою роботою „Заплутування веб-кешу: нові шляхи до отруєння”. Якщо вас цікавить детальний опис того, як ми виявили та використали ці вразливості в дикій природі, повний опис матеріалів доступний на нашій сторінці дослідження.

Дослідження

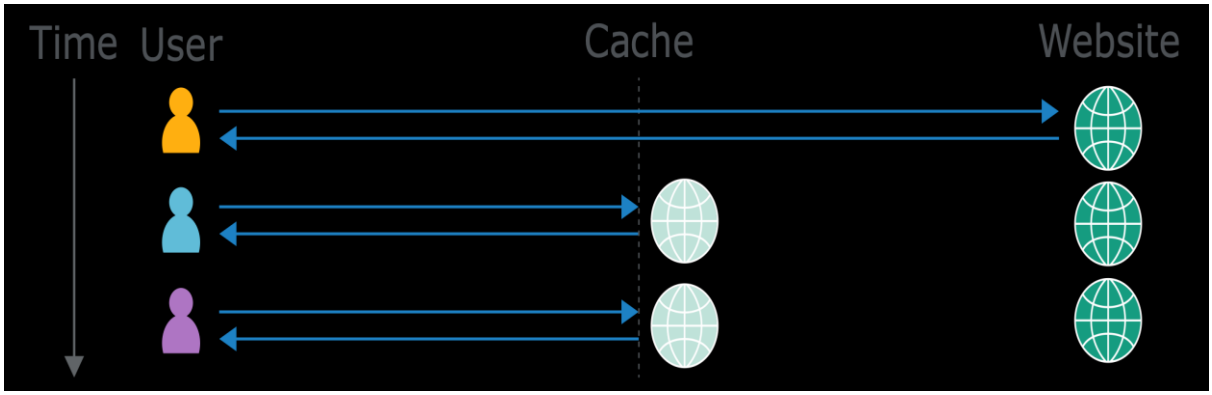
- Практичне отруєння веб-кешем
- Заплутаність веб-кешу: нові шляхи до отруєння

Як працює веб-кеш?

Щоб зрозуміти, як виникають вразливості до отруєння веб-кешем, важливо мати базове розуміння того, як працює веб-кеш.

Якщо серверу доводиться надсилати нову відповідь на кожен окремий HTTP-запит окремо, це, ймовірно, перевантажить сервер, що призведе до проблем із затримками та поганою взаємодією з користувачами, особливо під час напружених періодів. Кешування є насамперед засобом зменшення таких проблем.

Кеш знаходиться між сервером та користувачем, де він зберігає (кешує) відповіді на конкретні запити, як правило, протягом фіксованого періоду часу. Якщо інший користувач потім надсилає еквівалентний запит, кеш просто подає копію кешованої відповіді безпосередньо користувачеві, без будь-якої взаємодії з внутрішньої сторони. Це значно полегшує навантаження на сервер, зменшуючи кількість повторюваних запитів, якими він повинен обробляти.



Клавіші кешу.

Коли кеш отримує HTTP-запит, він спочатку повинен визначити, чи є кешована відповідь, яку він може безпосередньо обслуговувати, чи йому потрібно переслати запит на обробку на внутрішньому сервері. Кеші ідентифікують еквівалентні запити, порівнюючи заздалегідь визначену підмножину компонентів запиту, відому в сукупності як "ключ кешу". Як правило, це міститиме рядок запиту та Host заголовок. Компоненти запиту, які не включені до ключа кешу, називаються "нерозблокованими".

Якщо ключ кешу вхідного запиту відповідає ключу попереднього запиту, тоді кеш вважає їх еквівалентними. Як результат, він подасть копію кешованої відповіді, створеної для вихідного запиту. Це стосується всіх наступних запитів із відповідним ключем кешу, поки кешована відповідь не закінчиться.

Найважливіше, що інші компоненти запиту ігноруються кешем. Вплив такої поведінки ми детальніше вивчимо пізніше.

Який вплив атаки на отруєння веб-кешу?

Вплив отруєння веб-кешем сильно залежить від двох ключових факторів:

- **Що саме зловмисник може успішно кешувати**. Оскільки отруєний кеш є більше засобом розповсюдження, ніж автономною атакою, вплив отруєння веб-кешем нерозривно пов’язаний із тим, наскільки шкідливим є введене корисне навантаження. Як і для більшості видів атак, отруєння веб-кешем також можна використовувати в поєднанні з іншими атаками, щоб ще більше посилити потенційний вплив.
- **Обсяг трафіку на ураженій сторінці**. Отруєна відповідь буде надана лише тим користувачам, які відвідують уражену сторінку, поки кеш отруєний. Як результат, вплив може коливатися від неіснуючого до масштабного залежно від того, популярна сторінка чи ні. Наприклад, якщо зловмисникові вдалося отруїти кешовану відповідь на домашній сторінці великого веб-сайту, атака може вплинути на тисячі користувачів без подальшої взаємодії зловмисника.

Зверніть увагу, що тривалість введення кешу не обов'язково впливає на вплив отруєння веб-кешем. Зазвичай атака може бути написана таким чином, що вона повторно отруює кеш на невизначений час.

Побудова атаки на отруєння веб-кешу.

Взагалі кажучи, побудова базової атаки на отруєння веб-кешем включає такі кроки:

1. Визначте та оцініть необмежені вхідні дані
2. Викликати шкідливу реакцію з внутрішнього сервера
3. Отримати кешовану відповідь

Визначте та оцініть необмежені вхідні дані.

Будь-яка атака на отруєння веб-кешу покладається на маніпуляції з необмеженими вхідними даними, такими як заголовки. Веб-кеші ігнорують необмежені вводи, коли вирішують, чи подавати кешовану відповідь користувачеві. Така поведінка означає, що ви можете використовувати їх для ін'єкції вашого корисного навантаження та отримання "отруєної" відповіді, яка, якщо буде кешована, буде подана всім користувачам, чиї запити мають відповідний ключ кешу. Отже, першим кроком під час створення атаки на отруєння веб-кешу є виявлення необмежених входів, які підтримуються сервером.

Ви можете ідентифікувати необмежені входи вручну, додаючи випадкові входи до запитів і спостерігаючи, чи впливають вони на відповідь чи ні. Це може бути очевидним, наприклад, безпосередньо відображати вхідні дані у відповіді або викликати зовсім іншу відповідь. Однак іноді ефекти виявляються більш тонкими і вимагають трохи детективної роботи, щоб з’ясувати це. Ви можете використовувати такі інструменти, як Burp Comparer, щоб порівняти відповідь із введеним введенням та без нього, але це все одно передбачає значну кількість ручних зусиль.

Param Miner. На щастя, ви можете автоматизувати процес ідентифікації необмежених входів, додавши розширення Param Miner до Burp із магазину BApp. Щоб скористатися Param Miner, ви просто клацаєте правою кнопкою миші на запит, який ви хочете дослідити, і натискаєте "Відгадати заголовки". Потім Param Miner запускається у фоновому режимі, надсилаючи запити, що містять різні вхідні дані зі свого великого, вбудованого списку заголовків. Якщо запит, що містить один із введених входів, впливає на відповідь, Param Miner реєструє це в Burp або на панелі "Проблеми", якщо ви використовуєте Burp Suite Professional, або на вкладці "Вивід" розширення (" Extender ">" Extensions ">" Param Miner ">" Output "), якщо ви використовуєте Burp Suite Community Edition.

Наприклад, на наступному скріншоті Param Miner знайшов неодмінний заголовок X-Forwarded-Hostна домашній сторінці веб-сайту:

Увага: Під час тестування необмежених входів на веб-сайті, що працює, існує ризик ненавмисного змушення кеш-пам'яті подати ваші згенеровані відповіді реальним користувачам. Тому важливо переконатися, що всі ваші запити мають унікальний ключ кеш-пам’яті, щоб вони були подані лише вам. Для цього ви можете вручну додавати кеш-пам’ять (наприклад, унікальний параметр) до рядка запиту кожного разу, коли ви робите запит. Крім того, якщо ви використовуєте Param Miner, є варіанти автоматичного додавання кеш-пам’яті до кожного запиту.

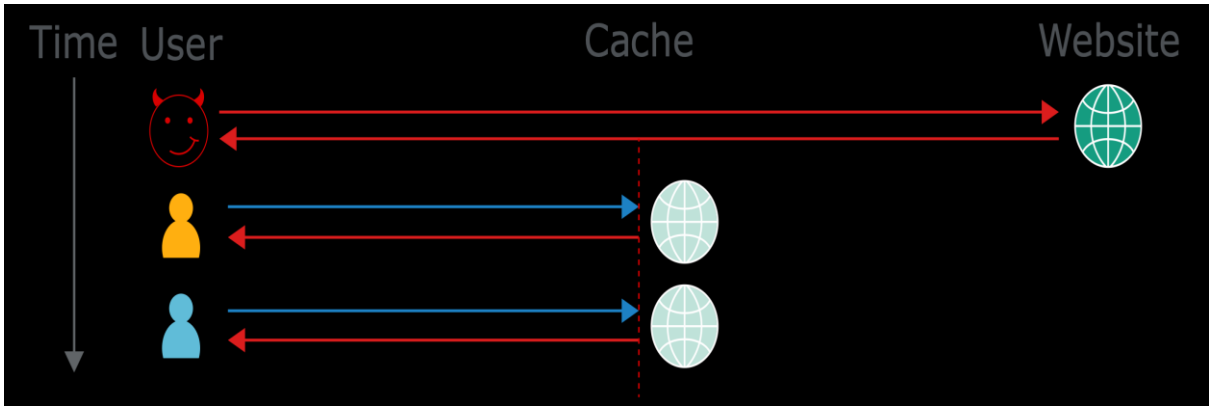
Викликати шкідливу реакцію з внутрішнього сервера.

Після того, як ви визначили необмежений вхід, наступним кроком є оцінка того, як саме веб-сайт його обробляє. Розуміння цього має важливе значення для успішного отримання шкідливої реакції. Якщо вхідні дані відображаються у відповіді сервера без належної санітарної обробки або використовуються для динамічного генерування інших даних, тоді це потенційна точка входу для отруєння веб-кешу.

Отримати кешовану відповідь.

Маніпулювання входами для отримання шкідливої реакції - це половина успіху, але це не дає великого результату, якщо ви не примусите кешувати відповідь, що іноді може бути складно.

Чи кешована відповідь, може залежати від усіх видів факторів, таких як розширення файлу, тип вмісту, маршрут, код стану та заголовки відповідей. Ймовірно, вам доведеться приділити трохи часу, щоб просто погратися із запитами на різних сторінках та вивчити, як поводиться кеш. Після того, як ви вирішите, як отримати кешовану відповідь, яка містить ваші зловмисні дані, ви готові надати експлойт потенційним жертвам.



Використання вразливостей, що отруюють веб-кеш.

Цей основний процес може бути використаний для виявлення та використання різноманітних вразливих місць, що отруюють веб-кеш.

У деяких випадках вразливості до отруєння веб-кешу виникають через загальні недоліки в розробці кеш-пам’яті. В інших випадках спосіб, яким кеш реалізується певним веб-сайтом, може спричинити несподівані химерності, якими можна скористатися.

У наступних розділах ми наведемо деякі найпоширеніші приклади обох цих сценаріїв. Ми також створили низку інтерактивних лабораторій, щоб ви могли бачити деякі з цих вразливостей у дії та практикувати їх використання.

Як запобігти отруєнню вразливостей веб-кешу.

Однозначним способом запобігання отруєння веб-кешу явно було б повністю відключити кешування. Хоча для багатьох веб-сайтів це може бути нереалістичним варіантом, в інших випадках це може бути здійсненно. Наприклад, якщо ви використовуєте кешування лише тому, що воно було ввімкнено за замовчуванням, коли ви прийняли CDN, можливо, варто оцінити, чи справді параметри кешування за замовчуванням відображають ваші потреби.

Навіть якщо вам потрібно використовувати кешування, обмеження його суто статичними відповідями також є ефективним, за умови, що ви достатньо обережно ставитесь до того, що ви класифікуєте як "статичне". Наприклад, переконайтеся, що зловмисник не може обдурити внутрішній сервер, щоб отримати свою шкідливу версію статичного ресурсу замість справжнього.

Це також пов’язано з ширшим пунктом про веб-безпеку. Зараз більшість веб-сайтів включають різноманітні сторонні технології як у свої процеси розробки, так і в повсякденні операції. Якою б надійною не була ваша власна поза внутрішньої безпеки, як тільки ви включите сторонні технології у своє середовище, ви покладаєтесь на те, що її розробники також відповідають за безпеку, як і ви. Виходячи з того, що ви захищені лише як найслабше місце, життєво важливо переконатись, що ви повністю розумієте наслідки будь-якої сторонньої технології, перш ніж її інтегрувати.

Зокрема, в контексті отруєння веб-кешем, це не тільки означає вирішити, чи залишати кешування увімкненим за замовчуванням, але також переглядати, які заголовки підтримує ваш CDN, наприклад. Деякі з вищезазначених вразливостей щодо отруєння веб-кешу розкриваються, оскільки зловмисник може маніпулювати низкою незрозумілих заголовків запитів, багато з яких абсолютно непотрібні для функціональності веб-сайту. Знову ж таки, ви можете піддаватися таким атакам, навіть не підозрюючи, лише тому, що ви впровадили якусь технологію, яка підтримує ці незаключені входи за замовчуванням. Якщо заголовок не потрібен для роботи сайту, його слід відключити.

Також слід вжити таких запобіжних заходів, застосовуючи кешування:

- Якщо ви плануєте виключити щось із ключа кешу з міркувань продуктивності, замість цього перепишіть запит.
- Не приймайте жирних GET прохань. Майте на увазі, що деякі сторонні технології можуть дозволяти це за замовчуванням.
- Виправляйте вразливості на стороні клієнта, навіть якщо вони здаються невикористаними. Деякі з цих вразливостей насправді можуть бути використані через непередбачувані химерності у поведінці кешу. Це може бути питанням часу, коли хтось знайде химерність, незалежно від того, заснована вона на кеш-пам'яті чи іншим чином, що робить цю вразливість придатною для використання.

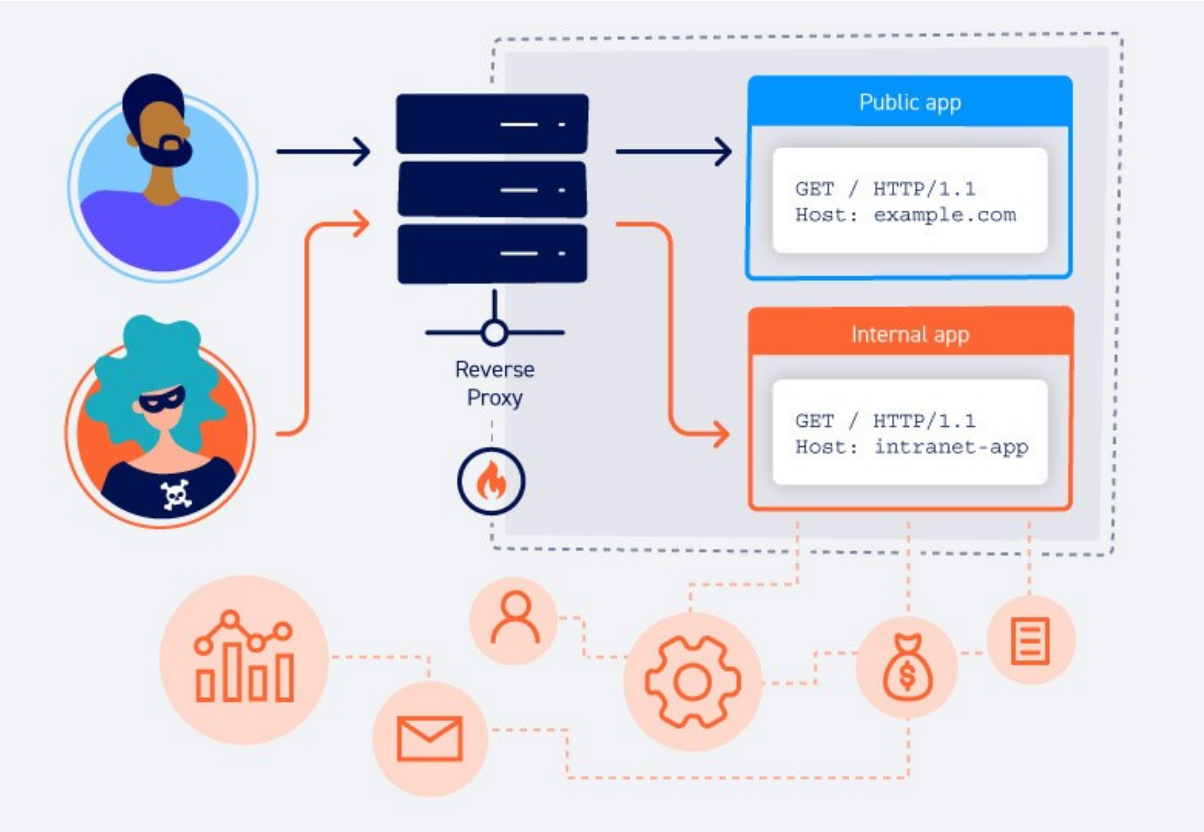
8. Атаки заголовку хоста HTTP

Що таке заголовок хосту HTTP?

Заголовок хосту HTTP є обов'язковим заголовком запиту станом на HTTP / 1.1. Він визначає доменне ім'я, до якого клієнт хоче отримати доступ. Наприклад, коли користувач відвідує https://www.google.com, його браузер складає запит, що містить заголовок Host, наступним чином:

GET /web-security HTTP/1.1
Host: www.google.com

У деяких випадках, наприклад, коли запит було переадресовано посередницькою системою, значення Host може бути змінено до того, як воно досягне передбачуваного внутрішнього компонента. Ми докладніше обговоримо цей сценарій нижче.



Яка мета заголовка хосту HTTP?

Призначення заголовка хосту HTTP - допомогти визначити, з яким внутрішнім компонентом хоче спілкуватися клієнт. Якщо запити не містять заголовків хосту або якщо заголовок хосту був дещо неправильним, це може призвести до проблем при маршрутизації вхідних запитів до передбачуваної програми.

Історично цієї двозначності не існувало, оскільки кожна IP-адреса містила вміст лише для одного домену. В даний час, здебільшого завдяки постійно зростаючій тенденції до хмарних рішень та передачі більшої частини відповідної архітектури на аутсорсинг, багато веб-сайтів та додатків доступні за однією IP-адресою. Цей підхід також збільшив свою популярність частково внаслідок вичерпання адрес IPv4.

Коли доступ до кількох програм здійснюється через одну і ту ж IP-адресу, це найчастіше є результатом одного з наступних сценаріїв.

Віртуальний хостинг.

Одним із можливих сценаріїв є те, коли на одному веб-сервері розміщено кілька веб-сайтів або додатків. Це може бути декілька веб-сайтів з одним власником, але також можливо, що веб-сайти з різними власниками розміщуються на одній спільній платформі. Це рідше, ніж було раніше, але все одно трапляється з деякими хмарними рішеннями SaaS.

У будь-якому випадку, хоча кожен із цих різних веб-сайтів матиме інше доменне ім’я, всі вони мають спільну IP-адресу із сервером. Веб-сайти, розміщені таким чином на одному сервері, відомі як "віртуальні хости".

Звичайного користувача, який отримує доступ до веб-сайту, віртуальний хост часто не відрізняє від веб-сайту, розміщеного на його власному виділеному сервері.

Маршрутизація трафіку через посередника.

Інший поширений сценарій - це те, коли веб-сайти розміщуються на різних серверних серверах, але весь трафік між клієнтом і серверами направляється через систему посередників. Це може бути простий балансир навантаження або якийсь зворотний проксі-сервер. Це налаштування особливо поширене у випадках, коли клієнти переходять на веб-сайт через мережу доставки вмісту (CDN).

У цьому випадку, незважаючи на те, що веб-сайти розміщені на окремих серверних серверах, усі їх доменні імена перетворюються на одну IP-адресу посередницького компонента. Це являє собою деякі ті самі проблеми, що і віртуальний хостинг, тому що зворотний проксі-сервер або балансир навантаження повинен знати відповідний сервер, до якого він повинен направляти кожен запит.

Як заголовок хосту HTTP вирішує цю проблему?

В обох цих сценаріях покладається на заголовок хоста, щоб вказати передбачуваного одержувача. Типовою аналогією є процес надсилання листа тому, хто живе в багатоквартирному будинку. Уся будівля має однакову вулицю, але за цією вулицею є багато різних квартир, кожна з яких повинна

якось отримати правильну пошту. Одним із варіантів вирішення цієї проблеми є просто включення до адреси номера квартири або імені одержувача. У випадку повідомлень HTTP, заголовок Host має аналогічну мету.

Коли браузер надсилає запит, цільова URL-адреса перетворюється на IP-адресу певного сервера. Коли цей сервер отримує запит, він посилається на заголовок хосту, щоб визначити передбачуваний серверний пакет і відповідно пересилає запит.

Що таке атака заголовка хосту HTTP?

Атаки заголовків хостів HTTP експлуатують вразливі веб-сайти, які небезпечно обробляють значення заголовка хосту. Якщо сервер неявно довіряє заголовку хосту і не може перевірити його або захистити його належним чином, зловмисник може використовувати цей вхід для введення шкідливих корисних навантажень, які маніпулюють поведінкою на стороні сервера. Атаки, що передбачають введення корисного навантаження безпосередньо в заголовок хосту, часто називають атаками "введення заголовка хосту".

Готові веб-програми зазвичай не знають, на якому домені вони розгорнуті, якщо це не вказано вручну у файлі конфігурації під час налаштування. Наприклад, коли їм потрібно знати поточний домен, щоб сформувати абсолютну URL-адресу, включену в електронне повідомлення, вони можуть вдатися до отримання домену із заголовка хосту:

```
<a href="https://_SERVER['HOST']/support">Contact support</a>
```

Значення заголовка може також використовуватися в різних взаємодіях між різними системами інфраструктури веб-сайту.

Оскільки заголовок Host фактично контролюється користувачем, така практика може призвести до низки проблем. Якщо вхідні дані не є належним чином захищеними або перевіреними, заголовок Host є потенційним вектором для використання ряду інших уразливостей, зокрема:

- Отруєння веб-кешем
- Недоліки бізнес- логіки в конкретних функціональних можливостях
- SSRF на основі маршрутизації
- Класичні вразливості на стороні сервера, такі як введення SQL

Як виникають уразливості заголовка хосту HTTP?

Вразливості заголовків хостів HTTP зазвичай виникають через хибне припущення, що заголовок не контролюється користувачем. Це створює неявну довіру до заголовка Host і призводить до неадекватної перевірки або виходу його значення, хоча зловмисник може легко змінити це за допомогою таких інструментів, як Burp Proxy.

Навіть якщо сам заголовок Host обробляється більш безпечно, залежно від конфігурації серверів, що працюють з вхідними запитами, Host потенційно може бути замінений шляхом введення інших заголовків. Іноді власники веб-сайтів не підозрюють, що ці заголовки підтримуються за замовчуванням, і, як наслідок, до них може не застосовуватися однаковий рівень перевірки.

Насправді багато з цих уразливостей виникають не через небезпечне кодування, а через небезпечну конфігурацію одного або декількох компонентів у відповідній інфраструктурі. Ці проблеми з конфігурацією можуть виникати через те, що веб-сайти інтегрують сторонні технології в свою архітектуру, не обов'язково розуміючи параметри конфігурації та їх наслідки для безпеки.

Як запобігти атакам заголовків хосту HTTP

Щоб запобігти атакам заголовків хостів HTTP, найпростішим підходом є уникнення використання заголовка хосту в коді на стороні сервера. Перевірте, чи дійсно кожна URL-адреса має бути абсолютною. Ви часто виявляєте, що замість цього ви можете просто використовувати відносну URL-адресу. Ця проста зміна може допомогти вам запобігти отруєнню вразливостей веб-кешу.

Інші способи запобігання атакам заголовків хосту HTTP включають:

Захист абсолютних URL-адрес.

Коли вам потрібно використовувати абсолютні URL-адреси, вам слід вимагати, щоб поточний домен був вказаний вручну у файлі конфігурації та посилався на це значення замість заголовка Host. Наприклад, такий підхід усуне загрозу отруєння перезавантаженням пароля.

Перевірте заголовок хосту.

Якщо вам потрібно використовувати заголовок Host, переконайтеся, що ви його правильно перевірили. Це повинно включати перевірку його з білого списку дозволених доменів та відхилення або перенаправлення будь-яких запитів на нерозпізнані хости. Вам слід ознайомитися з документацією вашого фреймворку, щоб отримати вказівки щодо того, як це зробити. Наприклад, фреймворк Django надає ALLOWED_HOSTSопцію у файлі налаштувань. Цей підхід зменшить ваш вплив на атаки введення заголовка хосту.

Не підтримує заголовки заміни хосту.

Важливо також перевірити, чи не підтримуєте ви додаткові заголовки, які можуть бути використані для побудови цих атак, зокрема X-Forwarded-Host. Пам’ятайте, що вони можуть підтримуватися за замовчуванням.

Білий список дозволених доменів.

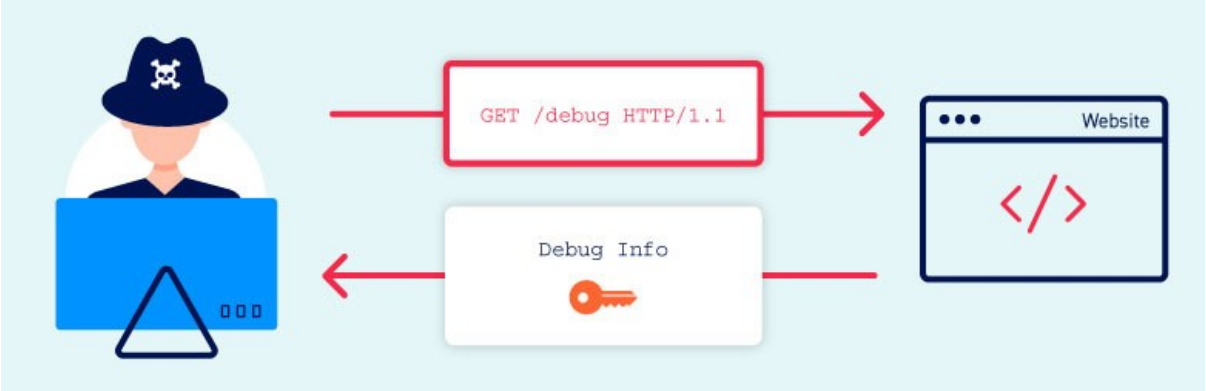
Щоб запобігти атакам на внутрішню інфраструктуру на основі маршрутизації, слід налаштувати балансування навантаження або будь-які зворотні проксі-сервери для пересилання запитів лише до білого списку дозволених доменів.

Будьте обережні з внутрішніми віртуальними хостами.

Використовуючи віртуальний хостинг, слід уникати розміщення лише внутрішніх веб-сайтів та програм на тому ж сервері, що і загальнодоступний вміст. В іншому випадку зловмисники можуть отримати доступ до внутрішніх доменів за допомогою маніпуляції заголовком хосту.

9. Вразливості розкриття інформації

Навчитися знаходити та використовувати розкриття інформації - життєво важлива навичка будь-якого тестувальника. Ви, швидше за все, стикаєтеся з ним регулярно, і, коли ви знаєте, як ефективно використовувати його, це може допомогти вам поліпшити ефективність тестування та дозволить вам знайти додаткові помилки високої серйозності.



Що таке розкриття інформації?

Розкриття інформації, також відоме як витік інформації, це коли веб-сайт ненавмисно розкриває конфіденційну інформацію своїм користувачам. Залежно від контексту, веб-сайти можуть передавати будь-яку інформацію потенційному зловмиснику, включаючи:

- Дані про інших користувачів, такі як імена користувачів або фінансова інформація
- Конфіденційні комерційні або ділові дані
- Технічна інформація про веб-сайт та його інфраструктуру

Небезпека витоку конфіденційних даних користувачів або бізнесу досить очевидна, але розкриття технічної інформації іноді може бути настільки ж серйозним. Хоча деяка частина цієї інформації буде обмеженою, вона потенційно може стати відправною точкою для виявлення додаткової поверхні атаки, яка може містити інші цікаві вразливості. Знання, які ви можете зібрати, можуть навіть надати відсутні фрагмент головоломки при спробі побудувати складні атаки високої тяжкості.

Іноді конфіденційна інформація може необережно потрапляти до користувачів, які просто переглядають веб-сайт у звичайному режимі. Однак частіше зловмиснику потрібно отримати інформацію про розкриття інформації, взаємодіючи з веб-сайтом несподіваними або зловмисними способами. Потім вони ретельно вивчать відповіді веб-сайту, щоб спробувати визначити цікаву поведінку.

Які приклади розкриття інформації?

Нижче наведено кілька основних прикладів розкриття інформації:

- Розкриття назв прихованих каталогів, їх структури та вмісту за допомогою robots.txt файлу або списку каталогів

- Надання доступу до файлів вихідного коду за допомогою тимчасових резервних копій
- Явне згадування імен таблиць та стовпців бази даних у повідомленнях про помилки
- Зайве викриття дуже конфіденційної інформації, наприклад даних кредитної картки
- Ключі API із жорстким кодуванням, IP-адреси, облікові дані бази даних тощо у вихідному коді
- Натякаючи на існування або відсутність ресурсів, імен користувачів тощо, через тонкі відмінності в поведінці додатків

Як виникають уразливості щодо розкриття інформації?

Вразливі місця щодо розкриття інформації можуть виникати незліченною кількістю різних способів, але їх можна загалом класифікувати наступним чином:

- **Не вдалося видалити внутрішній вміст із загальнодоступного**. Наприклад, коментарі розробників у розмітці іноді видно користувачам у виробничому середовищі.
- **Небезпечна конфігурація веб-сайту та супутніх технологій**. Наприклад, якщо не вимкнути функції налагодження та діагностики, іноді зловмисники можуть отримати корисні інструменти, які допоможуть їм отримати конфіденційну інформацію. Конфігурації за замовчуванням також можуть зробити веб-сайти вразливими, наприклад, відображаючи надмірно детальні повідомлення про помилки.
- **Неправильний дизайн та поведінка програми**. Наприклад, якщо веб-сайт повертає різні відповіді, коли виникають різні стани помилок, це також може дозволити зловмисникам перераховувати конфіденційні дані, такі як дійсні облікові дані користувача.

Який вплив мають вразливості щодо розкриття інформації?

Вразливі місця щодо розкриття інформації можуть мати як прямий, так і непрямий вплив, залежно від мети веб-сайту і, отже, якої інформації зловмисник може отримати. У деяких випадках сам факт розголошення конфіденційної інформації може мати великий вплив на постраждалі сторони. Наприклад, інтернет-магазин, що витікає дані кредитної картки своїх клієнтів, може мати серйозні наслідки.

З іншого боку, витік технічної інформації, наприклад, структури каталогів або використовуваних сторонніх платформ, може мати майже прямий вплив. Однак в чужих руках це може бути ключовою інформацією, необхідною для побудови будь-якої кількості інших подвигів. Тяжкість у цьому випадку залежить від того, що зловмисник може зробити з цією інформацією.

Як оцінити серйозність вразливостей при розкритті інформації.

Хоча кінцевий вплив може потенційно бути дуже серйозним, лише за певних обставин розкриття інформації є проблемою високої серйозності саме по собі. Під час тестування розкриття технічної інформації, зокрема, часто

представляє інтерес лише у тому випадку, якщо ви можете продемонструвати, як зловмисник може зробити з ним щось шкідливе.

Наприклад, відомості про те, що веб-сайт використовує певну фреймворкову версію, мають обмежене використання, якщо ця версія повністю виправлена. Однак ця інформація стає важливою, коли веб-сайт використовує стару версію, що містить відому вразливість. У цьому випадку здійснити руйнівну атаку може бути настільки просто, як застосувати публічно задокументований подвиг.

Важливо проявляти здоровий глузд, коли виявляєте, що відбувається витік потенційно конфіденційної інформації. Цілком імовірно, що незначні технічні деталі можна виявити різними способами на багатьох веб-сайтах, які ви перевіряєте. Отже, ваша основна увага повинна бути зосереджена на впливі та експлуатації витоку інформації, а не лише на розкритті інформації як окремої проблеми. Очевидний виняток з цього полягає в тому, що інформація, що просочилася, настільки чутлива, що сама по собі заслуговує на увагу.

Як запобігти уразливості при розкритті інформації.

Повністю запобігти розголошенню інформації досить складно через величезну різноманітність способів, якими це може відбутися. Однак є кілька загальних практик, яких ви можете дотримуватися, щоб мінімізувати ризик потрапляння подібних видів вразливості на ваші власні веб-сайти.

- Переконайтесь, що всі, хто бере участь у розробці веб-сайту, повністю знають, яку інформацію вважають конфіденційною. Іноді на перший погляд нешкідлива інформація може бути набагато кориснішою для зловмисника, ніж люди уявляють. Висвітлення цих небезпек може допомогти впевнитись, що ваша організація в цілому обробляє конфіденційну інформацію більш безпечно.
- Перевірте будь-який код на предмет можливого розкриття інформації як частини вашого контролю якості або процесів побудови. Автоматизація деяких пов’язаних завдань, наприклад видалення коментарів розробника, повинна бути відносно простою.
- Використовуйте загальні повідомлення про помилки якомога більше. Не надавайте зловмисникам підказки про поведінку додатків без потреби.
- Перевірте ще раз, що будь-які функції налагодження чи діагностики відключені у виробничому середовищі.
- Переконайтесь, що ви повністю розумієте параметри конфігурації та наслідки безпеки будь-якої сторонньої технології, яку ви впроваджуєте. Не поспішайте досліджувати та вимикати будь-які функції та налаштування, які вам насправді не потрібні.

Список використаних джерел

1. STUTTARD, Dafydd; PINTO, Marcus. The web application hacker's handbook: Finding and exploiting security flaws. John Wiley & Sons, 2011.
2. Kimminich B. Pwining OWASP Juice Shop. 2020. – 301 p.
3. Andrew Hoffman. Web Application Security. Published by O'Reilly Media, Inc. 2020. – 331 p.
4. PAIGE, Michael. The tangled web: A guide to securing modern web applications by michal zalewski. ACM SIGSOFT Software Engineering Notes, 2013, 38.4: 39-40.
5. CHRISTEY, Steve, et al. CWE/SANS top 25 most dangerous software errors. Common Weakness Enumeration, 2011.
6. SIMON, William L.; MITNICK, Kevin D. The Art of Deception: Controlling the Human Element of Security. Wiley, 2002.
7. YAWORSKI, Peter. Real-world bug hunting: a field guide to web hacking. No Starch Press, 2019.
8. CHRISTEY, Steve, et al. CWE/SANS top 25 most dangerous software errors. Common Weakness Enumeration, 2011.
9. DANISWARA, Rasendriya Revo; SASMITA, Gusti Made Arya; PRATAMA, I. P. A. E. The Testing for Information Gathering Using OWASP Testing Guide v4 (Case Study: Udayana University SIMAK-NG Application). JITTER: Jurnal Ilmiah Teknologi dan Komputer, 2020, 1.1.