1. ДОСЛІДЖЕННЯ ВРАЗЛИВОСТЕЙ ВЕБ-ЗАСТОСУНКІВ ТА МЕТОДІВ ЗАХИСТУ
   1. Аналіз загроз безпеки веб-застосунків

Безпека веб-застосунікв це розділ інформаційної безпеки який відповідає за вирішення питань безпеки веб-сайтів, веб-застосунків та веб-сервісів. На високому рівні безпека веб-застосунків базується на принципах безпеки прикладних програм які мають доступ до інтернету.

Більшість атак на веб-застосунки реалізуються шляхом міжсайтового скриптингу (XSS) і SQL-ін’єкцій які зазвичай можливі через недостатню професійність розробника і помилками зв’язаними з обробкою застосунком вхідних і вихідних даних. Зазвичай ці дві загрози стоять на початку усіх чартів які надають інформацію про найчастіші загрози які є наслідком погано написаного коду.

Відповідно до постачальника послуг безпеки Cenzic, топ загроз наведено у табл. 1.1.

Таблиця 1.1. – Топ загроз веб-застосунків від Cenzic [8]

|  |  |
| --- | --- |
| Відносна частота загрози | Назва загрози |
| 37% | Міжсайтовий скриптинг |
| 16% | SQL ін'єкції |
| 5% | Розкриття повного шляху у get запитах |
| 5% | Data breach (information disclosure) |
| 4% | Виконання довільного коду |
| 4% | Пошкодження пам'яті |
| 4% | Cross-site request forgery |
| 3% | Розкриття конфіденційної інформації |
| 3% | Довільне виконання файлів |
| 2% | Виконання локальних файлів серверу |
| 1% | Віддалене виконання файлів |
| 1% | Переповнення буферу |
| 15% | Інші, включаючи ін'єкції JS-коду |

* 1. Найпоширеніші загрози веб-застосунків за версією OWASP

Open Web Application Security Project (OWASP) – це відкритий проект забезпечення безпеки веб-застосунків. Товариство OWASP включає в себе корпорації і наукові заклади багатьох країн. OWASP працює над створенням статей, навчальних посібників, рекомендацій, документацій, інструментів і технологій, які зберігаються у відкритому доступі.

OWASP рекомендації щодо безпечної розробки коду описує деякі прийоми і методи які розробники можуть використовувати для створення захищених застосунків які забезпечують конфіденційність, цілісність і доступність інформації[7]:

* Валідація вхідних даних.
* Кодування вихідних даних відносно належного контексту.
* Автентифікація і менеджмент паролів.
* Менеджмент сеансів.
* Конроль доступу.
* Криптографічний захист.
* Обробка помилок і логування.
* Захист чутливих даних.
* Безпека зв’язку.
* Системна конфігурація.
* Безпека бази даних.
* Менеджмент файлів.
* Керування пам’яттю.
* Загальні прийоми кодування.

Один з найбільших вкладів OWASP вніс за рахунок проекту Top Ten Vulnerabilities – список найбільш частих 10 загроз веб-застосунків, зазвичай список оновлюється з періодом 3 роки[9].

Перелік вразливостей постійно оновлюється і на 2017 виглядає наступним чином:

* Ін'єкції (Injections).
* Недоліки системи аутентифікації і зберігання сесій (Broken Authentication and Session Management).
* Незахищеність критичних даних ([Sensitive Data Exposure](https://owasp.org/www-project-top-ten/2017/A3_2017-Sensitive_Data_Exposure)).
* Впровадження зовнішніх XML-сутностей (XXE).
* Порушення контролю доступа ([Broken Access Control](https://owasp.org/www-project-top-ten/2017/A5_2017-Broken_Access_Control).).
* Похибки в конфігуруванні ([Security Misconfiguration](https://owasp.org/www-project-top-ten/2017/A6_2017-Security_Misconfiguration)).
* Міжсайтовий скриптинг - XSS (Cross Site Scripting).
* Небезпечна десериалізація (Insecure Deserialization).
* Використання компонентів з відомими вразливостями ([Using Components with Known Vulnerabilities](https://owasp.org/www-project-top-ten/2017/A9_2017-Using_Components_with_Known_Vulnerabilities)).
* Недостатнє логування та моніторінг.

Окрім цих 10 найбільш популярних вразливостей веб-застосунків також дуже поширені:

* Небезпечні прямі посилання на об'єкти (Insecure Direct Object References).
* Відсутність функції контроля доступу (Missing Function Level Access Control).
* Межсайтова підробка запитів (Cross-Site Request Forgery, CSRF/XSRF).
* Неперевірені переадресації та пересилання (Unvalidated Redirects and Forwards).
  + 1. Ін'єкції (Injections)

Зазвичай в веб-додатках для зберігання даних використовуються бази даних, звернення до яких найчастіше відбувається за допомогою спеціальної мови SQL. Веб-додатки використовують SQL-запити для того, щоб отримувати, додавати, видаляти або змінювати дані в базі даних, наприклад, при редагуванні своїх даних в особистому кабінеті. При недостатній захищеності цих даних, зловмисник може впровадити код що містить SQL-запит.

Даний вид атаки називається SQL-ін'єкція і є одним з найпоширеніших у наш час. Це дуже небезпечна уразливість, яка може дозволити зловмиснику отримати доступ до бази даних і додавати змінювати та видаляти інформацію. Наприклад, змінити баланс свого рахунку в банку, переглянути конфіденційну інформацію інших користувачів.

Приклад класичної SQL ін'єкції швидше характерний саме для додатків Web Forms. Від атак допомагає захиститися використання параметрів в якості значень запиту. Entity Framework прикриває деякі уразливості. Для того, щоб в MVC або EF додатку спрацювала SQL ін'єкція потрібно дуже постаратись. Однак це можливо якщо ви виконуєте SQL код за допомогою ExecuteQuery методу або викликаєте погано написані збережені процедури.

Незважаючи на те, що ORM (Object-relational mapping) дозволяє уникнути SQL Injection (за винятком наведених вище прикладів), рекомендується обмежувати значення, які можуть приймати поля моделі, а значить і форми. Обмежити можливі значення можна за допомогою валідаторів.

* + 1. Недоліки системи аутентифікації і зберігання сесій (Broken Authentication and Session Management)

Для того, щоб відрізнити одного користувача від іншого, у веб додатку використовуються сесійні куки. Після того, як користувач виконав вхід на сайт, наприклад, за допомогою логіна і пароля, при встановленій опції в браузері зберігається спеціальний ідентифікатор, який потім надається веб-додатком при кожному зверненні. Саме таким чином веб-додаток розуміє, що Ви це Ви.

Суть даної атаки полягає в тому, що зловмисник намагається отримати цей унікальний ідентифікатор і скористатися ним для входу на сайт від вашого імені і, якщо на сайті немає додаткових перевірок на цей унікальний ідентифікатор, наприклад, ip-адреса первинного звернення або опція, яка забороняє вести одночасно дві сесії з одного облікового записа, то отримавши цей унікальний ідентифікатор, користувач зможе скористатися всіма можливостями вашого профілю в веб-застосунку.

Функції додатків, які відповідають за аутентифікацію і управління сесіями, найчастіше застосовуються неправильно, наслідком чого стане компрометація паролів, ключів, сеансових токенів, і навіть можливість повністю перехопити сеанс користувача.

* + 1. Незахищеність критичних даних ([**Sensitive Data Exposure**](https://owasp.org/www-project-top-ten/2017/A3_2017-Sensitive_Data_Exposure))

Багато веб-додатків та API можуть некоректно зберігати і обробляти важливу інформацію персональних даних. Зловмисники можуть вкрасти або змінити таку інформацію, що може стати основою для серйозних фінансових або репутаційних втрат. Інформація повинна зберігатися належним чином, а також повинна бути захищена при передачі по каналах зв'язку.

Деякі веб-додатки не вважають за потрібне захищати конфіденційні та персональні дані. Тому зловмисник може отримати до них доступ і впливати на них. Одним з таких прикладів є використання протоколу HTTP для передачі конфіденційних даних. Вся проблема полягає в тому, що при передачі даних через протокол HTTP не відбувається їх шифрування. І використовуючи звичайний сниффер, зловмисник може перехопити ці дані.

Щоб уникнути такої ситуації необхідно використовувати протокол HTTPS.

Так само використання HTTPS протоколу для передачі конфіденційних даних надає підтвердження вашого права на інформацію за допомогою SSL сертифікату. Так як його можна перевірити на валідність або термін дії.

Так само наявність шифрування захищає конфіденційні дані від їх крадіжки, навіть якщо зловмисник отримав доступ до бази даних. Так, для паролів, зазвичай, використовується хеш-функція, і відновити його з хешу дуже складно, а його перевірка відбувається шляхом порівняння хешу, введеного пароля і хешу який зберігається в базі даних.

* + 1. Міжсайтовий скриптинг - XSS (Cross Site Scripting)

Міжсайтовий скриптинг - ще одна помилка валідації призначених для користувача даних, яка дозволяє передати JavaScript код на виконання в браузер користувача. Атаки такого роду часто також називають HTML-ін'єкціями, адже механізм їх впровадження дуже схожий з SQL-ін'єкціями, але на відміну від останніх, впроваджуваний код виповнюється в браузері користувача.

Чим це небезпечно? По-перше, зловмисник може викрасти унікальний ідентифікатор (сесійний cookie). І скористатися ним, як описано в розділі 1.2.2 Broken Authentication and Session Management.

По-друге, зловмисник може викрати дані, які вводяться у форми.

По-третє, зловмисник може змінити дані, які виводяться на сторінці. Додати вірусні посилання або замінити, наприклад, банківські реквізити.

XSS вразливість виникає тоді, коли дані, прийняті від користувача, виводяться в браузер без належної фільтрації. Наприклад, програмний код рекламного банера може містити скрипт для перехоплення призначених для користувача даних або навіть прозоре перенаправлення на інші сайти.

Щоб запобігти XSS атаці потрібно валідувати та кодувати вхідні данні та санітизувати на виході.

В першу чергу, необхідно виявити всі потоки даних, цілісність або автентичність яких не контролюється всередині розглянутого компонента веб-додатку. Під компонентом веб-додатку, як правило (хоча і не завжди), маються на увазі елементи його серверної або клієнтської частини, що виконуються в рамках одного процесу ОС.

Як можна ближче до місця появи таких даних в компоненті, необхідно забезпечити їх приведення до очікуваних типів. Вся подальша робота з даними всередині компонента повинна здійснюватися тільки через створені на основі вхідних даних об'єкти. Важливо пам'ятати, що основним принципом етапу типізації є якомога менша кількість об'єктів строкового типу на його виході.URL, адреси електронної пошти, дата, часта інше по стандартизації повинні представляти із себе об'єкти конкретних типів, відмінних від строкового. У вигляді рядків повинні бути представлені тільки ті дані, які насправді є рядком, тобто які дійсно можуть містити довільний повноалфавітний текст. Одразу після типізації по стандартизації, семантику отриманих об'єктів необхідно перевірити на відповідність функціоналу компонента. Наприклад, для цілочисельних типів або дати та часу - це буде перевірка діапазону, для строкових, в більшості випадків, буде достатньо перевірки на відповідність регулярними виразами, а для об'єктів більш комплексних типів необхідно реалізовувати перевірку семантики кожного з його полів і властивостей. Будь-які перевірки, пов'язані з валідацією завжди повинні здійснюватися на основі принципу білого списку, тобто семантика даних повинна відповідати дозволеним критеріям, а не навпаки. Метою даного етапу є отримання гарантії того, що всі дані всередині компонента будуть відповідати реалізованому в ньому функціоналу і не зможуть його порушити.

У всіх місцях, де валідовані і типізовані нами об'єкти залишають межі компонента (або там, де на їх основі формуються вихідні дані), необхідно забезпечити їх приведення до виду, безпечного для приймаючої сторони. Як правило, це досягається шляхом видалення з них небезпечних елементів (фільтрації) або ж їх перетворення до безпечних еквівалентів (екранування). Санітизацію необхідно реалізовувати адекватно до того місця, в яке потраплять дані в кінцевому підсумку.

* + 1. Небезпечні прямі посилання на об'єкти (Insecure Direct Object References)

Ця вразливість є наслідком того, що доступ до конфіденційних даних користувачів здійснюється не з прав доступу до даного об'єкта, а за будь-яким значенням, наприклад ідентифікатором або унікальним ім’ям. В такому випадку маючи сторінку виду: «testsite.ru/read\_user\_message?userName=userName» зловмиснику досить перебирати різні комбінації і підставляючи їх в параметр userName, він зможе читати повідомлення, які належать користувачам.

Пропозиція полягає у використанні хеша для заміни прямого ідентифікатора. Цей хеш солиться зі значенням, визначеним на рівні програми, щоб підтримувати топологію, в якій додаток розгортається в режимі декількох екземплярів. Використання хешу дозволяє наступні властивості:

* Не потрібно підтримувати таблицю зіставлення (реальний ідентифікатор проти ідентифікатора інтерфейсу) у сеансі користувача або кеші рівня програми.
* Ускладнює створення колекції значень перерахування, оскільки навіть якщо зловмисник може вгадати хеш-алгоритм за розміром ідентифікатора, він не може відтворити значення через сіль, яка не прив’язана до прихованого значення.
  + 1. Похибки в конфігуруванні ([**Security Misconfiguration**](https://owasp.org/www-project-top-ten/2017/A6_2017-Security_Misconfiguration))

Безпечність будь-якої системи завжди залежить від правильного налаштування безпеки, веб-додатки не є виключенням.

Неправильна конфігурація безпеки є найбільш часто проблемою. Зазвичай це результат небезпечних конфігурацій за замовчуванням, неповних або спеціальних конфігурацій, відкритого хмарного сховища, неправильно налаштованих заголовків HTTP та детальних повідомлень про помилки, що містять конфіденційну інформацію.

Налаштування компонентів сервера за замовчуванням найчастіше небезпечні і відкривають можливості до атак. Наприклад, крадіжка сесійного cookie через JavaScript при XSS-атакці стає можливою завдяки виключеною за замовчуванням налаштування cookie\_http only.

Наступна помилка в конфігурації – це використання даних, для доступу до будь-яких елементів веб-додатків за замовчуванням. Одним з найяскравіших є злом корейського облікового записа facebook за допомогою логін / пароля адміністратора admin / admin.

Висновком цієї атаки є те, що необхідно підтримувати актуальність програмного забезпечення, так як нові вразливості знаходіться кожен день.

* 1. Аналіз методів хешування паролів

Хешування паролів – метод, що дозволяє користувачам запам`ятовувати не дуже довгий ключ, а деякий осмислений вираз, слово чи послідовність символів, що називається паролем. Дійсно, при розробці будь-якого криптоалгоритмами слід враховувати, що в половині випадків кінцевим користувачем системи є людина, а не автоматична система. Це ставить питання про те, зручно, і взагалі чи реально людині запам'ятати 128-бітний ключ. Насправді межа запам'ятовуваності лежить на кордоні 8-12 подібних символів, а, отже, якщо ми будемо примушувати користувача оперувати саме ключем, тим самим ми практично змусимо його до запису ключа на будь-якому листку паперу або електронному носії, наприклад, в текстовому файлі. Це, звичайно, різко знижує захищеність системи. Для вирішення цієї проблеми були розроблені методи, які перетворюють осмислений рядок довільної довжини – пароль, у вказаний ключ заздалегідь заданої довжини. У переважній більшості випадків для цієї операції використовуються так звані хеш-функції. Хеш-функцією в даному випадку називається таке математичне або алгоритмічне перетворення заданого блоку даних, яке володіє наступними властивостями:

* хеш-функція має нескінченну область визначення;
* хеш-функція має кінцеву область значень;
* вона незворотня;
* зміна вхідного потоку інформації на один біт змінює близько половини всіх біт вихідного потоку, тобто результату хеш-функції.

Ці властивості дозволяють подавати на вхід хеш-функції паролі, тобто текстові рядки довільної довжини на будь-якою національною мовою і, обмеживши область значень функції діапазоном 0 .. 2N-1, де N - довжина ключа в бітах, отримувати на виході досить рівномірно розподілені по області значення блоки інформації – ключі.